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:**

- 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:00401043j
.text:0040104E 8B C6                        mov     eax, esi
.text:00401050 5E                       pop     esi
.text:00401051 C2 04 00                         retn    4
.text:00401051                      ; ---------------------------------------------------------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 20 00 02 00 04
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
```

```
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
```

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

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

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRILGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [4]:

```
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
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 [7]:

```
source = 'D:/microsoft_malware/train'
files = os.listdir(source)

for file in files[10000:]:
        os.remove(os.path.join(source,file))
```

In [10]:

```
# filename = 'D:/Tester/Train/'
# readfile = open(filename, "r")
# for line in readfile:
#     line.split(".")
#     print(line[0])
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
<ipython-input-10-e3bac2bb177c> in <module>
      1 filename = 'D:/Tester/Train/'
----> 2 readfile = open(filename, "r")
      3 for line in readfile:
      4     line.split(".")
      5     print(line[0])

PermissionError: [Errno 13] Permission denied: 'D:/Tester/Train/'
```

In [14]:

```
# files = os.listdir('D:/Tester/Train')

# for file in files:
#     file.split('.')
#     print (file)
```

```
0A32eTdBKayjCWhZqDOQ.asm
0A32eTdBKayjCWhZqDOQ.bytes
0ACDbR5M3ZhBJajygTuf.asm
0ACDbR5M3ZhBJajygTuf.bytes
0AguvpOCcaf2myVDYFGb.asm
0AguvpOCcaf2myVDYFGb.bytes
0aklgwhWHYm1dzsNqBFx.bytes
0AnoOZDNbPXlr2MRBSCJ.asm
0AnoOZDNbPXlr2MRBSCJ.bytes
0ASH2csN7k8jZyoRaqtn.asm
0aSTGBVRXeJhx5OcpsgC.asm
```

```
0aSTGBVRXeJnx5OcpsgC.bytes
0aU7XWsr8RtN94jvo3lG.bytes
0AV6MPlrTWG4fYI7NBtQ.asm
0AV6MPlrTWG4fYI7NBtQ.bytes
```

In [ ]:

```
```

In [ ]:

```
# #separating byte files and asm files

# source = 'train'
# destination = 'byteFiles'

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

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

In [ ]:

```
```

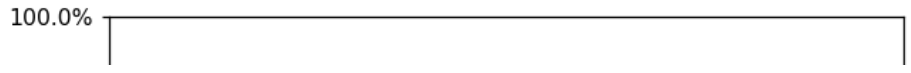
In [19]:

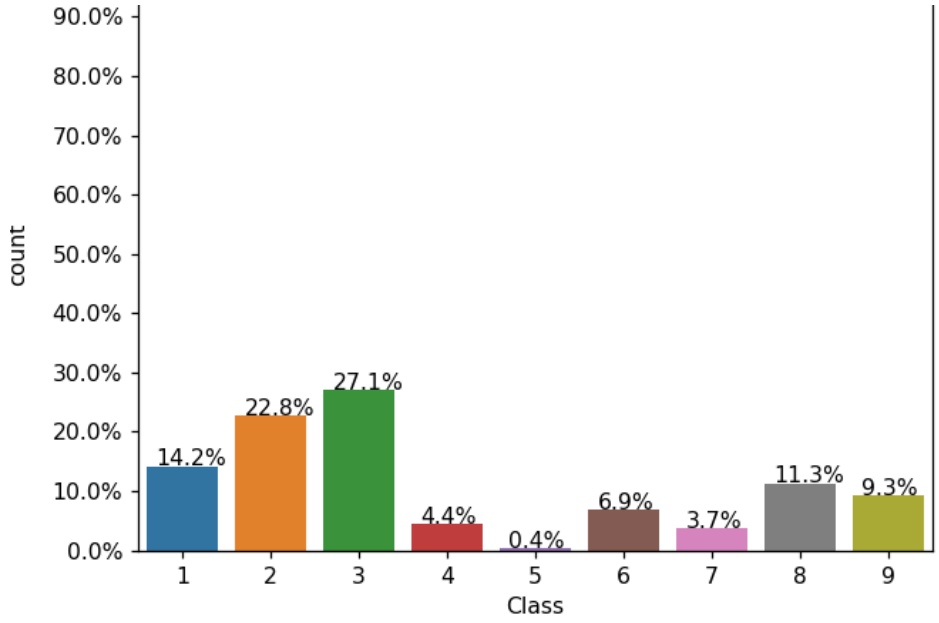
```
#separating byte files and asm files
source = 'D:/microsoft_malware/train/byteFiles/asmFiles/'
destination = 'D:/microsoft_malware/train/byteFiles/byteFiles/'

# 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):
    os.makedirs(destination)

data_files = os.listdir(source)

for f in data_files:
    if(f.endswith('bytes')):
        shutil.move(source+f,destination)
```

3.1. Distribution of malware classes in whole data set

In [2]:

```
Y=pd.read_csv("D:/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 [3]:

```
#file sizes of byte files

files=os.listdir('D:/microsoft_malware/train/byteFiles/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('D:/microsoft_malware/train/byteFiles/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  01azqd4InC7m9JpocGv5  5.012695      9
1  01IsoiSMh5gxyDYTl4CB  6.556152      2
2  01jsnpXSAlgw6aPeDxrU  4.602051      9
3  01kcPWA9K2BOxQeS5Rju  0.679688      1
4  01SuzwMJEIXsK7A8dQbl  0.438965      8
```

In [4]:

```
data_size_byte.shape
```

Out[4]:

```
(5000, 3)
```

In [0]:

```
# #file sizes of byte files

# files=os.listdir('D:/microsoft_malware/train/byteFiles/byteFiles')
```

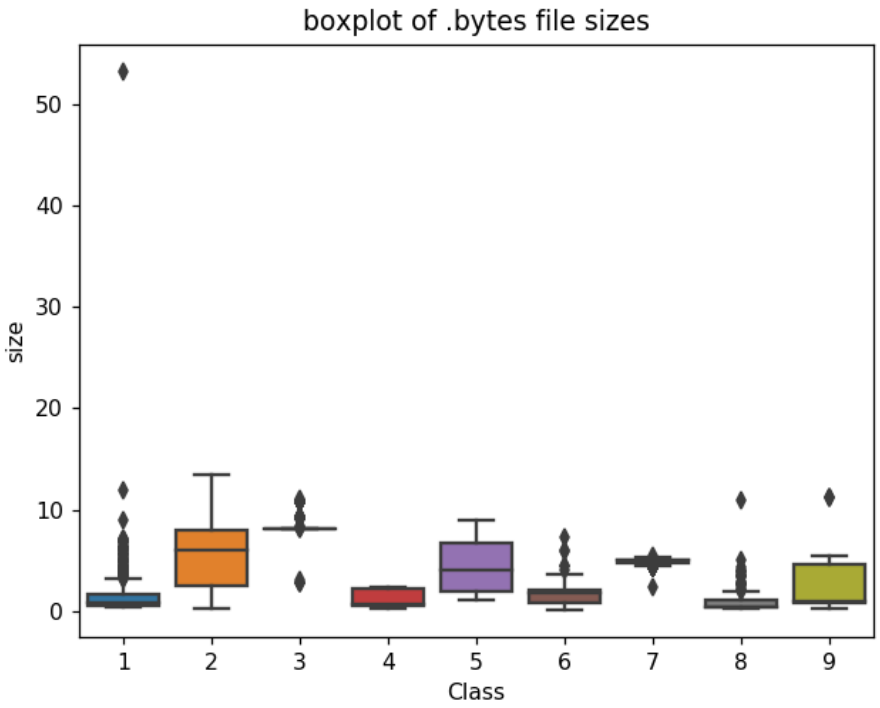
```
# filenames=Y['Id'].tolist()
# class_y=Y['Class'].tolist()
# class_bytes=[]
# sizebytes=[]
# fnames=[]
# for file in files:
#     # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
#     # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,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

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

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

In [25]:

```
#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 [0]:

```
#removal of addres from byte files
# contents of .byte files
# ----------------
```

```
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000

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

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


#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")
for file in files:
    filenames2.append(f)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i)+",")
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [ ]:

```

```

In [5]:

```
byte_features=pd.read_csv("D:/result.csv")
print (byte_features.head())
```

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

     8  ...   f7   f8   f9   fa   fb   fc   fd   fe   ff   ??
0  2965  ...  2804  3687  3101  3211  3097  2758  3099  2759  5753  1824
```

```
1  9291  ...   451  6536   439   281   302  7639   518  17001  54902   8588
2  9107  ...  2325  2358  2242  2885  2863  2471  2786   2680  49144    468
3  1078  ...   478   873   485   462   516  1133   471    761   7998  13940
4   422  ...   847   947   350   209   239   653   221    242   2199   9008

[5 rows x 258 columns]
```

In [ ]:

```
```

In [6]:

```
final_bytes = pd.concat([data_size_byte,byte_features], axis=1)
```

In [7]:

```
final_bytes = final_bytes.dropna(axis=0)
```

In [8]:

```
final_bytes.shape
```

Out[8]:

```
(5000, 261)
```

In [9]:

```
final_bytes.head()
```

Out[9]:

	ID	size	Class	ID	0	1	2	3	4	5	...	f7	f8
0	01azqd4InC7m9JpocGv5	5.012695	9.0	01azqd4InC7m9JpocGv5.txt	601905	3905	2816	3832	3345	3242	...	2804	3687
1	01IsoiSMh5gxyDYTI4CB	6.556152	2.0	01IsoiSMh5gxyDYTI4CB.txt	39755	8337	7249	7186	8663	6844	...	451	6536
2	01jsnpXSAlgw6aPeDxrU	4.602051	9.0	01jsnpXSAlgw6aPeDxrU.txt	93506	9542	2568	2438	8925	9330	...	2325	2358
3	01kcPWA9K2BOxQeS5Rju	0.679688	1.0	01kcPWA9K2BOxQeS5Rju.txt	21091	1213	726	817	1257	625	...	478	873
4	01SuzwMJEIXsK7A8dQbl	0.438965	8.0	01SuzwMJEIXsK7A8dQbl.txt	19764	710	302	433	559	410	...	847	947

5 rows × 261 columns

In [10]:

```
final_bytes = final_bytes.loc[:,~final_bytes.columns.duplicated()]
```

In [11]:

```
final_bytes.head()
```

Out[11]:

	ID	size	Class	0	1	2	3	4	5	6	...	f7	f8	f9	fa	fb	f
0	01azqd4InC7m9JpocGv5	5.012695	9.0	601905	3905	2816	3832	3345	3242	3650	...	2804	3687	3101	3211	3097	275
1	01IsoiSMh5gxyDYTI4CB	6.556152	2.0	39755	8337	7249	7186	8663	6844	8420	...	451	6536	439	281	302	763
2	01jsnpXSAlgw6aPeDxrU	4.602051	9.0	93506	9542	2568	2438	8925	9330	9007	...	2325	2358	2242	2885	2863	247
3	01kcPWA9K2BOxQeS5Rju	0.679688	1.0	21091	1213	726	817	1257	625	550	...	478	873	485	462	516	113
4	01SuzwMJEIXsK7A8dQbl	0.438965	8.0	19764	710	302	433	559	410	262	...	847	947	350	209	239	65

5 rows × 260 columns

In [12]:

```
# data = final_bytes[['size','Class']]
```

In [13]:

```
# https://stackoverflow.com/a/29651514
```

```
# 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(result)
```

In [14]:

```
final_bytes = normalize(final_bytes)
```

In [15]:

```
final_bytes.head()
```

Out[15]:

	ID	size	Class	0	1	2	3	4	5	6	...	f7
0	01azqd4InC7m9JpocGv5	0.091636	9.0	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	...	0.012546
1	01IsoiSMh5gxyDYTl4CB	0.120671	2.0	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	...	0.002018
2	01jsnpXSAlgw6aPeDxrU	0.083910	9.0	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	...	0.010402
3	01kcPWA9K2BOxQeS5Rju	0.010123	1.0	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	...	0.002139
4	01SuzwMJEIXsK7A8dQbl	0.005594	8.0	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	...	0.003790

5 rows × 260 columns

In [16]:

```
bytes_y = final_bytes['Class']
```

In [17]:

```
type(final_bytes)
```

Out[17]:

pandas.core.frame.DataFrame

In [18]:

```
final_bytes.shape
```

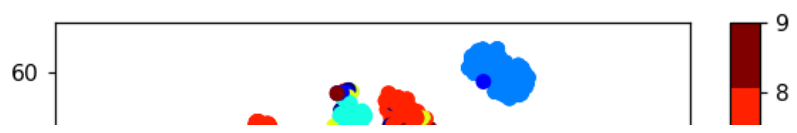
Out[18]:

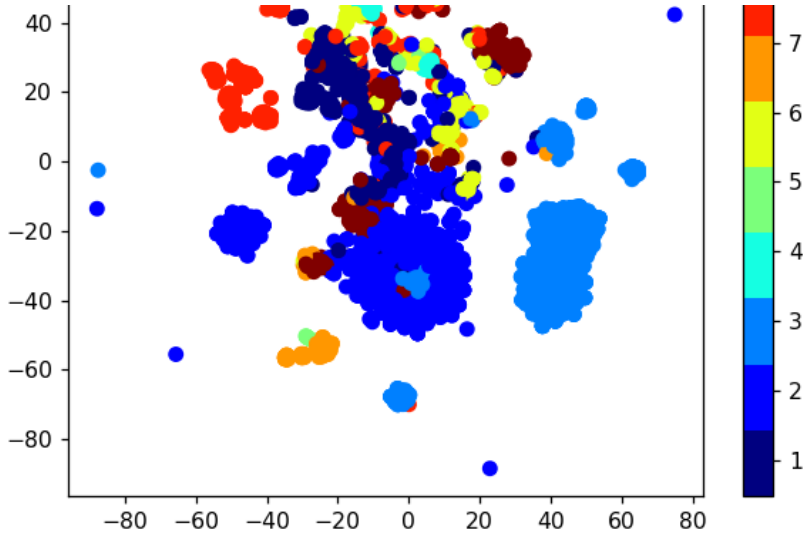
(5000, 260)

3.2.4 Multivariate Analysis

In [121]:

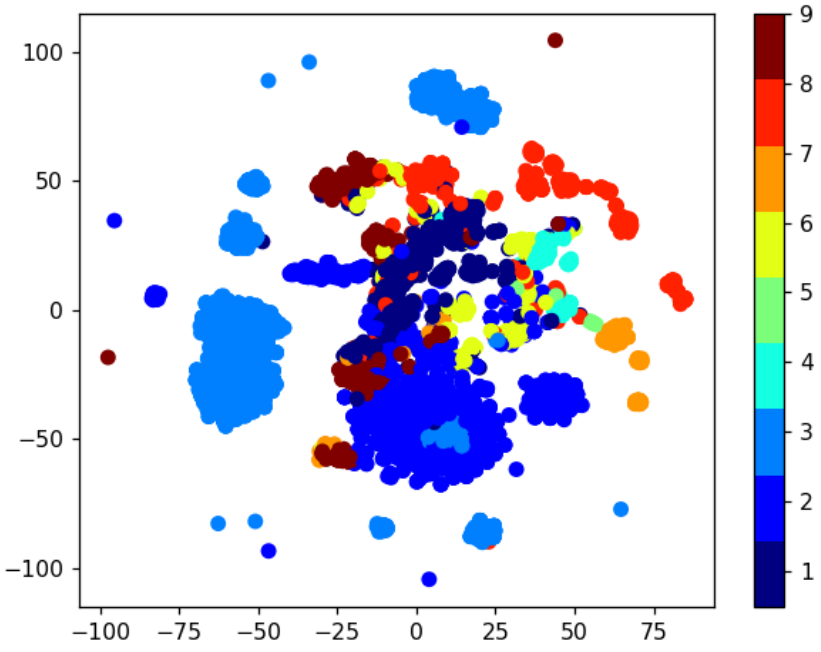
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
final_bytes = xtsne.fit_transform(final_bytes.drop(['ID','Class'], axis=1))
vis_x = final_bytes[:, 0]
vis_y = final_bytes[:, 1]
plt.scatter(vis_x, vis_y, c=bytes_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))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





In [136]:

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



Train Test split

In [151]:

```
bytes_y = final_bytes['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(final_bytes.drop(['ID','Class'], axis=1), bytes_y,stratify=bytes_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)
```

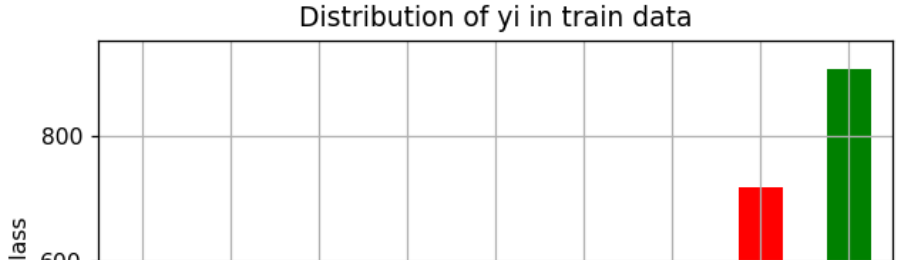
In [152]:

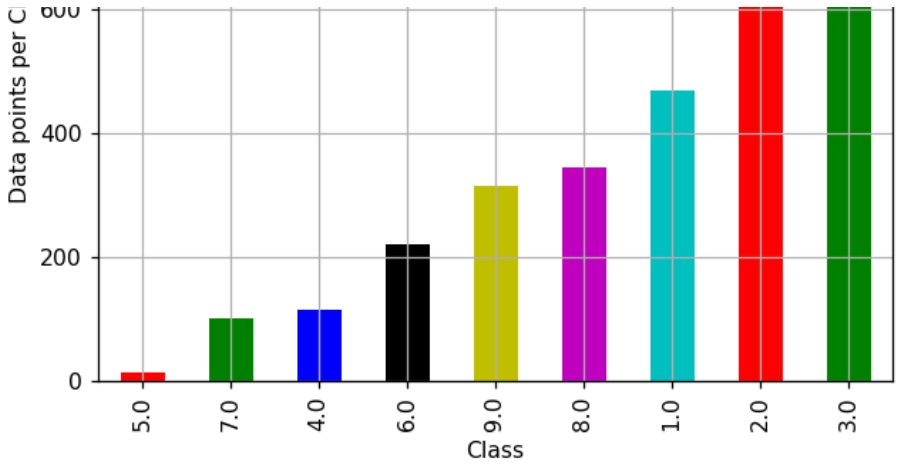
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
Number of data points in train data: 3200
Number of data points in test data: 1000
Number of data points in cross validation data: 800
```

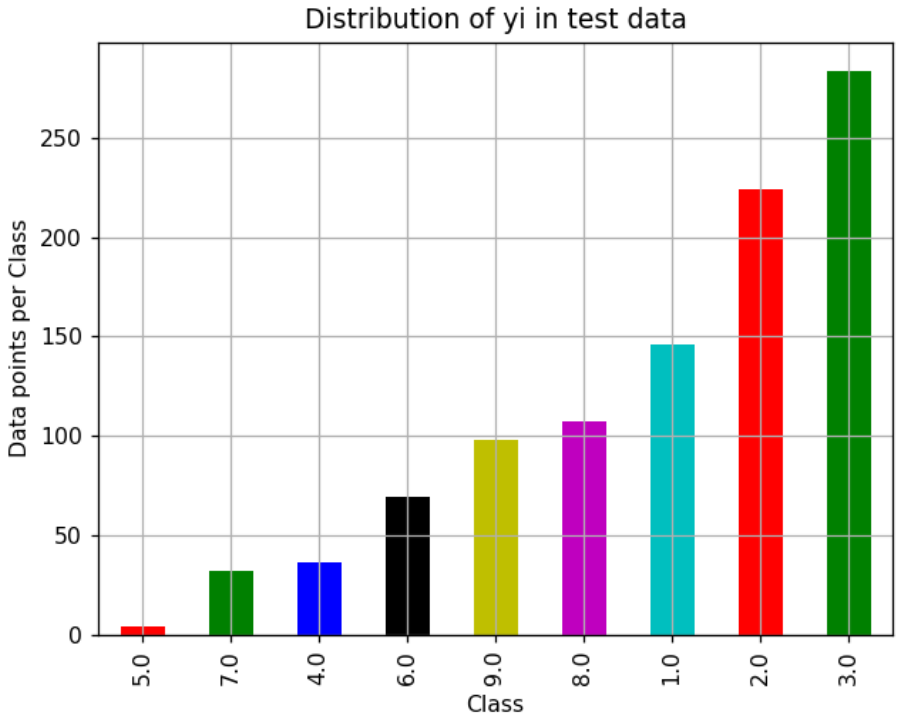
In [156]:

```
# 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_values()
test_class_distribution = y_test.value_counts().sort_values()
cv_class_distribution = y_cv.value_counts().sort_values()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_class_distribution
.values[i]/y_train.shape[0]*100), 3), '%)')


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_test.shape[0]*100), 3), '%)')

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.val
ues[i]/y_cv.shape[0]*100), 3), '%)')
```

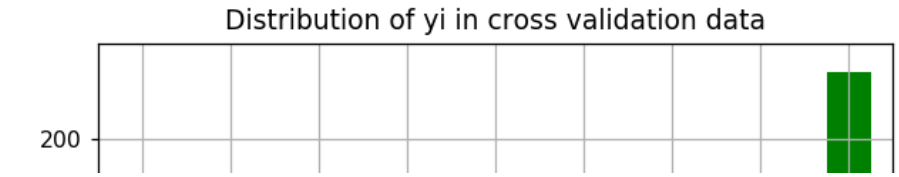


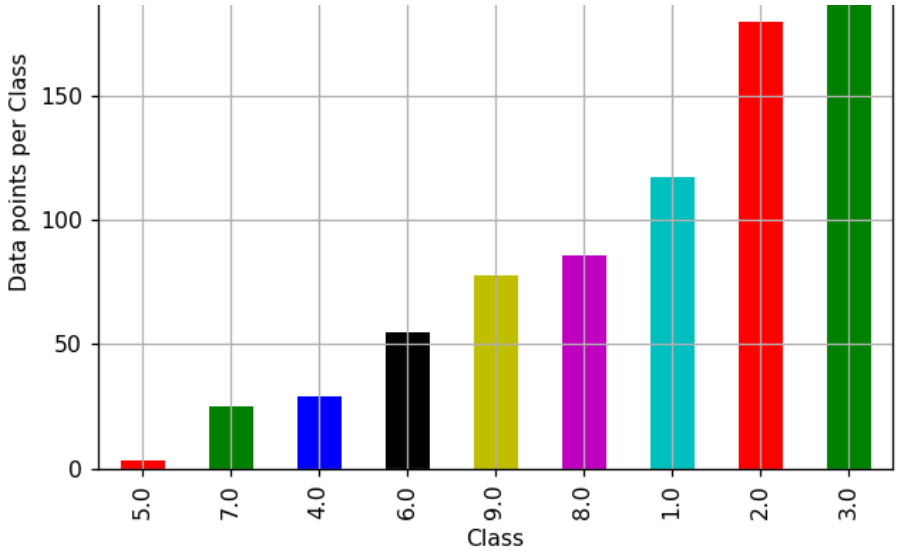


Number of data points in class 9 : 910 (28.438 %)
Number of data points in class 8 : 718 (22.438 %)
Number of data points in class 7 : 467 (14.594 %)
Number of data points in class 6 : 344 (10.75 %)
Number of data points in class 5 : 314 (9.812 %)
Number of data points in class 4 : 219 (6.844 %)
Number of data points in class 3 : 115 (3.594 %)
Number of data points in class 2 : 101 (3.156 %)
Number of data points in class 1 : 12 (0.375 %)



Number of data points in class 9 : 284 (28.4 %)
Number of data points in class 8 : 224 (22.4 %)
Number of data points in class 7 : 146 (14.6 %)
Number of data points in class 6 : 107 (10.7 %)
Number of data points in class 5 : 98 (9.8 %)
Number of data points in class 4 : 69 (6.9 %)
Number of data points in class 3 : 36 (3.6 %)
Number of data points in class 2 : 32 (3.2 %)
Number of data points in class 1 : 4 (0.4 %)





Number of data points in class 9 : 227 (28.375 %)
Number of data points in class 8 : 180 (22.5 %)
Number of data points in class 7 : 117 (14.625 %)
Number of data points in class 6 : 86 (10.75 %)
Number of data points in class 5 : 78 (9.75 %)
Number of data points in class 4 : 55 (6.875 %)
Number of data points in class 3 : 29 (3.625 %)
Number of data points in class 2 : 25 (3.125 %)
Number of data points in class 1 : 3 (0.375 %)

In [157]:

```
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

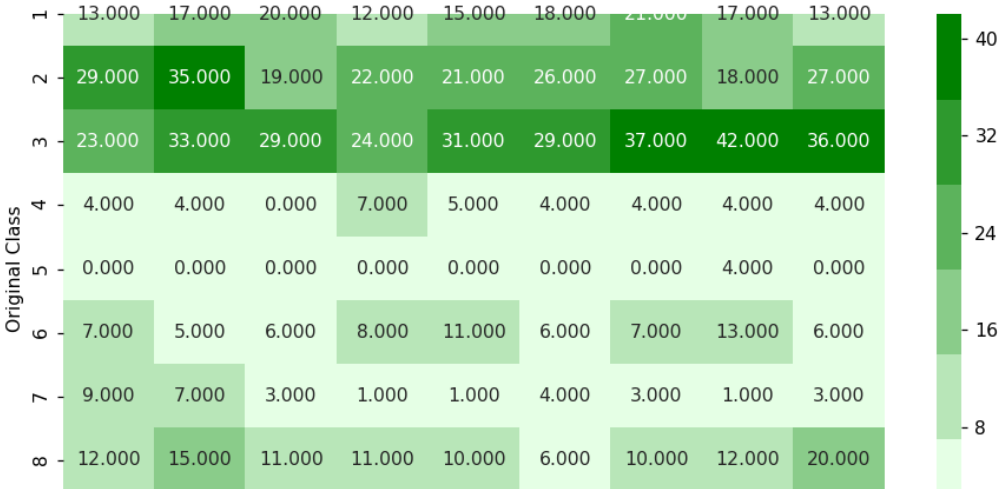
4.1. Machine Leaning Models on bytes files

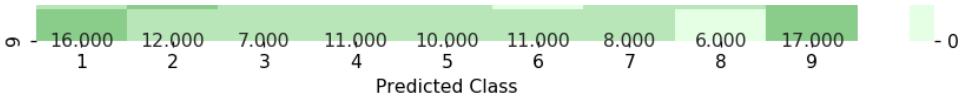
4.1.1. Random Model

In [158]:

```
# 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 2.4853988056359118
Log loss on Test Data using Random Model 2.4917828513210947
Number of misclassified points  87.8
---------------------------------------------------- Confusion matrix -----------------------------------------------------
```

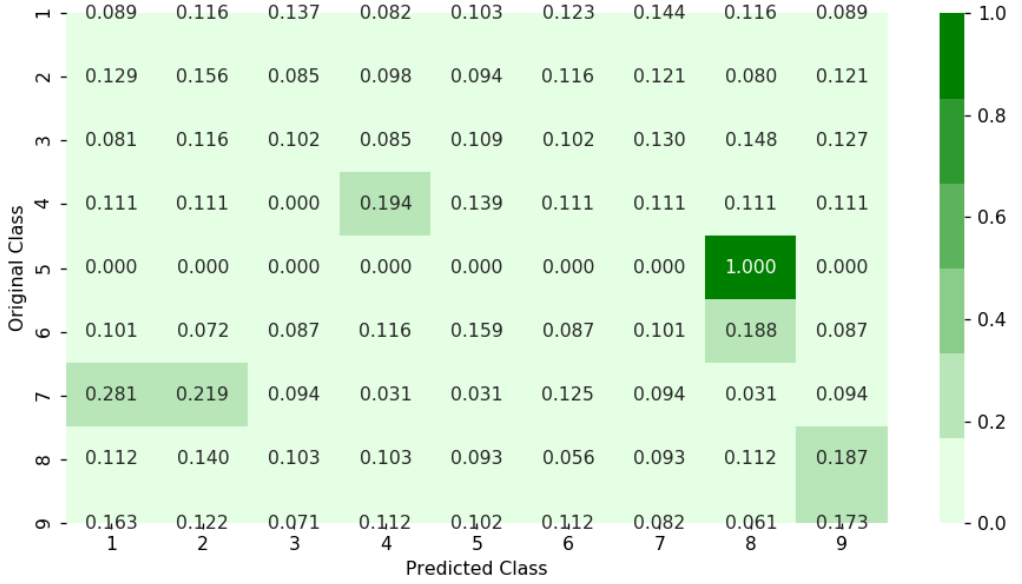




---------- Precision matrix ----------

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.]

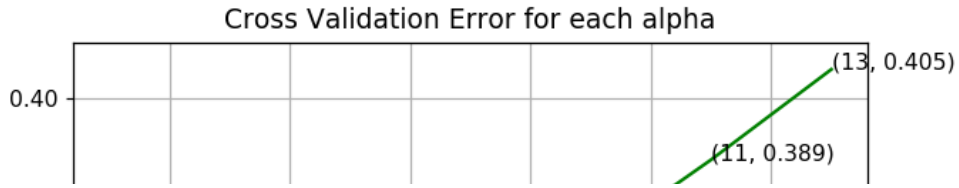
4.1.2. K Nearest Neighbour Classification

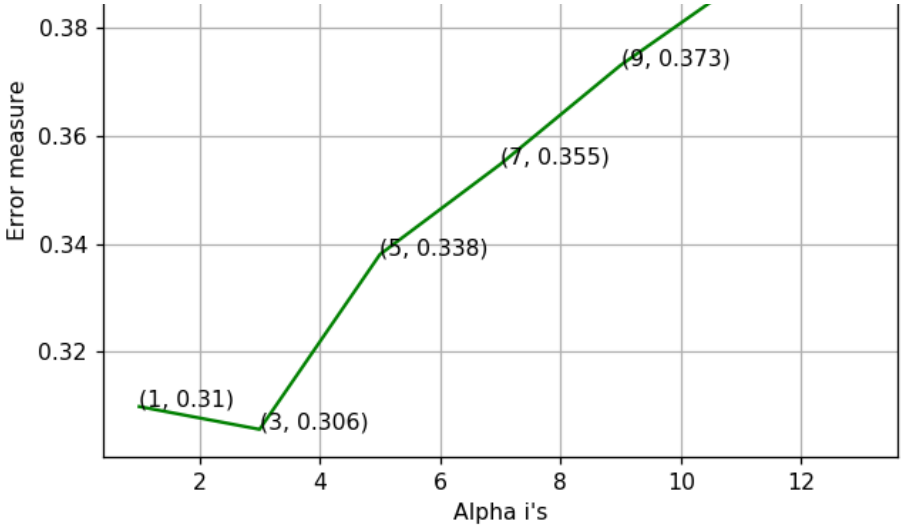
In [159]:

```
# 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(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-example-1/
#--------------------------------------


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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for k =  1 is 0.30993591982088575
log_loss for k =  3 is 0.30572813568906876
log_loss for k =  5 is 0.3380180961994511
log_loss for k =  7 is 0.35465643962379245
log_loss for k =  9 is 0.3729498618392467
log_loss for k =  11 is 0.388529421692902
log_loss for k =  13 is 0.40522853359428146
```

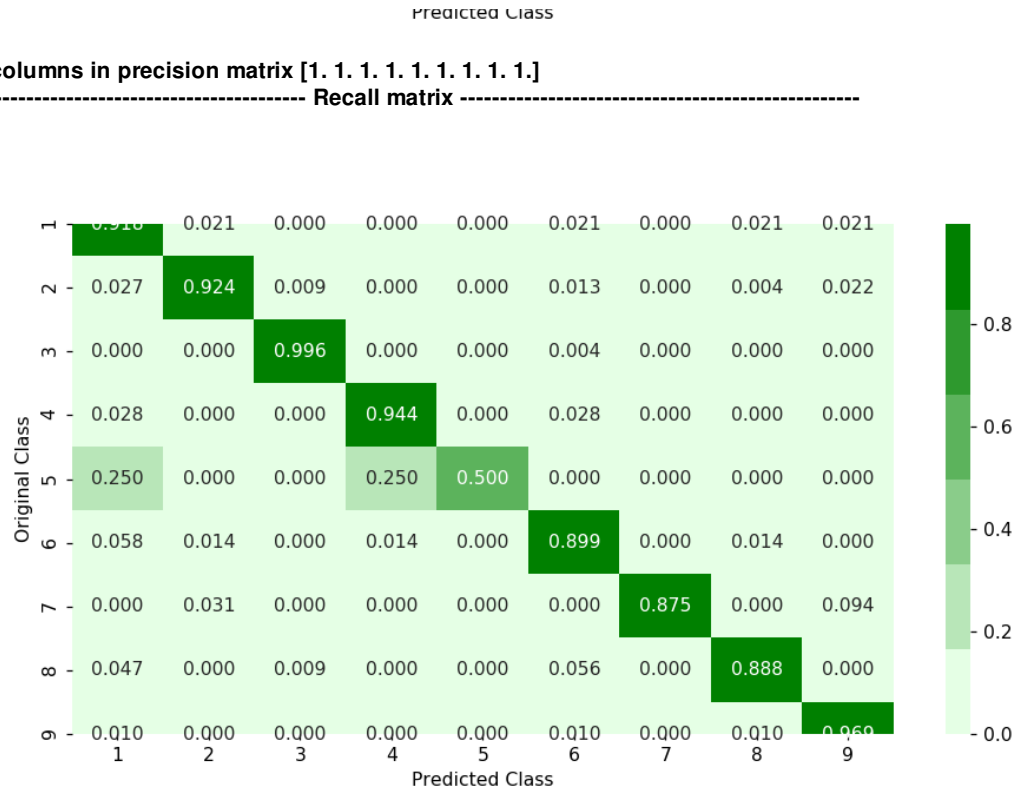




For values of best alpha = 3 The train log loss is: 0.14179244027526627
For values of best alpha = 3 The cross validation log loss is: 0.30572813568906876
For values of best alpha = 3 The test log loss is: 0.2664233946373553
Number of misclassified points 6.0

---------- 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.]

