Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- **Source : https://www.kaggle.com/c/malware-classification/data**
- **For every malware, we have two files**
  1. **.asm file (read more: https://www.reviversoft.com/file-extensions/asm)**
  2. **.bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)**

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

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,
s: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 c
f_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 c
f_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 c
f_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    ; operato
delete(void *)
.text:0040104B 83 C4 04                            add     esp, 4
.text:0040104E
.text:0040104E                         loc_40104E:                 ; CODE XREF:
.text:00401043 j
.text:0040104E 8B C6                               mov     eax, esi
.text:00401050 5E                              pop     esi
.text:00401051 C2 04 00                            retn    4
.text:00401051                         ; ------------------------------------------------
------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
```

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

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

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y

https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_plB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [1]:

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

%matplotlib inline
```

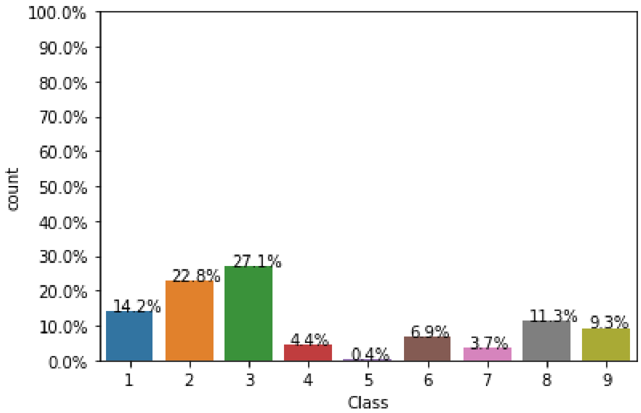
In [2]:

```
#separating byte files and asm files

source = r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\train'
destination = r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\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 wil
l 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,r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\asmFiles'
)
    source=r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\asmFiles'
    data_files = os.listdir(source)
    for file in data_files:
        if (file.endswith("bytes")):
            shutil.move(source+"\\"+file,destination+"\\"+file)
```

3.1. Distribution of malware classes in whole data set

In [10]:

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

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

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



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [4]:

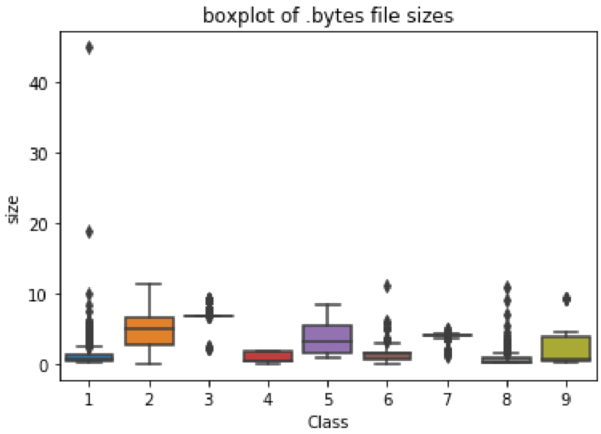
```
#file sizes of byte files
byteFilesPath = r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\byteFiles'
files=os.listdir(byteFilesPath)
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # 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(byteFilesPath+'\\'+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  4.234863      9
1  01IsoiSMh5gxyDYTl4CB  5.538818      2
2  01jsnpXSAlgw6aPeDxrU  3.887939      9
3  01kcPWA9K2BOxQeS5Rju  0.574219      1
4  01SuzwMJEIXsK7A8dQbl  0.370850      8
```

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

In [5]:

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



3.2.3 feature extraction from byte files

In [6]:

```
#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(byteFilesPath)
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        f=file.split('.')[0]
        text_file = open(byteFilesPath+'\\'+f+".txt", 'w+')
        with open(byteFilesPath+'\\'+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(byteFilesPath+'\\'+file)
        text_file.close()
```

In [7]:

```
files = os.listdir(byteFilesPath)
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,
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,
e,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,5
,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
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,
5,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,c
,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
```

```
,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
ea,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open(byteFilesPath+'\\'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i, row in enumerate(feature_matrix[k]):
        if i!=len(feature_matrix[k])-1:
            byte_feature_file.write(str(row)+",")
        else:
            byte_feature_file.write(str(row))
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [8]:

```
byte_features=pd.read_csv("result.csv")
```

In [9]:

```
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)
```

Out[9]:

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

2 rows × 258 columns

In [10]:

```
result = pd.merge(byte_features, data_size_byte,on='ID', how='left')
result.head()
```

Out[10]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211	3097	2758	3099	2759	57
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281	302	7639	518	17001	549
2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2242	2885	2863	2471	2786	2680	491
3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	1078	...	485	462	516	1133	471	761	79
4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	422	...	350	209	239	653	221	242	21

5 rows × 260 columns

In [11]:

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

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

In [12]:

```
type(result)
```

Out[12]:

```
pandas.core.frame.DataFrame
```

In [13]:

```
result.to_csv("byteFeatures.csv")
```

In [14]:

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

Out[14]:

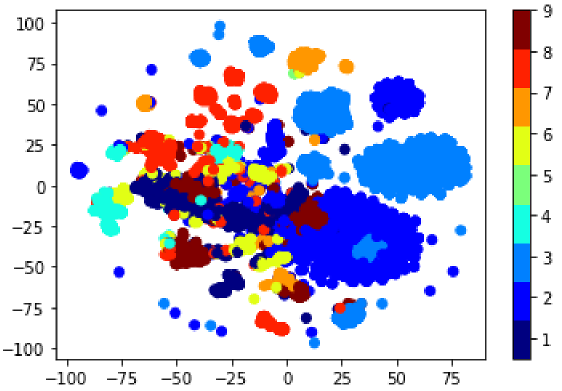
	ID	0	1	2	3	4	5	6	7	8	...	f9
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.013560
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.001920
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434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

5 rows × 260 columns

3.2.4 Multivariate Analysis

In [15]:

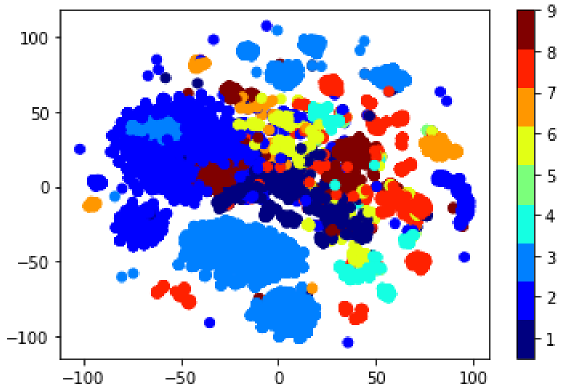
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))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



In [16]:

In [16]:

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



Train Test split

In [17]:

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

In [18]:

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

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

In [19]:

```
y_train.value_counts().sort_index()
```

Out[19]:

```
1     986
2    1586
3    1883
4     304
5      27
6     481
7     254
8     786
9     648
Name: Class, dtype: int64
```

In [20]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
```

```
train_class_distribution = y_train.value_counts().sort_index()
test_class_distribution = y_test.value_counts().sort_index()
cv_class_distribution = y_cv.value_counts().sort_index()
```

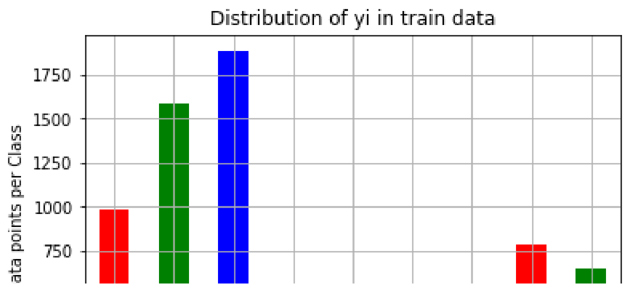
In [21]:

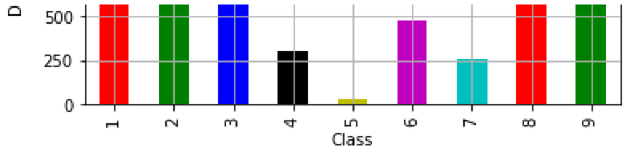
```
print(type(train_class_distribution))
```

```
<class 'pandas.core.series.Series'>
```