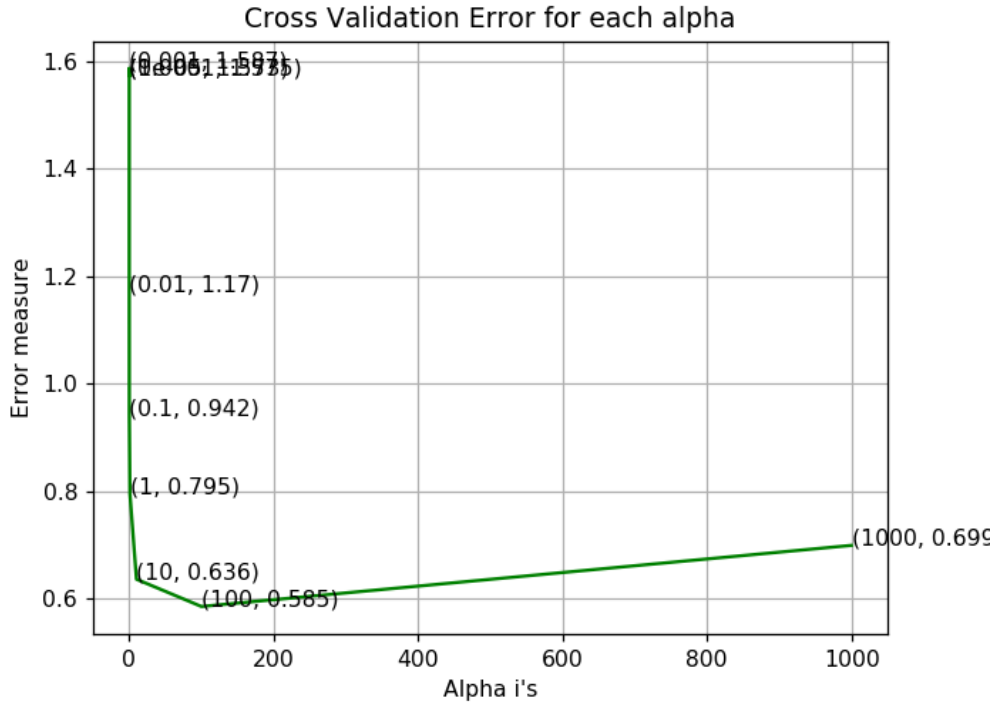
4.1.3. Logistic Regression

In [160]:

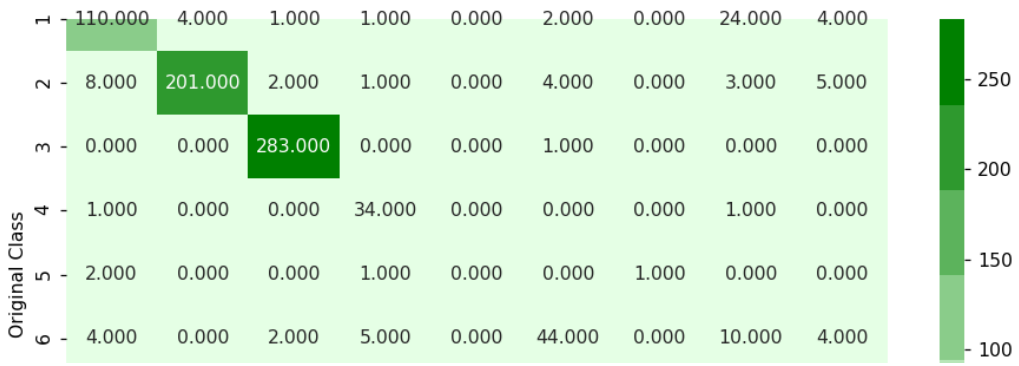
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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)

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")
```

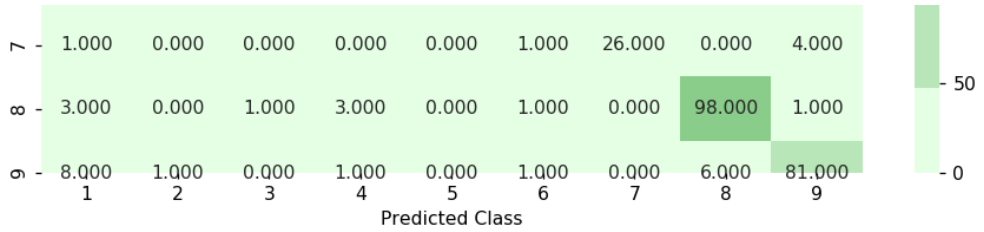
```
plt.ylabel("Error measure")
plt.show()

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 1.573370124979183
log_loss for c =  0.0001 is 1.5745266943225948
log_loss for c =  0.001 is 1.5870470140051742
log_loss for c =  0.01 is 1.1704765158801869
log_loss for c =  0.1 is 0.9421569004507702
log_loss for c =  1 is 0.7947519369618701
log_loss for c =  10 is 0.6361511204664396
log_loss for c =  100 is 0.5853974727882401
log_loss for c =  1000 is 0.6991053224429564
```

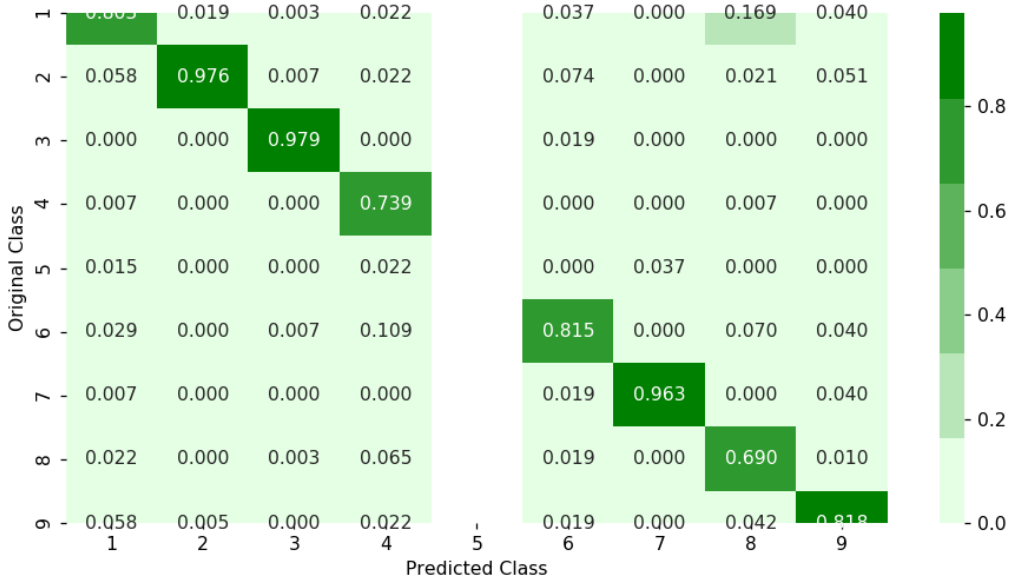



```
log loss for train data 0.4832327007745169
log loss for cv data 0.5853974727882401
log loss for test data 0.5649433249003583
Number of misclassified points  12.3
---------------------------------------------------- Confusion matrix ----------------------------------------------------
```





---------- 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.]

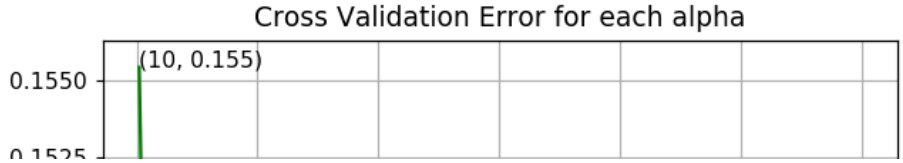
4.1.4. Random Forest Classifier

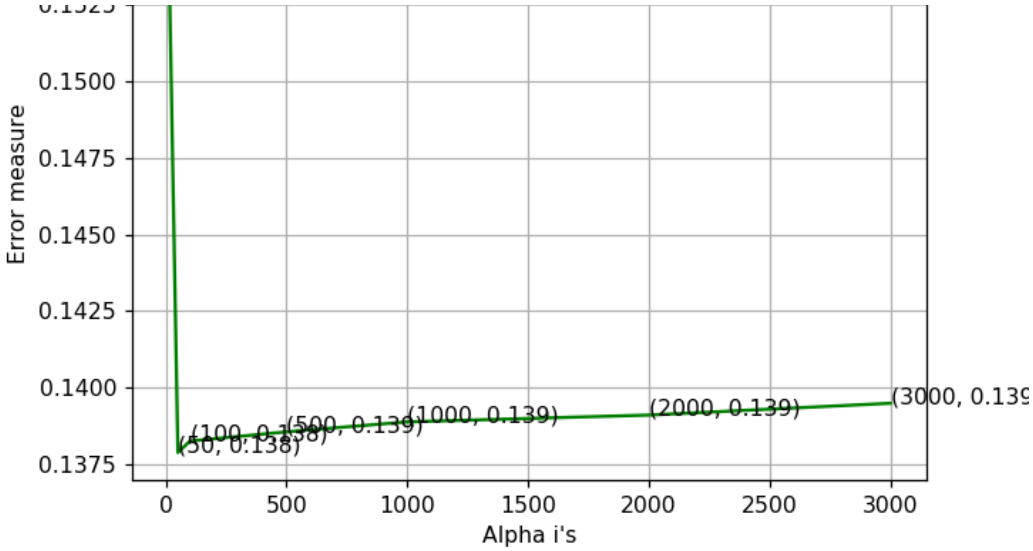
In [161]:

```
# ----------------------------------
# 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_decr
```

```
ease=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-const
ruction-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)

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))
```

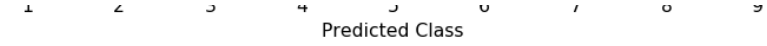
```
log_loss for c =  10 is 0.1554429689198267
log_loss for c =  50 is 0.13789033279233023
log_loss for c =  100 is 0.1382623847309797
log_loss for c =  500 is 0.1385641431647808
log_loss for c =  1000 is 0.1388887122528967
log_loss for c =  2000 is 0.13911064892301606
log_loss for c =  3000 is 0.13949508772068298
```





For values of best alpha = 50 The train log loss is: 0.047936938264203074
For values of best alpha = 50 The cross validation log loss is: 0.13789033279233023
For values of best alpha = 50 The test log loss is: 0.13268423025949708
Number of misclassified points 3.4000000000000004
---------- 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.]

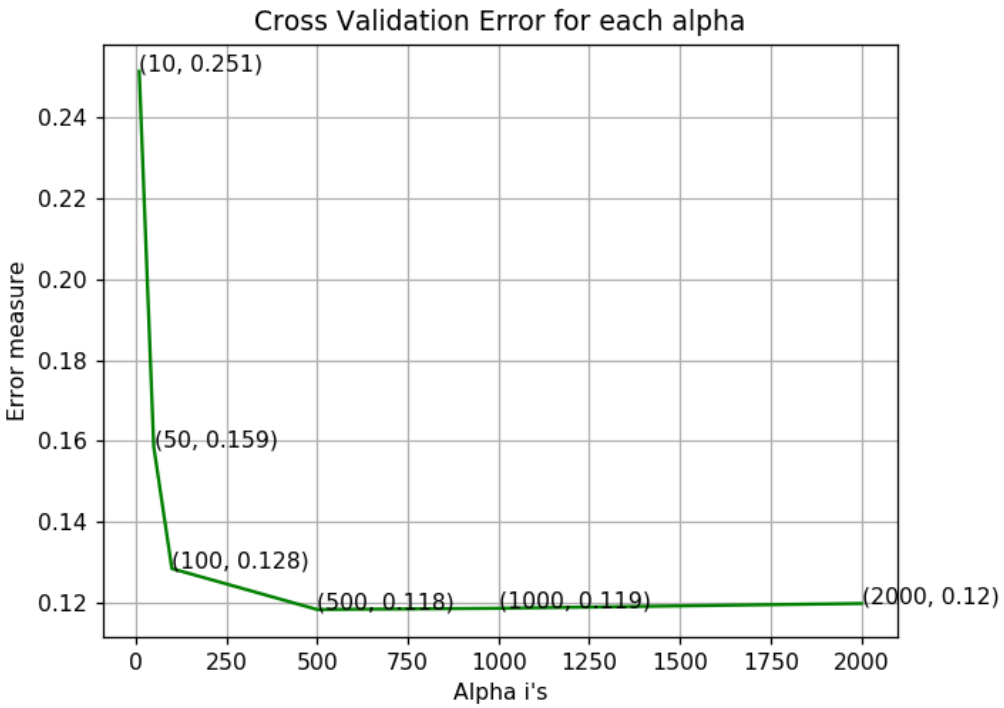
4.1.5. XgBoost Classification

In [162]:

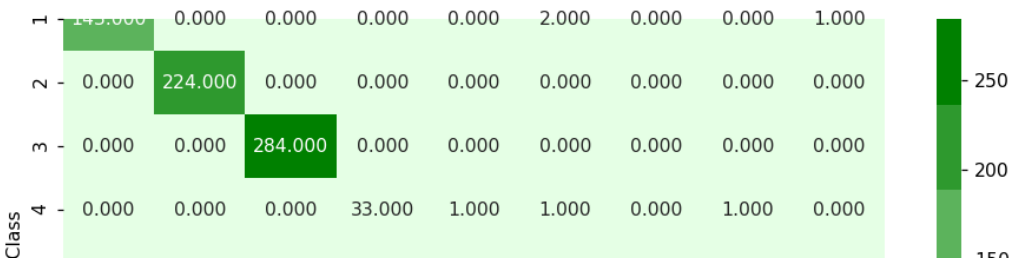
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# 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=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
```

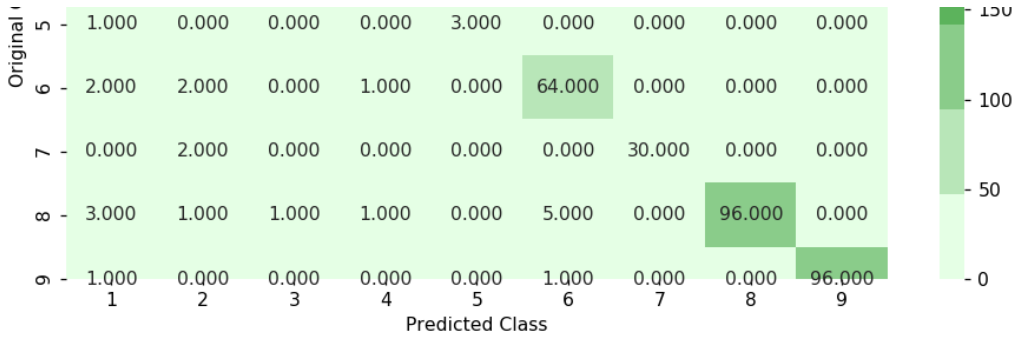
```
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```
log_loss for c =  10 is 0.25140361033179326
log_loss for c =  50 is 0.15857152430357202
log_loss for c =  100 is 0.12844120251753532
log_loss for c =  500 is 0.1183296134178272
log_loss for c =  1000 is 0.11862469966858534
log_loss for c =  2000 is 0.11981340066871113
```




```
For values of best alpha =  500 The train log loss is: 0.039464139177552274
For values of best alpha =  500 The cross validation log loss is: 0.1183296134178272
For values of best alpha =  500 The test log loss is: 0.13282584155795696
Number of misclassified points  2.7
---------------------------------------------------- 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 [0]:

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

```
x_cfl=XGBClassifier()

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

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

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

Out[0]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max
_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgbo
ost.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=2000, learning_rate=0.05, colsample_bytree=1, max_depth=3)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

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

test loss 0.07020000387096

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 pre sent 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 [0]:

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

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

In [0]:

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

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

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

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

    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    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', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
```

```
    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', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    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', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    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', 'call',
'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    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 [0]:

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

Out[0]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	e

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

5 rows × 53 columns

In [19]:

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

Out[19]:

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

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [20]:

```
#file sizes of asm files

files=os.listdir('D:/microsoft_malware/train/byteFiles/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('D:/microsoft_malware/train/byteFiles/asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

```
                ID      size  Class
0  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 [21]:

```
asm_size_byte.shape
```

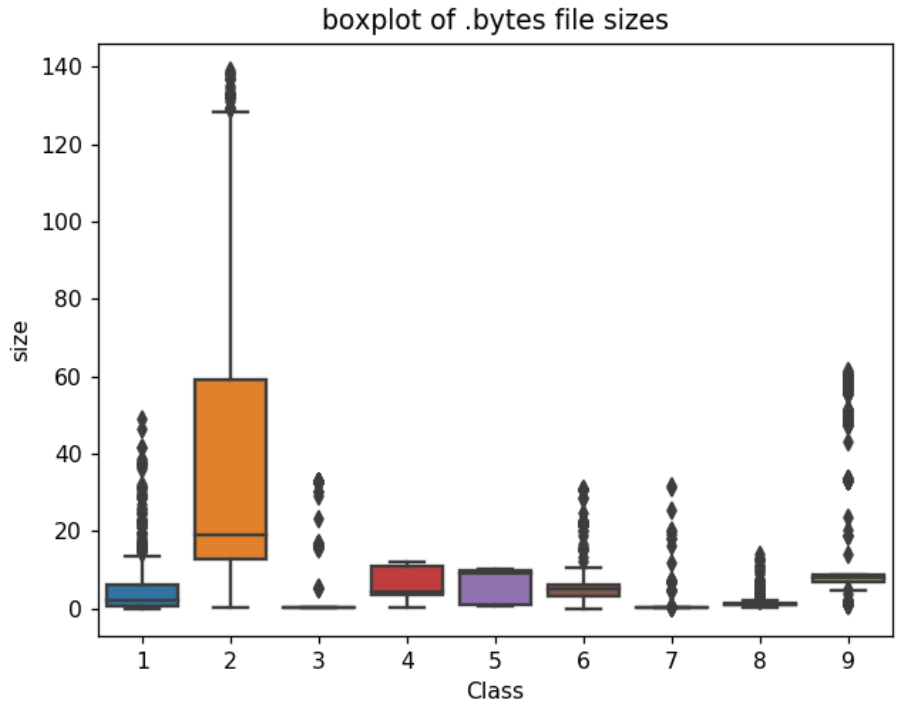
Out[21]:

(4997, 3)

4.2.1.2 Distribution of .asm file sizes

In [167]:

```
#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 [22]:

```
result_asm.shape
```

Out[22]:

(10868, 53)

In [23]:

```
asm_size_byte.shape
```

Out[23]:

(4997, 3)

In [24]:

```
final_bytes = pd.concat([data_size_byte,byte_features], axis=1)
final_bytes = final_bytes.dropna(axis=0)

final_bytes.shape

final_bytes = final_bytes.loc[:,~final_bytes.columns.duplicated()]
```

In [ ]:

```

```

In [25]:

```
final_asm = pd.concat([asm_size_byte,result_asm], axis=1)
```

In [26]:

```
final_asm = final_asm.dropna(axis = 0)
```

In [27]:

```
final_asm.head()
```

Out[27]:

	ID	size	Class	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	...	edx	esi
0	01azqd4InC7m9JpocGv5	56.229886	9.0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	...	18	66
1	01IsoiSMh5gxyDYTl4CB	13.999378	2.0	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	...	18	29
2	01jsnpXSAlgw6aPeDxrU	8.507785	9.0	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	...	13	42
3	01kcPWA9K2BOxQeS5Rju	0.078190	1.0	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	...	6	8
4	01SuzwMJEIXsK7A8dQbl	0.996723	8.0	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	...	12	9

5 rows × 56 columns

In [28]:

```
final_asm = final_asm.loc[:,~final_asm.columns.duplicated()]
```

In [29]:

```
final_asm.head()
```

Out[29]:

	ID	size	Class	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	...	:dword	edx	esi	eax	ebx
0	01azqd4InC7m9JpocGv5	56.229886	9.0	19	744	0	127	57	0	323	...	137	18	66	15	43
1	01IsoiSMh5gxyDYTl4CB	13.999378	2.0	17	838	0	103	49	0	0	...	130	18	29	48	82
2	01jsnpXSAlgw6aPeDxrU	8.507785	9.0	17	427	0	50	43	0	145	...	84	13	42	10	67
3	01kcPWA9K2BOxQeS5Rju	0.078190	1.0	17	227	0	43	19	0	0	...	25	6	8	14	7
4	01SuzwMJEIXsK7A8dQbl	0.996723	8.0	17	402	0	59	170	0	0	...	18	12	9	18	29

5 rows × 54 columns

In [30]:

```
final_asm = pd.merge(asm_size_byte.drop(['Class'],axis=1),result_asm,on='ID', how='left')
final_asm.head()
```

Out[30]:

	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	edx	esi	eax	ebx
0	01azqd4InC7m9JpocGv5	56.229886	18	22430	0	1158	1366754	0	1794	0	...	808	2290	1281	587
1	01IsoiSMh5gxyDYTl4CB	13.999378	0	109939	0	616	24568	0	26405	0	...	260	1090	391	905
2	01jsnpXSAlgw6aPeDxrU	8.507785	18	68883	0	304	662	0	1093	0	...	5	547	5	451
3	01kcPWA9K2BOxQeS5Rju	0.078190	19	744	0	127	57	0	323	0	...	18	66	15	43
4	01SuzwMJEIXsK7A8dQbl	0.996723	18	10368	0	206	4595	92	0	0	...	18	1228	24	1546

5 rows × 54 columns

In [31]:

```
result_asm.shape
```

Out[31]:

(10868, 53)

In [32]:

```
# final_asm = final_asm.drop(['size_x','size_y'],axis = 1)
```

In [33]:

```
final_asm.head()
```

Out[33]:

	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	edx	esi	eax	ebx
0	01azqd4InC7m9JpocGv5	56.229886	18	22430	0	1158	1366754	0	1794	0	...	808	2290	1281	587
1	01IsoiSMh5gxyDYTl4CB	13.999378	0	109939	0	616	24568	0	26405	0	...	260	1090	391	905
2	01jsnpXSAlgw6aPeDxrU	8.507785	18	68883	0	304	662	0	1093	0	...	5	547	5	451
3	01kcPWA9K2BOxQeS5Rju	0.078190	19	744	0	127	57	0	323	0	...	18	66	15	43
4	01SuzwMJEIXsK7A8dQbl	0.996723	18	10368	0	206	4595	92	0	0	...	18	1228	24	1546

5 rows × 54 columns

In [34]:

```
final_asm.loc[final_asm['ID'] == '01kcPWA9K2BOxQeS5Rju']
```

Out[34]:

	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	edx	esi	eax	ebx	ecx	edi
3	01kcPWA9K2BOxQeS5Rju	0.07819	19	744	0	127	57	0	323	0	...	18	66	15	43	83	0

1 rows × 54 columns

In [35]:

```
final_asm = normalize(final_asm)
```

In [36]:

```
final_asm.head()
```

Out[36]:

	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	edx	
0	01azqd4InC7m9JpocGv5	0.403912	0.204545	0.032928	0.0	0.006937	0.543192	0.000000	0.000467	0.0	...	0.016262	0.0
1	01IsoiSMh5gxyDYTl4CB	0.100466	0.000000	0.161392	0.0	0.003690	0.009764	0.000000	0.006877	0.0	...	0.005233	0.0
2	01jsnpXSAlgw6aPeDxrU	0.061006	0.204545	0.101121	0.0	0.001821	0.000263	0.000000	0.000285	0.0	...	0.000101	0.0
3	01kcPWA9K2BOxQeS5Rju	0.000436	0.215909	0.001092	0.0	0.000761	0.000023	0.000000	0.000084	0.0	...	0.000362	0.0
4	01SuzwMJEIXsK7A8dQbl	0.007036	0.204545	0.015220	0.0	0.001234	0.001826	0.012842	0.000000	0.0	...	0.000362	0.0

5 rows × 54 columns

In [37]:

```
asm_y = final_asm['Class']
```

In [38]:

```
final_asm.shape
```

Out[38]:

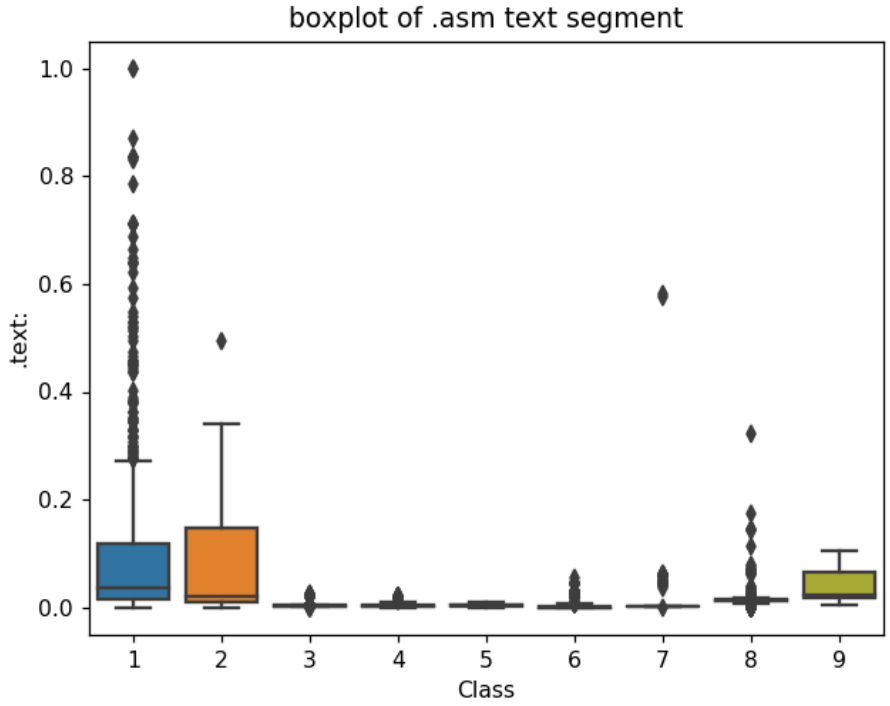
(4997, 54)

4.2.2 Univariate analysis on asm file features

In [230]:

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

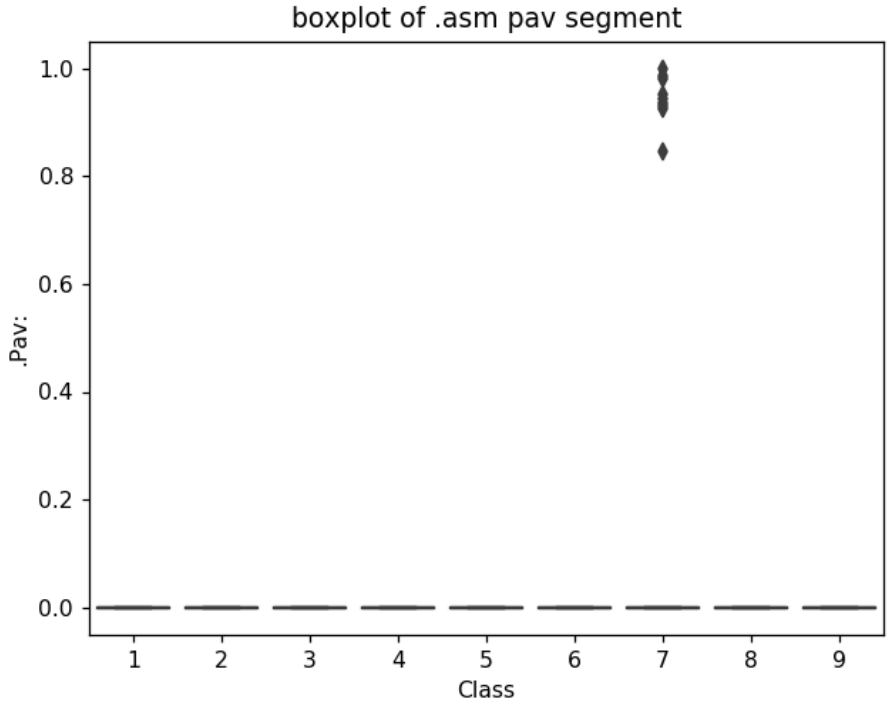
```
plt.show()
```



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

In [231]:

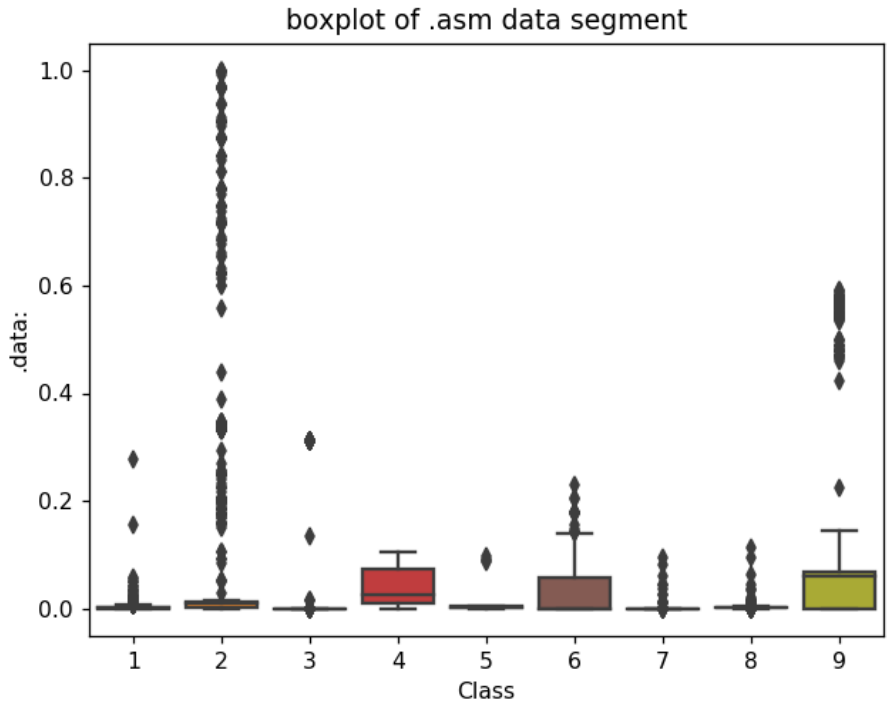
```
ax = sns.boxplot(x="Class", y=".Pav:", data = final_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```



In [232]:

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

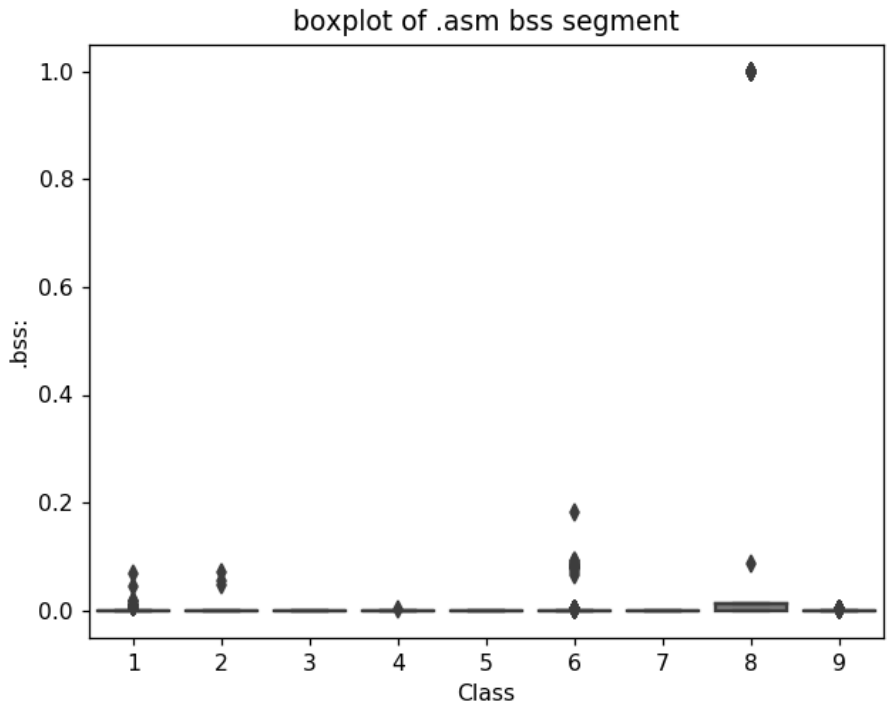
```
plt.show()
```



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

In [233]:

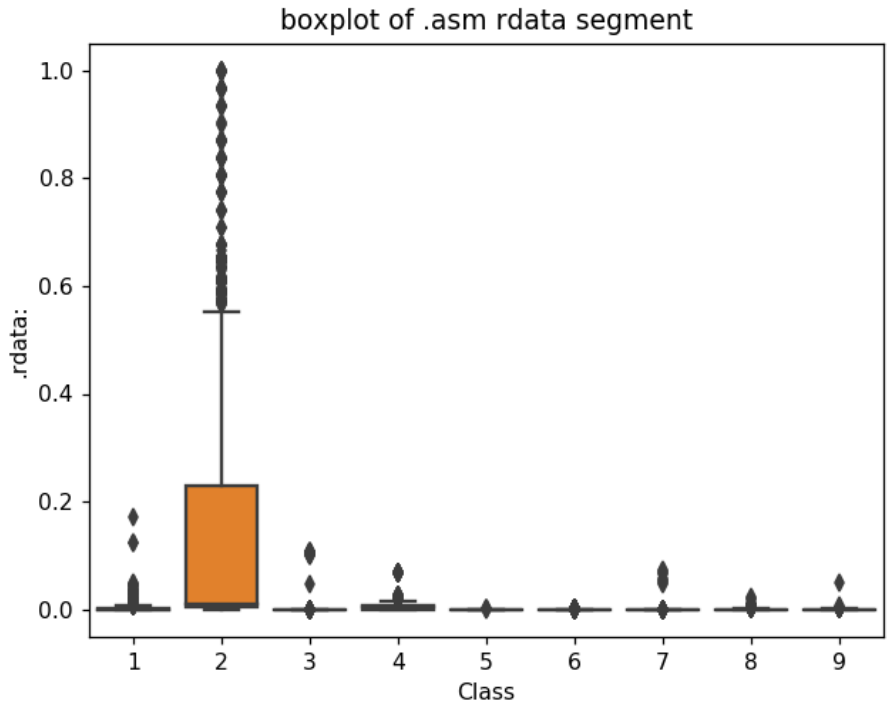
```
ax = sns.boxplot(x="Class", y=".bss:", data = final_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [234]:

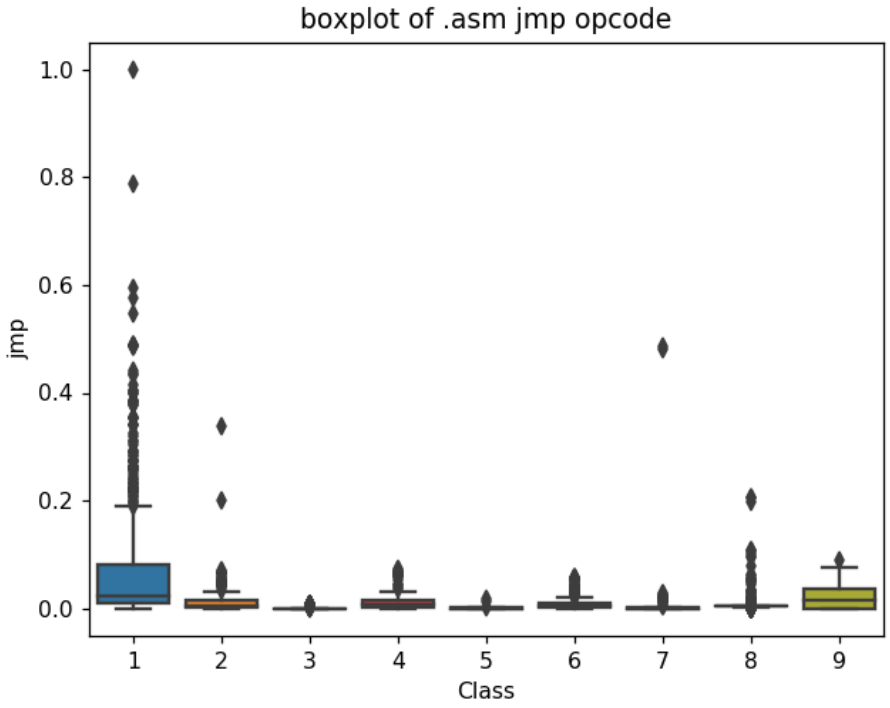
```
ax = sns.boxplot(x="Class", y=".rdata:", data = final_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

In [235]:

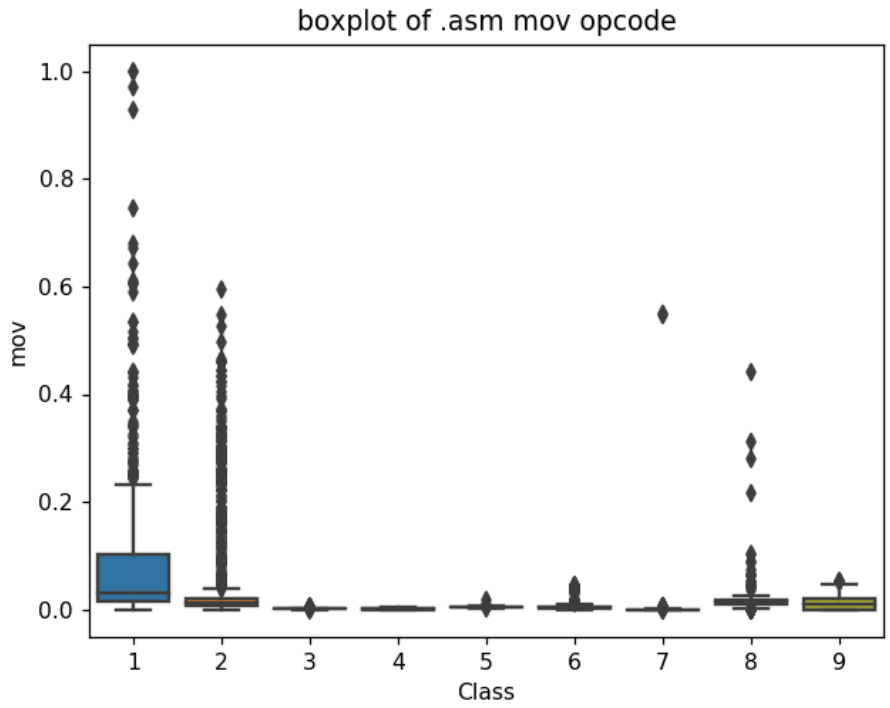
```
ax = sns.boxplot(x="Class", y="jmp", data = final_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



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

In [236]:

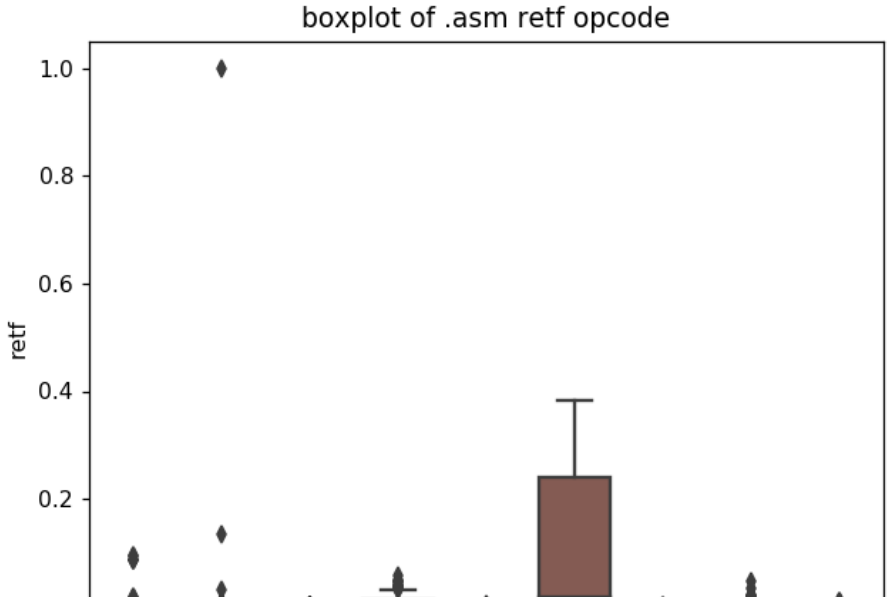
```
ax = sns.boxplot(x="Class", y="mov", data = final_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```

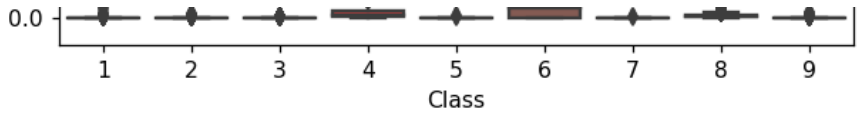


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [237]:

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

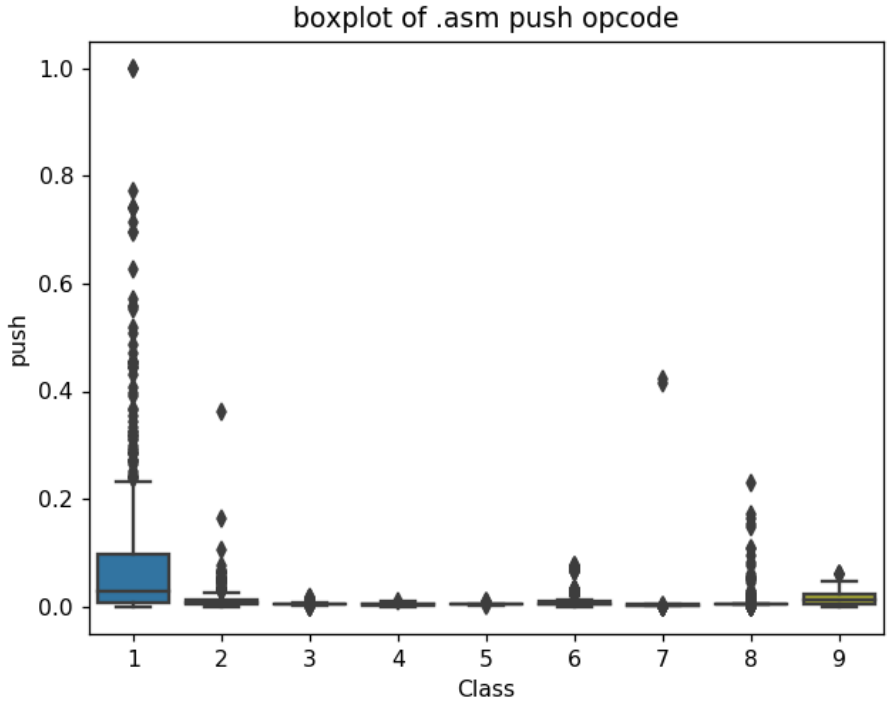




plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [238]:

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

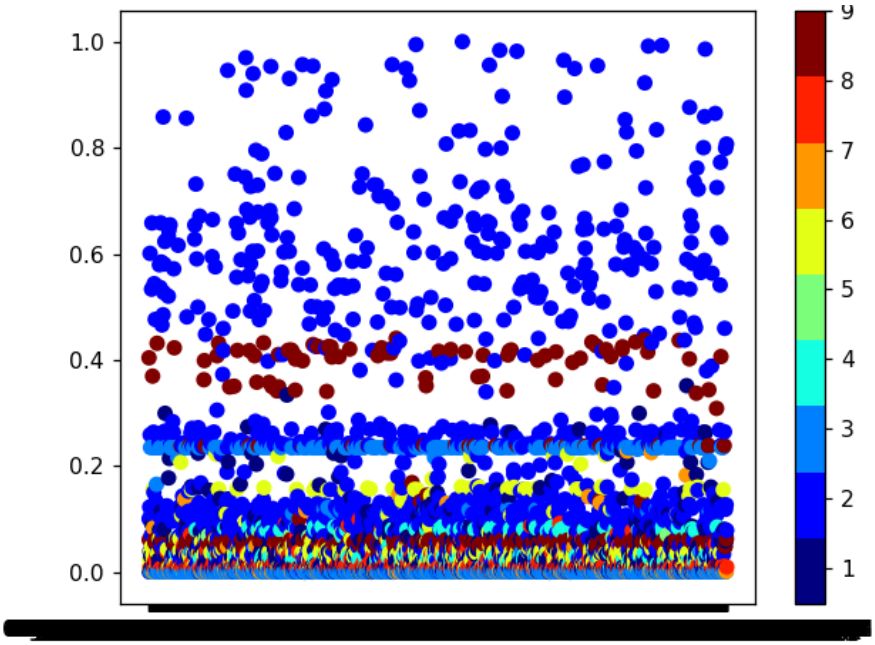


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

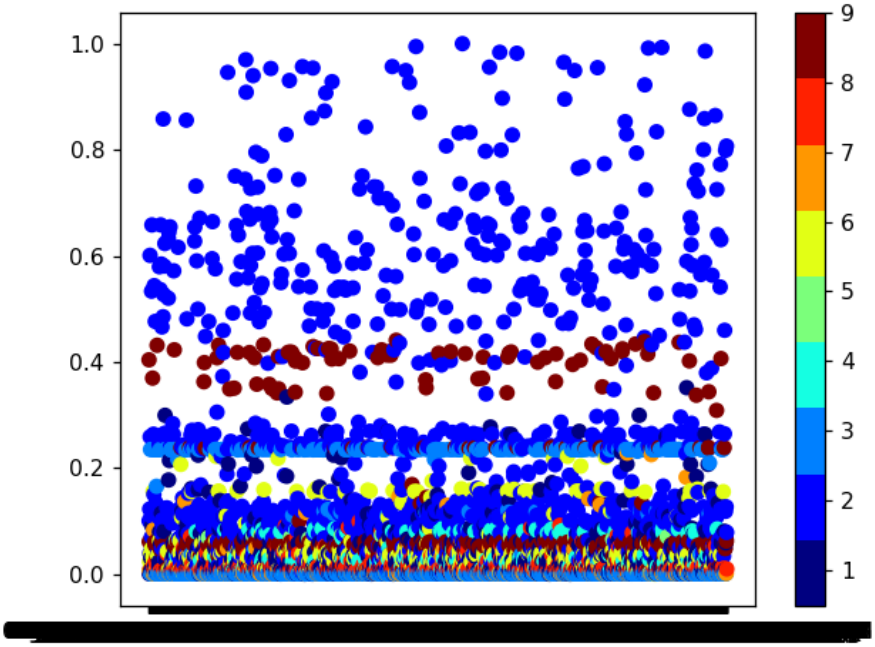
In [242]:

```
# 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(final_asm.drop(['ID','Class'], axis=1).fillna(0))
vis_x = final_asm.iloc[:,0]
vis_y = final_asm.iloc[:,1]
plt.scatter(vis_x, vis_y, c = asm_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



In [243]:

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

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



TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

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

4.3 Train and test split

In [244]:

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

In [245]:

```
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)
```

In [247]:

```
print(X_train_asm.shape)
print(X_test_asm.shape)
print(X_cv_asm.shape)
```

```
(3200, 49)
(1000, 49)
(800, 49)
```

In [246]:

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

```
size       False
HEADER:    False
.text:     False
.Pav:      False
.idata:    False
.data:     False
.bss:      False
.rdata:    False
.edata:    False
.rsrc:     False
.tls:      False
.reloc:    False
jmp        False
mov        False
retf       False
push       False
pop        False
xor        False
retn       False
nop        False
sub        False
inc        False
dec        False
add        False
imul       False
xchg       False
or         False
shr        False
cmp        False
call       False
shl        False
ror        False
rol        False
jnb        False
jz         False
lea        False
movzx      False
.dll       False
std::      False
```

```
:dword     False
edx        False
esi        False
eax        False
ebx        False
ecx        False
edi        False
ebp        False
esp        False
eip        False
dtype: bool
```

4.4. Machine Learning models on features of .asm files

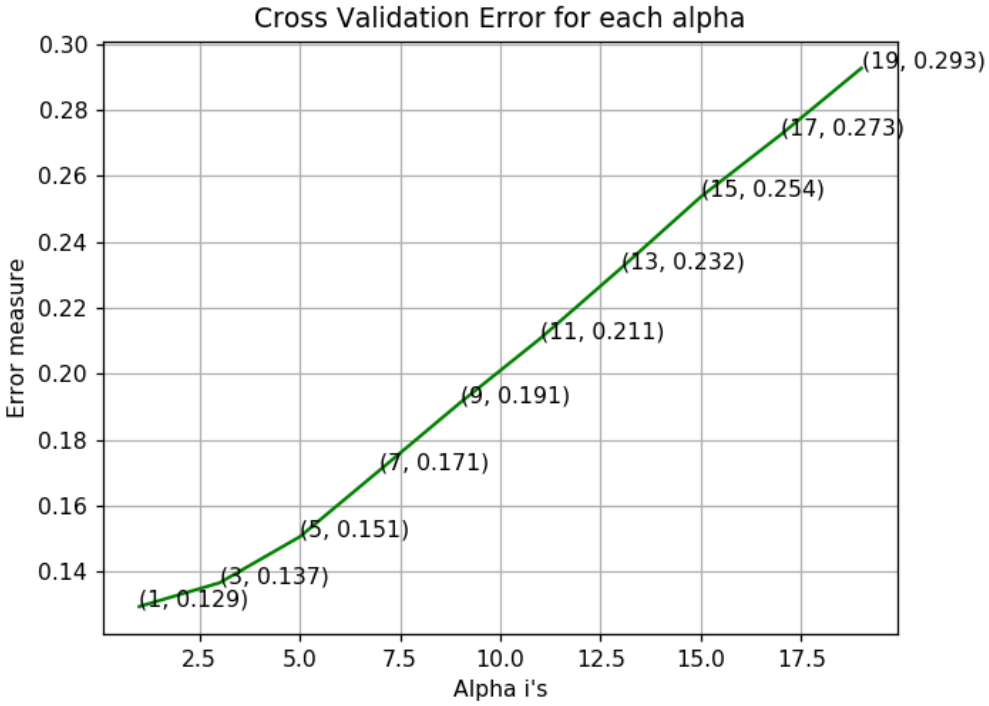
4.4.1 K-Nearest Neigbors

In [248]:

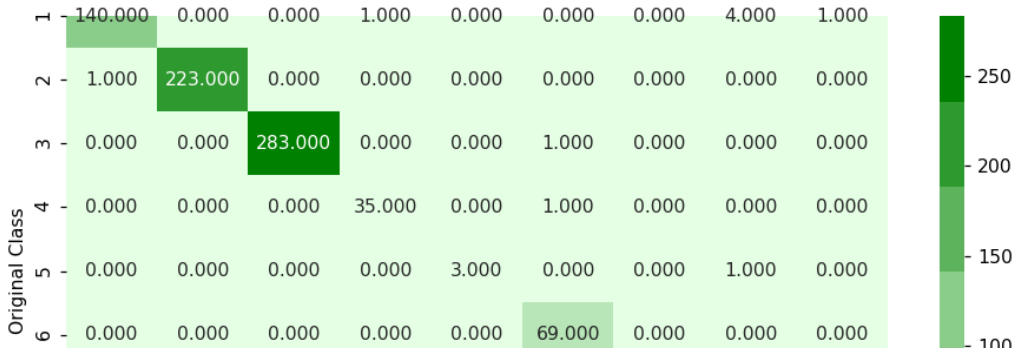
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', 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-example-1/
#--------------------------------------


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

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

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

best_alpha = np.argmin(cv_log_error_array)

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

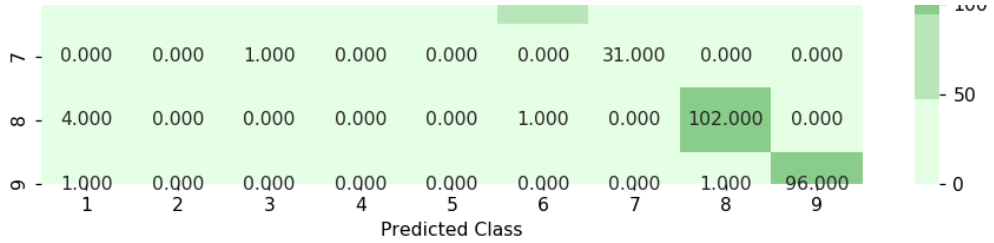
```
k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


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

log_loss for k = 1 is 0.12949056818665164
log_loss for k = 3 is 0.1366613146080358
log_loss for k = 5 is 0.15060972105718556
log_loss for k = 7 is 0.17087031896443614
log_loss for k = 9 is 0.1912783046768009
log_loss for k = 11 is 0.2107728160977453
log_loss for k = 13 is 0.23195047488675122
log_loss for k = 15 is 0.25370502334676637
log_loss for k = 17 is 0.27259770002232997
log_loss for k = 19 is 0.29267202004241716



log loss for train data 0.04449232129473724
log loss for cv data 0.12949056818665164
log loss for test data 0.11285424146983077
Number of misclassified points 1.7999999999999998
---------- Confusion matrix ----------