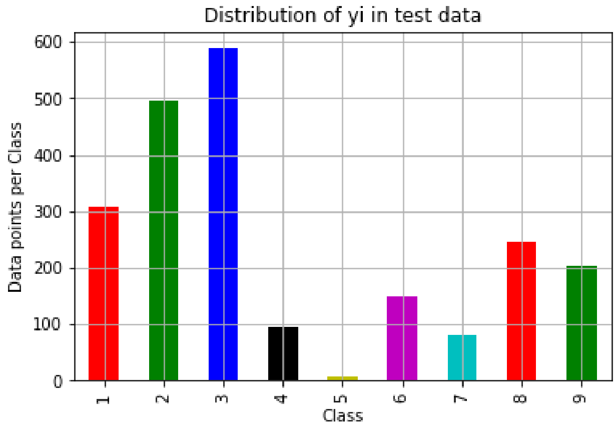
In [22]:

```
#my_colors = 'rgbkymc'
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']  # red, green, blue, black, etc.
#my_colormap = ListedColormap(my_colors)
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.ro
und((train_class_distribution.values[i]/y_train.shape[0]*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.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.rou
nd((test_class_distribution.values[i]/y_test.shape[0]*100), 3), '%)')

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

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

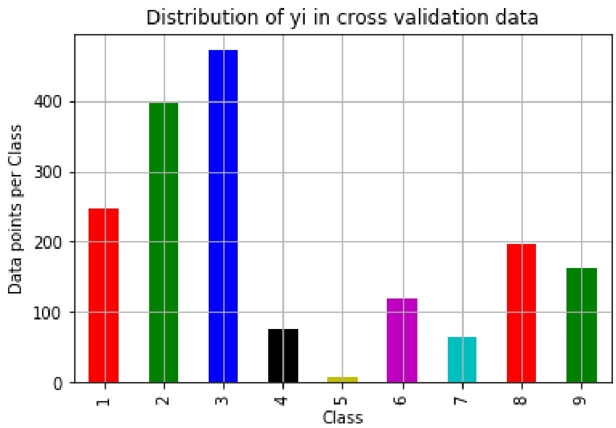





```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 (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