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

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

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

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

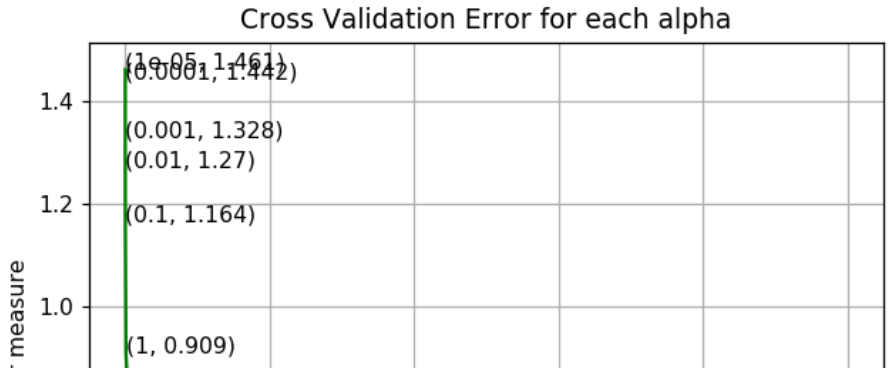
4.4.2 Logistic Regression

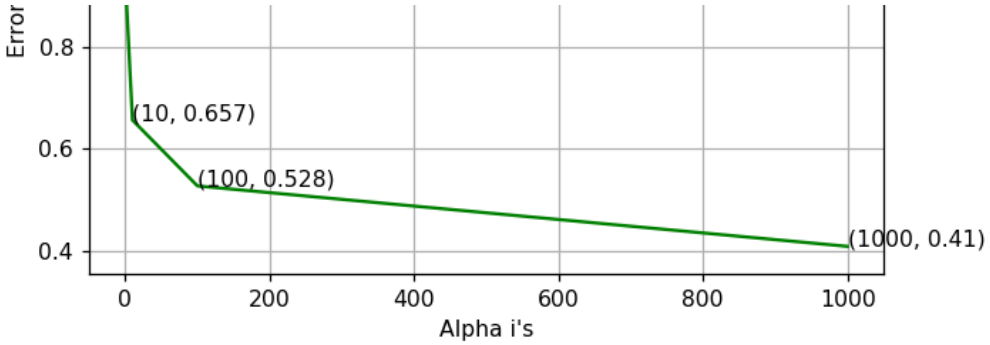
In [249]:

```
# 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)

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

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

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

```
log_loss for c =  1e-05 is 1.4607478144110317
log_loss for c =  0.0001 is 1.4417178329616969
log_loss for c =  0.001 is 1.3281257180378683
log_loss for c =  0.01 is 1.2697683218881801
log_loss for c =  0.1 is 1.1640366758139644
log_loss for c =  1 is 0.9085228490157402
log_loss for c =  10 is 0.656565128178767
log_loss for c =  100 is 0.5281852303641492
log_loss for c =  1000 is 0.4104629589481204
```

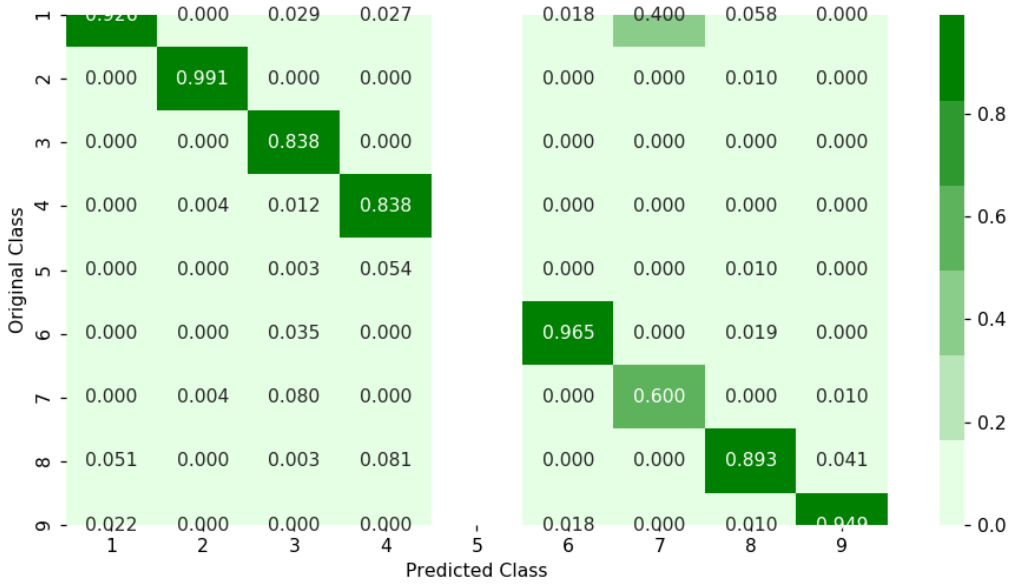




log loss for train data 0.3400082158008072
log loss for cv data 0.4104629589481204
log loss for test data 0.35520187256705854
Number of misclassified points 9.3

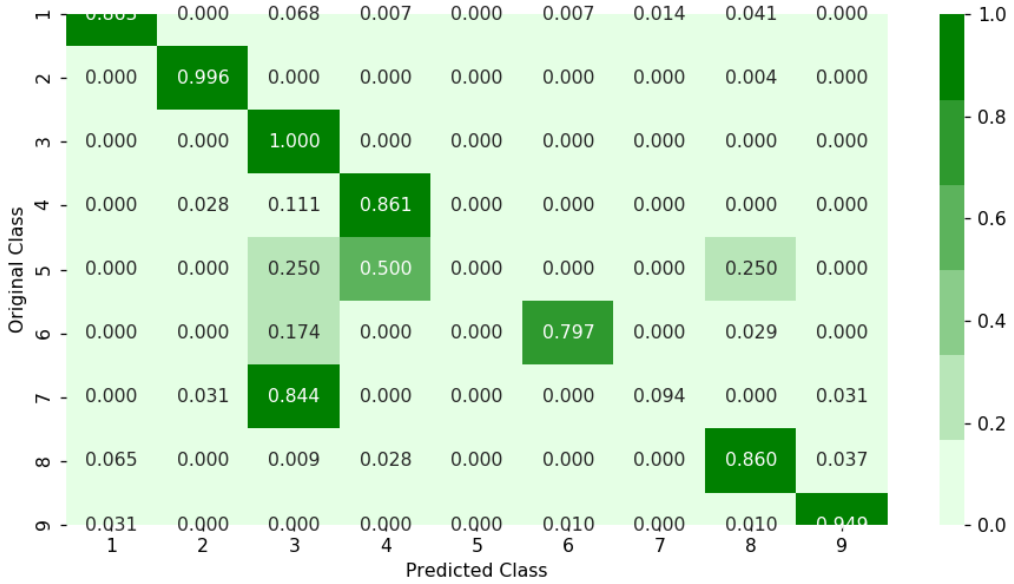
---------- Confusion matrix ----------

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



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

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



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

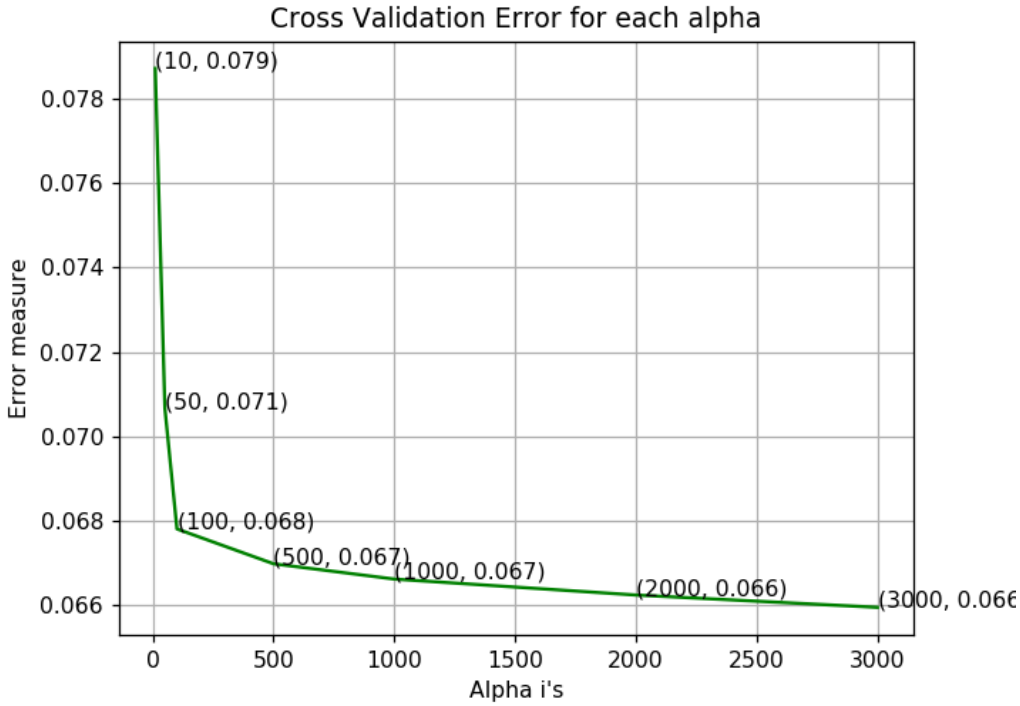
4.4.3 Random Forest Classifier

In [250]:

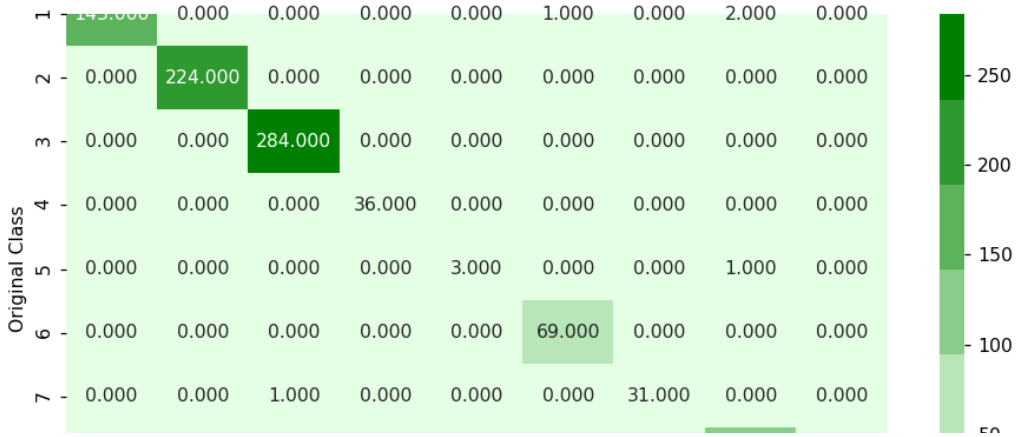
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,

# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decr
ease=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-const
ruction-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)

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")
```

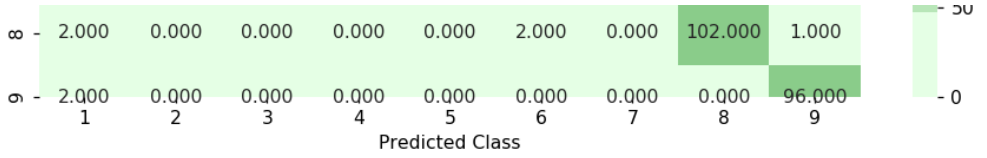
```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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 0.07871832065433097
log_loss for c = 50 is 0.07066739249383945
log_loss for c = 100 is 0.06781111127185036
log_loss for c = 500 is 0.06698598834474077
log_loss for c = 1000 is 0.06662408209013196
log_loss for c = 2000 is 0.06624639627048456
log_loss for c = 3000 is 0.06595462150051937



log loss for train data 0.021182739229039863
log loss for cv data 0.06595462150051937
log loss for test data 0.0636196808834797
Number of misclassified points 1.2
---------- 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.]

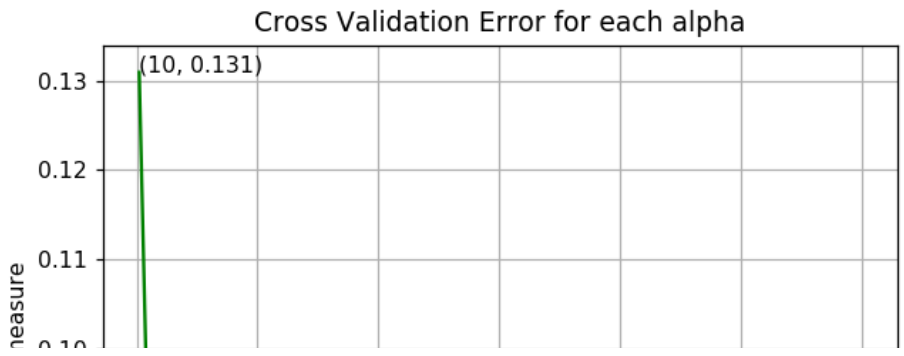
4.4.4 XgBoost Classifier

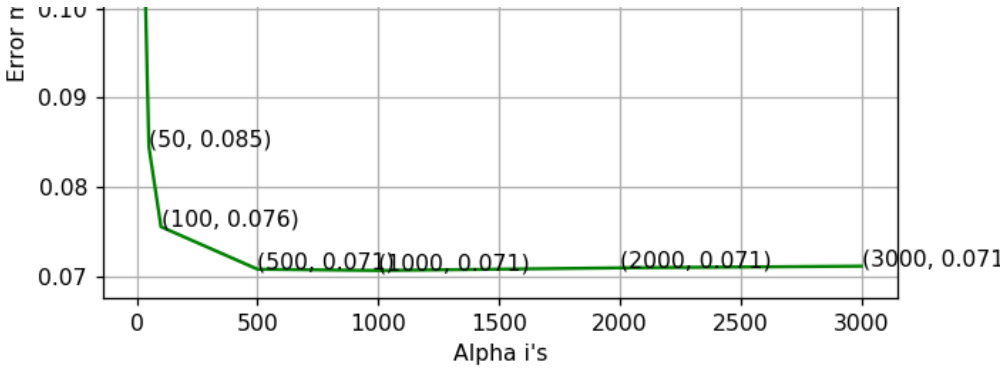
In [251]:

```
# 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
```

```
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

```
log_loss for c =  10 is 0.13098348409609714
log_loss for c =  50 is 0.08451010044000658
log_loss for c =  100 is 0.07553449545792704
log_loss for c =  500 is 0.07077163185093106
log_loss for c =  1000 is 0.07063053978561896
log_loss for c =  2000 is 0.07092705191991847
log_loss for c =  3000 is 0.07111821955656407
```





For values of best alpha = 1000 The train log loss is: 0.020652354037663603
For values of best alpha = 1000 The cross validation log loss is: 0.07063053978561896
For values of best alpha = 1000 The test log loss is: 0.061433103921705784
Number of misclassified points 1.0999999999999999
---------- 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 [0]:

```
x_cfl=XGBClassifier()

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

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

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

Out[0]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max
_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgbo
ost.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=0.5,learning_rate=0.15,colsample_bytree=0.5,max_depth=3)
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

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

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

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

4.5.1. Merging both asm and byte file features

In [149]:

```
final_bytes.head()
```

Out[149]:

	ID	size	Class	0	1	2	3	4	5	6	...	f7	f8	f9	fa	fb	
0	01azqd4InC7m9JpocGv5	5.012695	9.0	601905	3905	2816	3832	3345	3242	3650	...	2804	3687	3101	3211	3097	275
1	01IsoiSMh5gxyDYTl4CB	6.556152	2.0	39755	8337	7249	7186	8663	6844	8420	...	451	6536	439	281	302	763
2	01jsnpXSAlgw6aPeDxrU	4.602051	9.0	93506	9542	2568	2438	8925	9330	9007	...	2325	2358	2242	2885	2863	247
3	01kcPWA9K2BOxQeS5Rju	0.679688	1.0	21091	1213	726	817	1257	625	550	...	478	873	485	462	516	113
4	01SuzwMJEIXsK7A8dQbl	0.438965	8.0	19764	710	302	433	559	410	262	...	847	947	350	209	239	65

5 rows × 260 columns

In [150]:

```
final_asm.head()
```

Out[150]:

	ID	size	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	edx	
0	01azqd4InC7m9JpocGv5	0.403912	0.204545	0.032928	0.0	0.006937	0.543192	0.000000	0.000467	0.0	...	0.016262	0.0
1	01IsoiSMh5gxyDYTl4CB	0.100466	0.000000	0.161392	0.0	0.003690	0.009764	0.000000	0.006877	0.0	...	0.005233	0.0
2	01jsnpXSAlgw6aPeDxrU	0.061006	0.204545	0.101121	0.0	0.001821	0.000263	0.000000	0.000285	0.0	...	0.000101	0.0
3	01kcPWA9K2BOxQeS5Rju	0.000436	0.215909	0.001092	0.0	0.000761	0.000023	0.000000	0.000084	0.0	...	0.000362	0.0
4	01SuzwMJEIXsK7A8dQbl	0.007036	0.204545	0.015220	0.0	0.001234	0.001826	0.012842	0.000000	0.0	...	0.000362	0.0

5 rows × 54 columns

In [151]:

```
print(final_bytes.shape)
print(final_asm.shape)
```

```
(5000, 260)
(4997, 54)
```

In [255]:

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

Out[255]:

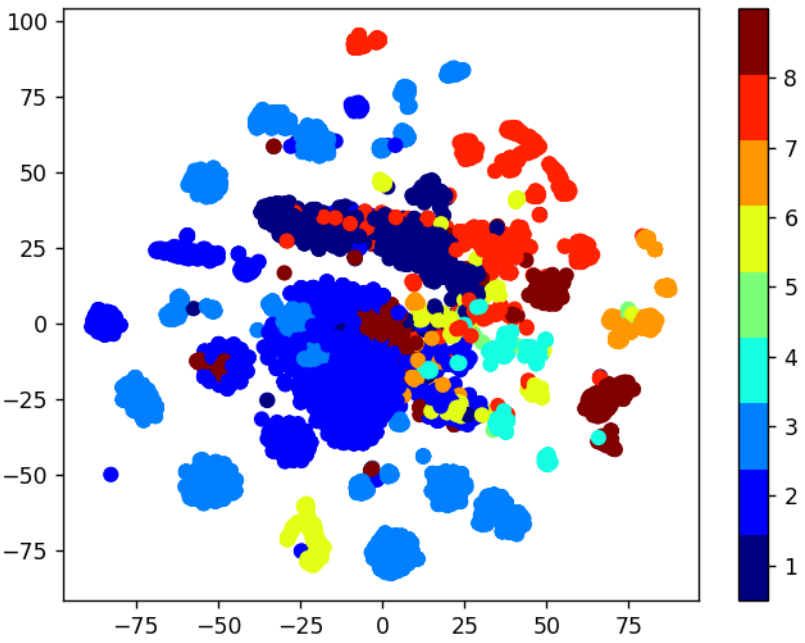
	size_x	0	1	2	3	4	5	6	7	8	...	:dword	edx	
0	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.003511	...	0.034960	0.016262	0.0
1	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.011003	...	0.011566	0.005233	0.0
2	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.010785	...	0.007222	0.000101	0.0
3	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.001277	...	0.001096	0.000362	0.0
4	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000430	0.000500	...	0.009758	0.000362	0.0

5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

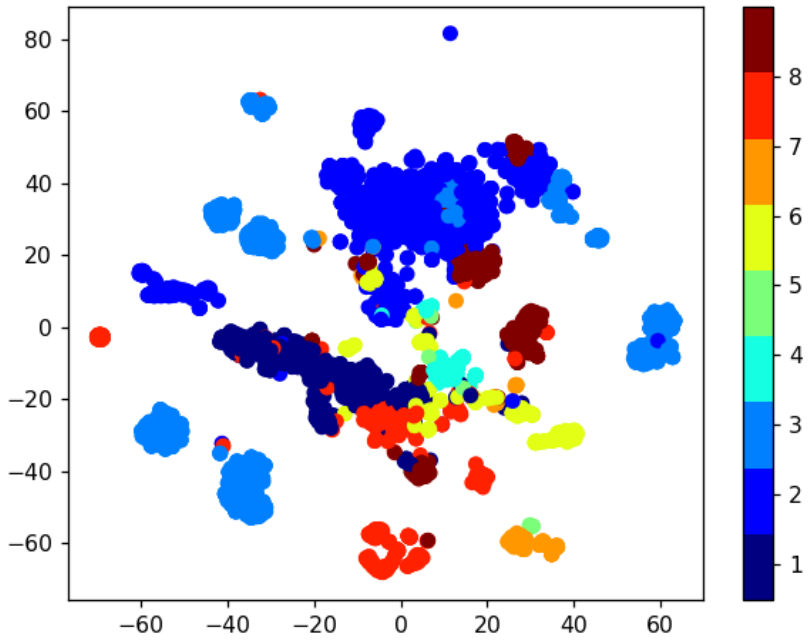
In [0]:

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



In [257]:

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



4.5.3. Train and Test split

In [0]:

```
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)
```

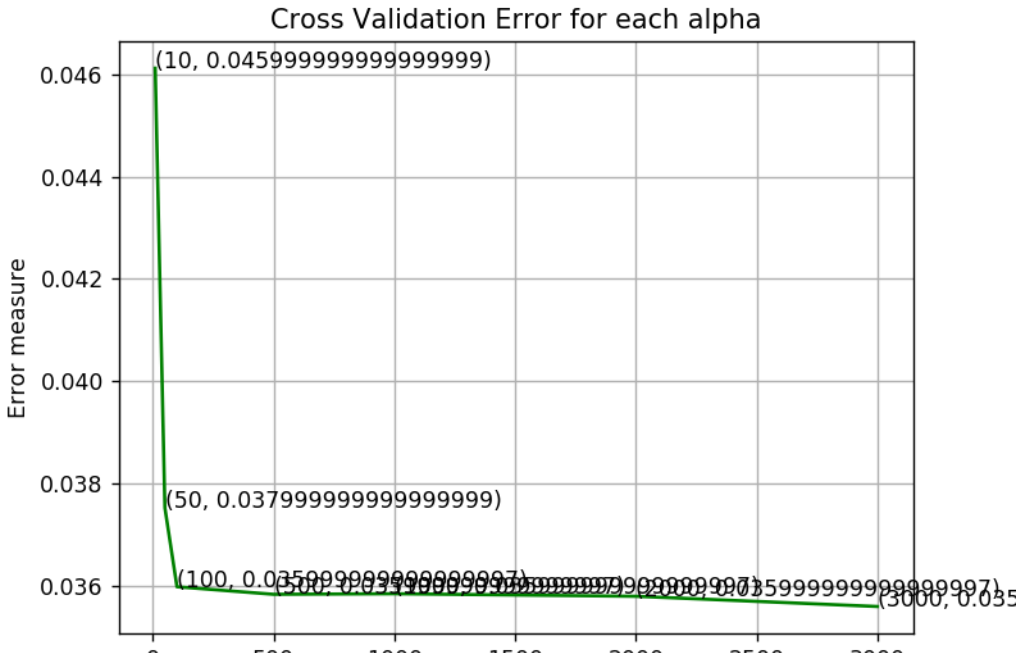
4.5.4. Random Forest Classifier on final features

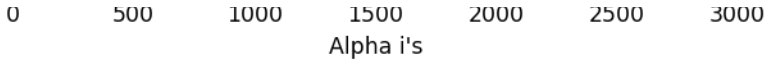
In [0]:

```
# -------------------------------
# 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=[]
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 0.0461221662017
log_loss for c =  50 is 0.0375229563452
log_loss for c =  100 is 0.0359765822455
log_loss for c =  500 is 0.0358291883873
log_loss for c =  1000 is 0.0358403093496
log_loss for c =  2000 is 0.0357908022178
log_loss for c =  3000 is 0.0355909487962
```





For values of best alpha = 3000 The train log loss is: 0.0166267614753
For values of best alpha = 3000 The cross validation log loss is: 0.0355909487962
For values of best alpha = 3000 The test log loss is: 0.0401141303589

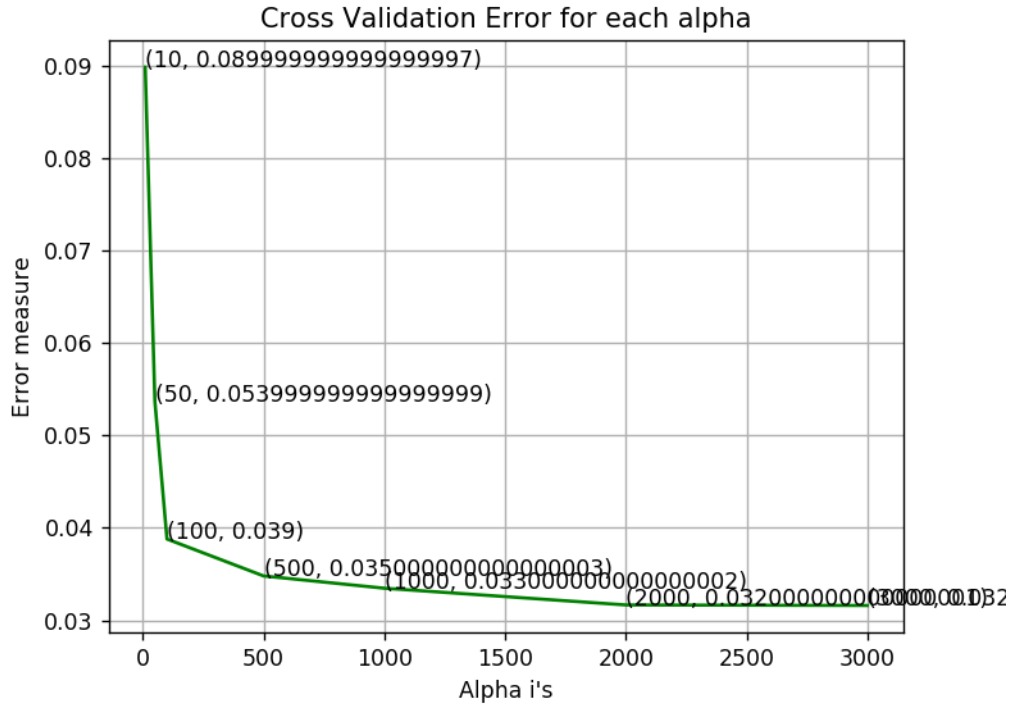
4.5.5. XgBoost Classifier on final features

In [0]:

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

# find more about XGBClassifier function here http://xgboost.readthedocs.io/en/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=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

x_cfl=XGBClassifier(n_estimators=3000,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv_merge, predict_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 0.0898979446265
log_loss for c = 50 is 0.0536946658041
log_loss for c = 100 is 0.0387968186177
log_loss for c = 500 is 0.0347960327293
log_loss for c = 1000 is 0.0334668083237

log_loss for c = 2000 is 0.0316569078846
log_loss for c = 3000 is 0.0315972694477



For values of best alpha = 3000 The train log loss is: 0.0111918809342
For values of best alpha = 3000 The cross validation log loss is: 0.0315972694477
For values of best alpha = 3000 The test log loss is: 0.0323978515915

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

In [0]:

```
x_cfl=XGBClassifier()

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

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

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

Out[0]:

```
RandomizedSearchCV(cv=None, error_score='raise',
          estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
       gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1),
          fit_params=None, iid=True, n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators': [100, 200, 500, 1000, 2000], 'max
_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score=True, scoring=None, verbose=10)
```

In [0]:

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

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

In [0]:

```
# 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=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.3,subsample=1,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  3000 The train log loss is: 0.0121922832297
For values of best alpha =  3000 The cross validation log loss is: 0.0344955487471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

5. Assignments

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

Merging all byte and asm features

In [46]:

```
final_asm.shape
```

Out[46]:

```
(4997, 54)
```

In [47]:

```
final_bytes.shape
```

Out[47]:

```
(5000, 260)
```

In [48]:

```
final_bytes.head()
```

Out[48]:

	ID	size	Class	0	1	2	3	4	5	6	...	f7	f8	f9	fa	fb	
0	01azqd4InC7m9JpocGv5	5.012695	9.0	601905	3905	2816	3832	3345	3242	3650	...	2804	3687	3101	3211	3097	275
1	01IsoiSMh5gxyDYTI4CB	6.556152	2.0	39755	8337	7249	7186	8663	6844	8420	...	451	6536	439	281	302	763
2	01jsnpXSAlgw6aPeDxrU	4.602051	9.0	93506	9542	2568	2438	8925	9330	9007	...	2325	2358	2242	2885	2863	247
3	01kcPWA9K2BOxQeS5Rju	0.679688	1.0	21091	1213	726	817	1257	625	550	...	478	873	485	462	516	113
4	01SuzwMJEIXsK7A8dQbl	0.438965	8.0	19764	710	302	433	559	410	262	...	847	947	350	209	239	65

5 rows × 260 columns

- final_bytes_1 has only 4997 rows as 3 files from asm are corrupted

In [49]:

```
final_bytes_1 = final_bytes[final_bytes.ID != '1F7jeznmCoiBxNdIKsWT']
```

In [50]:

```
final_bytes_1 = final_bytes_1[final_bytes_1.ID != '1JZBS8ICkPfIYzXbOypc']
```

In [51]:

```
final_bytes_1 = final_bytes_1[final_bytes_1.ID != '4PLRF3BAbEtZTXnaGqfD']
```

In [52]:

```
final_bytes_1.shape
```

Out[52]:

(4997, 260)

In [53]:

```
final_bytes_1 = normalize(final_bytes_1)
```

In [54]:

```
result_x = pd.merge(final_bytes_1,final_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[54]:

	size_x	0	1	2	3	4	5	6	7	8	...	:dword	edx	
0	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	0.005531	0.003511	...	0.034960	0.016262	0.0
1	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	0.013114	0.011003	...	0.011566	0.005233	0.0
2	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	0.004047	0.010785	...	0.007222	0.000101	0.0
3	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	0.000904	0.001277	...	0.001096	0.000362	0.0
4	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	0.000430	0.000500	...	0.009758	0.000362	0.0

5 rows × 307 columns

In [55]:

```
result_y.head()
```

Out[55]:

```
0    9.0
1    2.0
2    9.0
3    1.0
4    8.0
Name: Class, dtype: float64
```

In [56]:

```
result_x.shape
```

Out[56]:

(4997, 307)

In [ ]:

In [ ]:

1.Extracting Bigrams from bytes files

In [5]:

```
byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [6]:

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

Out[6]:

66049

In [7]:

```
byte_bigram_vocab[:5]
```

Out[7]:

['00 00', '00 01', '00 02', '00 03', '00 04']

In [8]:

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

In [9]:

```
# a = []s
vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
bytebigram_vect = scipy.sparse.csr_matrix((5000,66049))
for i, file in tqdm(enumerate(os.listdir('F:/microsoft_bytes/byteFiles'))):
    f = open('F:/microsoft_bytes/byteFiles/' + file)
    bytebigram_vect[i,:] =  bytebigram_vect[i,:] +  scipy.sparse.csr_matrix(vector.fit_transform([f.read().replace('\n', ' ').lower()]))
    f.close()
```

5000it [6:46:11, 4.87s/it]

In [10]:

```
scipy.sparse.save_npz('bytebigram.npz', bytebigram_vect)
```

In [70]:

```
type(bytebigram_vect)
```

Out[70]:

scipy.sparse.csr.csr_matrix

In [12]:

```
from sklearn.preprocessing import normalize
import scipy
byte_bigram_vect = normalize(scipy.sparse.load_npz('F:/bytebigram.npz'), axis = 0)

type(byte_bigram_vect)
```

Out[12]:

scipy.sparse.csc.csc_matrix

In [13]:

```
byte_bigram_vect.toarray()
```

Out[13]:

```
array([[0.07176836, 0.01576409, 0.01615476, ..., 0.        , 0.        ,
        0.        ],
       [0.00521783, 0.01131176, 0.00129077, ..., 0.        , 0.        ,
        0.        ],
       [0.0042138 , 0.00931371, 0.00316641, ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.00453893, 0.02443276, 0.02067244, ..., 0.        , 0.        ,
        0.        ],
       [0.00456758, 0.01843863, 0.0102858 , ..., 0.        , 0.        ,
        0.        ],
       [0.00411234, 0.00084956, 0.00046387, ..., 0.        , 0.        ,
        0.        ]])
```

In [60]:

```
byte_bigram_DataFrame = pd.SparseDataFrame(byte_bigram_vect.toarray(),columns = byte_bigram_vocab)

byte_bigram_DataFrame.shape
```

Out[60]:

(5000, 66049)

In [62]:

```
# byte_bigram_DataFrame = byte_bigram_DataFrame.sum(axis=0,skipna=True)
```

In [74]:

```
final_asm.to_csv("final_asm.csv")
```

In [75]:

```
final_bytes_1.to_csv("final_bytes_1.csv")
```

In [76]:

```
final_bytes.to_csv("final_bytes.csv")
```

In [77]:

```
result_x.to_csv("result_x.csv")
```

In [78]:

```
result_y.to_csv("result_y.csv")
```

2. Extracting Bigrams and Trigrams from asm files

In [2]:

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

In [3]:

```python
len(opcodes)
```

Out[3]:

```
26
```

3. Function for extracting opcodes and writing it to a file

In [ ]:

```python
def opcode_collect():
    op_file = open("opcode_file.txt", "w+")
    for asmfile in os.listdir('D:/microsoft_malware/train/byteFiles/asmFiles'):
        opcode_str = ""
        with codecs.open('D:/microsoft_malware/train/byteFiles/asmFiles/' + asmfile, encoding='cp1252', errors ='replace')
as file:
            for lines in file:
                line = lines.rstrip().split()
                for i in line:
                    if i in opcodes:
                        opcode_str += i + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
```

In [92]:

```python
opcode_collect()
```

In [5]:

```python
# asmopcodetetragram = []
# for i, v in enumerate(opcodes):
#     for j in range(0, len(opcodes)):
#         for k in range(0, len(opcodes)):
#             for l in range(0, len(opcodes)):
#                 asmopcodetetragram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k] + ' ' + opcodes[l])
# len(asmopcodetetragram)
```

Out[5]:

```
456976
```

4. Vectorizing ASM OPCODES BIGRAMS

In [6]:

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

Out[6]:

```
676
```

In [95]:

```python
vector = CountVectorizer(ngram_range=(2, 2), vocabulary = asmopcodebigram)
asm_opcode_bigram_vect = scipy.sparse.csr_matrix((4997, len(asmopcodebigram)))
raw_opcode = open('D:/opcode_file.txt').read().split('\n')
for indx in tqdm(range(4997)):
    asm_opcode_bigram_vect[indx, :] += scipy.sparse.csr_matrix(vector.transform([raw_opcode[indx]]))
```

```
100%|██████████████████████████████████████████████████████████████████████|
4997/4997 [02:41<00:00, 25.47it/s]
```

In [96]:

```python
asm_opcode_bigram_vect
```

Out[96]:

```
<4997x676 sparse matrix of type '<class 'numpy.float64'>'
 with 856179 stored elements in Compressed Sparse Row format>
```

In [97]:

```
scipy.sparse.save_npz('D:/asm_opcode_bigram_vect.npz', asm_opcode_bigram_vect)
```

In [98]:

```
op_bi_df = pd.SparseDataFrame(normalize(asm_opcode_bigram_vect, axis = 0), columns = asmopcodebigram)
```

In [99]:

```
op_bi_df.head()
```

Out[99]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	jmp inc	...	movzx cmp	movzx call	movzx shl	movzx ror
0	0.046114	0.005294	NaN	0.000578	NaN	0.003371	NaN	NaN	0.011986	NaN	...	NaN	NaN	NaN	NaN
1	NaN	0.000882	NaN	0.000289	0.000556	0.000595	0.001035	NaN	NaN	NaN	...	0.000654	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.007850	NaN	NaN	NaN
3	NaN	0.000138	NaN	0.000096	NaN	0.000397	0.001035	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	0.000524	0.001572	0.002237	0.000385	0.000556	0.000198	NaN	NaN	NaN	NaN	...	0.007196	NaN	NaN	NaN

5 rows × 676 columns

In [100]:

```
asm_opcode_binary_df = op_bi_df.fillna(0)
```

In [102]:

```
asm_opcode_binary_df['ID'] = final_asm.ID
```

In [103]:

```
asm_opcode_binary_df.head()
```

Out[103]:

	jmp jmp	jmp mov	jmp retf	jmp push	jmp pop	jmp xor	jmp retn	jmp nop	jmp sub	jmp inc	...	movzx call	movzx shl	movzx ror	movzx rol	m
0	0.046114	0.005294	0.000000	0.000578	0.000000	0.003371	0.000000	0.0	0.011986	0.0	...	0.0	0.0	0.0	0.0	
1	0.000000	0.000882	0.000000	0.000289	0.000556	0.000595	0.001035	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	
3	0.000000	0.000138	0.000000	0.000096	0.000000	0.000397	0.001035	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	
4	0.000524	0.001572	0.002237	0.000385	0.000556	0.000198	0.000000	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	

5 rows × 677 columns

In [104]:

```
asm_opcode_binary_df.shape
```

Out[104]:

(4997, 677)

In [105]:

```
asm_opcode_binary_df.to_csv("asm_opcode_binary_df.csv")
```

In [ ]:

```

```

5. Important features

In [40]:

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

6. Loading files

In [64]:

```
import joblib
import scipy
final_bytes_1 = pd.read_csv('D:/final_bytes_1.csv')
final_asm = pd.read_csv('D:/final_asm.csv')
image_dataframe = joblib.load('D:/image_dataframe')
asm_opcode_bigram_vect = scipy.sparse.load_npz('D:/asm_opcode_bigram_vect.npz')
asm_opcode_trigram_vect = scipy.sparse.load_npz('D:/asm_opcode_trigram_vect.npz')
asm_opcode_tetragram_vect = scipy.sparse.load_npz('D:/asm_opcode_tetragram_vect.npz')
```

In [65]:

```
image_dataframe = image_dataframe.drop(image_dataframe.index[451])
image_dataframe = image_dataframe.drop(image_dataframe.index[512])
image_dataframe = image_dataframe.drop(image_dataframe.index[1601])
image_dataframe['ID'] = final_asm.ID
image_dataframe.shape
```

Out[65]:

(4997, 201)

In [66]:

```
result_x = pd.merge(final_bytes_1,final_asm.drop(['Class'], axis=1),on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[66]:

	Unnamed: 0_x	ID	size_x	0	1	2	3	4	5	6	...	:dwo
0	0	01azqd4InC7m9JpocGv5	0.091636	0.527809	0.008309	0.002647	0.002067	0.002048	0.001835	0.002058	...	0.0349
1	1	01IsoiSMh5gxyDYTl4CB	0.120671	0.034861	0.017739	0.006813	0.003876	0.005303	0.003873	0.004747	...	0.0115
2	2	01jsnpXSAlgw6aPeDxrU	0.083910	0.081995	0.020303	0.002414	0.001315	0.005464	0.005280	0.005078	...	0.0072
3	3	01kcPWA9K2BOxQeS5Rju	0.010123	0.018495	0.002581	0.000682	0.000441	0.000770	0.000354	0.000310	...	0.0010
4	4	01SuzwMJEIXsK7A8dQbl	0.005594	0.017331	0.001511	0.000284	0.000234	0.000342	0.000232	0.000148	...	0.0097

5 rows × 310 columns

In [67]:

```
print(result_x.shape)
print(result_y.shape)
```

(4997, 310)
(4997,)

7. TOP 200 features from asm opcode bigram

In [42]:

```
from sklearn.preprocessing import normalize
```

In [54]:

```
opcode_bigram_index = imp_features(normalize(asm_opcode_bigram_vect, axis = 0), asmopcodebigram, 200)
```

In [57]:

```
asm_opcode_bigram_df = pd.SparseDataFrame(normalize(asm_opcode_bigram_vect, axis = 0), columns = asmopcode
bigram)
for col in asm_opcode_bigram_df.columns:
    if col not in np.take(asmopcodebigram, opcode_bigram_index):
        asm_opcode_bigram_df.drop(col, axis = 1, inplace = True)
```

In [61]:

```
asm_opcode_bigram_df = asm_opcode_bigram_df.to_dense()
```

In [62]:

```
asm_opcode_bigram_df.shape
```

Out[62]:

(4997, 200)

In [63]:

```
asm_opcode_bigram_df.head()
```

Out[63]:

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp dec	jmp add	jmp cmp	jmp jz	...	lea movzx	movzx mov	movz pus
0	0.046114	0.005294	0.000578	NaN	0.003371	0.011986	0.001871	0.023819	0.000151	NaN	...	NaN	NaN	Nal
1	NaN	0.000882	0.000289	0.000556	0.000595	NaN	NaN	0.002802	NaN	NaN	...	NaN	0.002972	0.00047
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.005430	NaN	0.00811
3	NaN	0.000138	0.000096	NaN	0.000397	NaN	NaN	NaN	NaN	NaN	...	0.000543	NaN	Nal
4	0.000524	0.001572	0.000385	0.000556	0.000198	NaN	NaN	NaN	0.000151	NaN	...	NaN	0.000283	Nal

5 rows × 200 columns

In [68]:

```
asm_opcode_bigram_df = asm_opcode_bigram_df.fillna(0)
```

In [69]:

```
asm_opcode_bigram_df['ID'] = result_x.ID
asm_opcode_bigram_df.head()
```

Out[69]:

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp dec	jmp add	jmp cmp	jmp jz	...	movzx mov	movzx push	movzx xor
0	0.046114	0.005294	0.000578	0.000000	0.003371	0.011986	0.001871	0.023819	0.000151	0.0	...	0.000000	0.000000	0.000000
1	0.000000	0.000882	0.000289	0.000556	0.000595	0.000000	0.000000	0.002802	0.000000	0.0	...	0.002972	0.000477	0.000000
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.008111	0.009354
3	0.000000	0.000138	0.000096	0.000000	0.000397	0.000000	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.000302
4	0.000524	0.001572	0.000385	0.000556	0.000198	0.000000	0.000000	0.000000	0.000151	0.0	...	0.000283	0.000000	0.000905

5 rows × 201 columns

In [70]:

```
asm_opcode_bigram_df.to_csv('asm_opcode_bigram_df.csv')
```

In [71]:

```
asm_opcode_bigram_df.shape
```

Out[71]:

(4997, 201)

8. Extracting Image features from ASM files

In [2]:

```
import array
import cv2
from PIL import Image
import cv2
import os
```

In [3]:

```
from tqdm import tqdm
```

In [4]:

```
for asmfile in tqdm(os.listdir("D:/microsoft_malware/train/byteFiles/asmFiles")):
        filename = asmfile.split('.')[0]
        file = open(os.path.join("D:/microsoft_malware/train/byteFiles/asmFiles/" + asmfile), 'rb')
#         print(file)
        filelen = os.path.getsize("D:/microsoft_malware/train/byteFiles/asmFiles/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        try:
            arr = array.array('B')
            arr.frombytes(file.read())
            file.close()
            reshaped = np.reshape(arr[:width * width], (width, width))
            reshaped = np.uint8(reshaped)
    #         reshaped.resize((1000,))
            cv2.imwrite('D:/asm_images_1/' + filename + '.png',reshaped)
        except Exception as e:
            print('File = {}'.format(filename))
```

```
 9%|▉         | 450/5000 [04:32<28:30,  2.66it/s]
File = 1F7jeznmCoiBxNdIKsWT
 10%|▉         | 512/5000 [05:27<13:15,  5.64it/s]
File = 1JZBS8ICkPfIYzXbOypc
 32%|▉         | 1601/5000 [16:52<35:14,  1.61it/s]
File = 4PLRF3BAbEtZTXnaGqfD
100%|▉         | 5000/5000 [53:25<00:00,  1.56it/s]
```

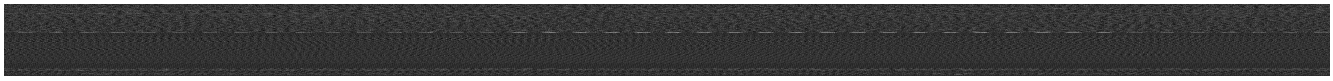
- Here in the above code we have added an exception handling block to check which files are not readable

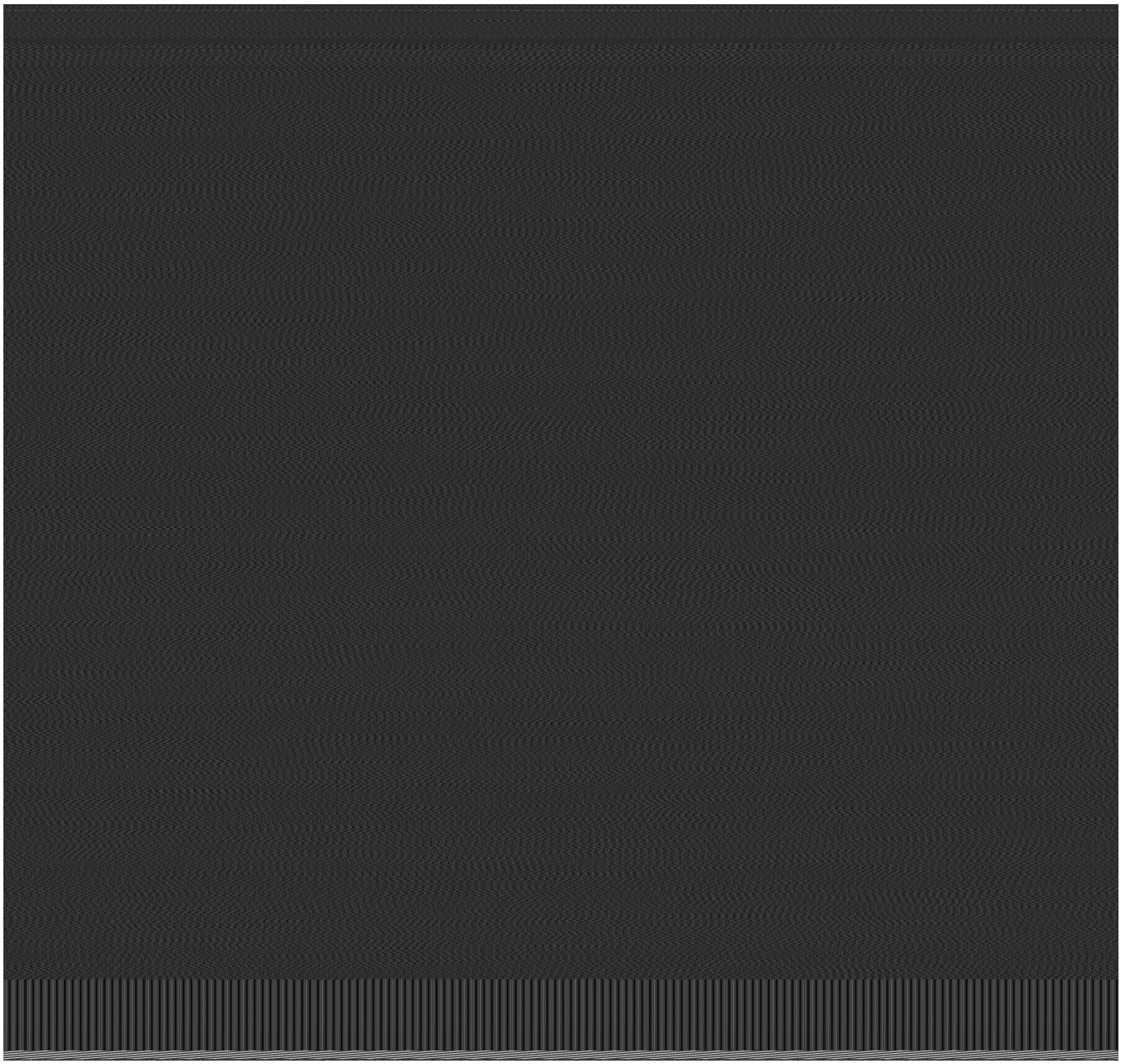
Viewing image

In [5]:

```
from IPython.display import Image
Image(filename='D:/asm_images_1/0A32eTdBKayjCWhZqDOQ.png')
```

Out[5]:





In [4]:

```
import cv2
imagefeatures = np.zeros((5000, 200))
```

In [5]:

```
def create_blank(width, height, rgb_color=(0, 0, 0)):
    """Create new image(numpy array) filled with certain color in RGB"""
    # Create black blank image
    image = np.zeros((height, width, 3), np.uint8)

    # Since OpenCV uses BGR, convert the color first
    color = tuple(reversed(rgb_color))
    # Fill image with color
    image[:] = color

    return image
```

In [6]:

```
for i, asmfile in tqdm(enumerate(os.listdir("D:/microsoft_malware/train/byteFiles/asmFiles"))):
    try:
        img = cv2.imread("D:/asm_images_1/" + asmfile.split('.')[0] + '.png')
        img_array = img.flatten()[:200]
#         imagefeatures[i, :] += img_array
    except Exception as e:
        img = create_blank(300,300,rgb_color = (0,0,0))
        img_array = img.flatten()[:200]
    imagefeatures[i, :] += img_array
```

```
5000it [24:00,  3.47it/s]
```

- for files which are not readable we have created a blank black image for those three files

In [13]:

```
imagefeatures
```

Out[13]:

```
array([[72., 72., 72., ..., 45., 45., 45.],
       [46., 46., 46., ..., 45., 45., 45.],
       [72., 72., 72., ..., 45., 45., 45.],
       ...,
       [46., 46., 46., ..., 45., 45., 45.],
       [46., 46., 46., ..., 45., 45., 45.],
       [72., 72., 72., ..., 45., 45., 45.]])
```

In [14]:

```
imagefeatures.shape
```

Out[14]:

```
(5000, 200)
```

In [23]:

```
type(imagefeatures)
```

Out[23]:

```
numpy.ndarray
```

In [35]:

```
from sklearn.preprocessing import normalize
```

In [34]:

```
image_dataframe = pd.DataFrame(data = normalize(imagefeatures[0:,0:]),
                               index = [i for i in range(imagefeatures.shape[0])],
                               columns = ['pixel '+str(i) for i in range(imagefeatures.shape[1])])
```

In [35]:

```
image_dataframe.head()
```

Out[35]:

	pixel 0	pixel 1	pixel 2	pixel 3	pixel 4	pixel 5	pixel 6	pixel 7	pixel 8	pixel 9	...	pixel 190	pixel 191	pix
0	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.0
1	0.060808	0.060808	0.060808	0.153341	0.153341	0.153341	0.133512	0.133512	0.133512	0.158629	...	0.059486	0.059486	0.0
2	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.0
3	0.107571	0.107571	0.107571	0.103089	0.103089	0.103089	0.097113	0.097113	0.097113	0.101595	...	0.067232	0.067232	0.0
4	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.0

5 rows × 200 columns

In [40]:

```
image_dataframe.iloc[451] #451,512,1601
```

Out[40]:

```
pixel 0    0.0
pixel 1    0.0
pixel 2    0.0
pixel 3    0.0
pixel 4    0.0
pixel 5    0.0
pixel 6    0.0
```

```
pixel 7      0.0
pixel 8      0.0
pixel 9      0.0
pixel 10     0.0
pixel 11     0.0
pixel 12     0.0
pixel 13     0.0
pixel 14     0.0
pixel 15     0.0
pixel 16     0.0
pixel 17     0.0
pixel 18     0.0
pixel 19     0.0
pixel 20     0.0
pixel 21     0.0
pixel 22     0.0
pixel 23     0.0
pixel 24     0.0
pixel 25     0.0
pixel 26     0.0
pixel 27     0.0
pixel 28     0.0
pixel 29     0.0
            ...
pixel 170    0.0
pixel 171    0.0
pixel 172    0.0
pixel 173    0.0
pixel 174    0.0
pixel 175    0.0
pixel 176    0.0
pixel 177    0.0
pixel 178    0.0
pixel 179    0.0
pixel 180    0.0
pixel 181    0.0
pixel 182    0.0
pixel 183    0.0
pixel 184    0.0
pixel 185    0.0
pixel 186    0.0
pixel 187    0.0
pixel 188    0.0
pixel 189    0.0
pixel 190    0.0
pixel 191    0.0
pixel 192    0.0
pixel 193    0.0
pixel 194    0.0
pixel 195    0.0
pixel 196    0.0
pixel 197    0.0
pixel 198    0.0
pixel 199    0.0
Name: 451, Length: 200, dtype: float64
```

In [47]:

```
import joblib
```

In [48]:

```
joblib.dump(image_dataframe, 'image_datafr-ame')
```

Out[48]:

```
['image_dataframe']
```

In [ ]:

```

```

In [ ]:

```

```