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




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

In [67]:

```
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)))
    # 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

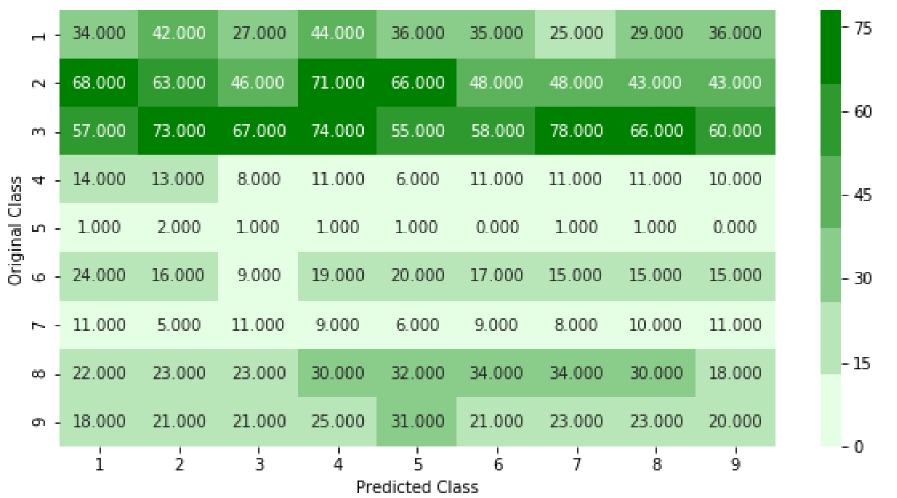
4.1.1. Random Model

In [53]:

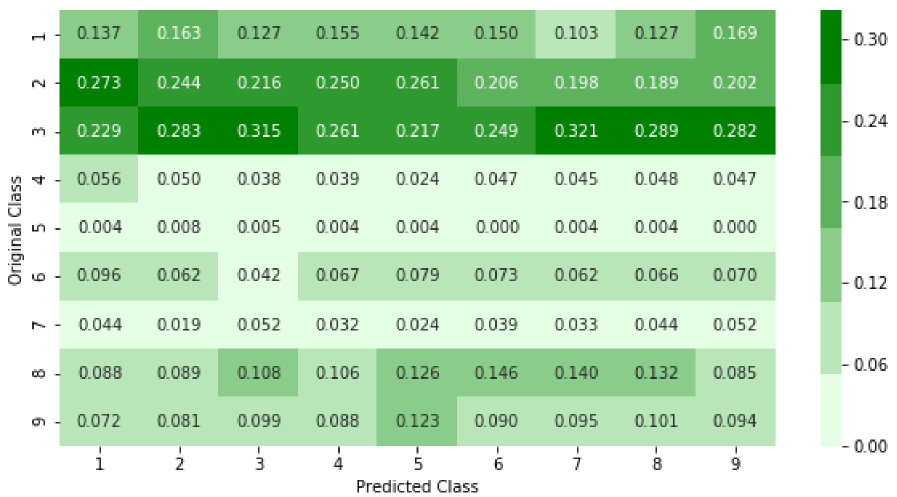
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
```

```
# 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 2.476030530408164
Log loss on Test Data using Random Model 2.474610533684162
Number of misclassified points  88.45446182152715
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

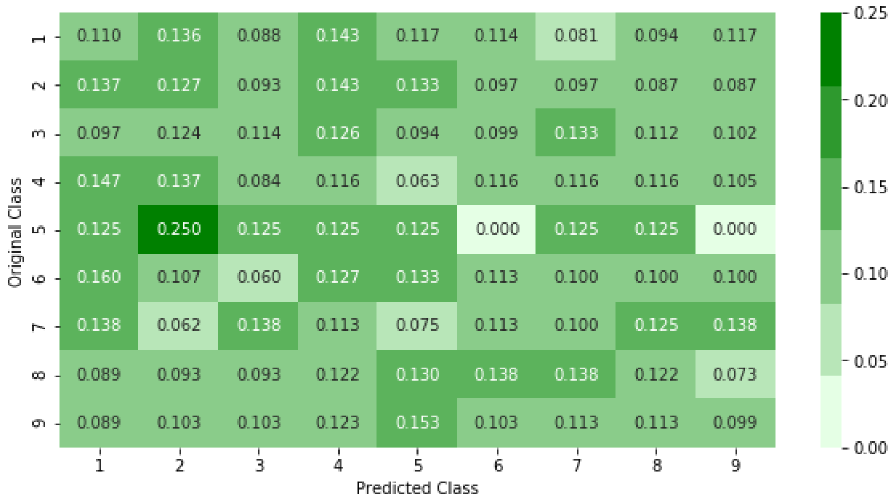



```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```




```
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.]
```

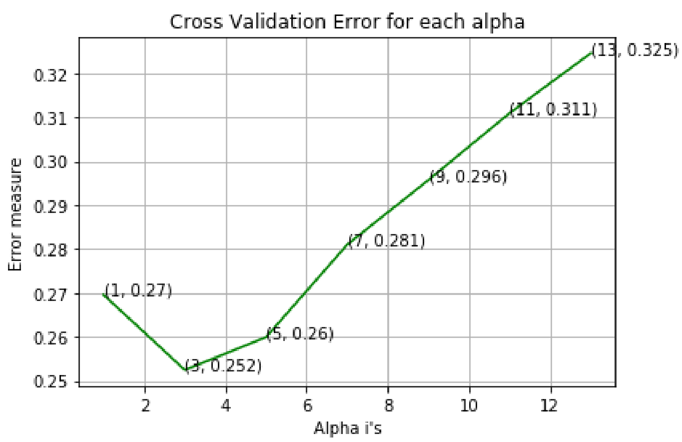
4.1.2. K Nearest Neighbour Classification

In [54]:

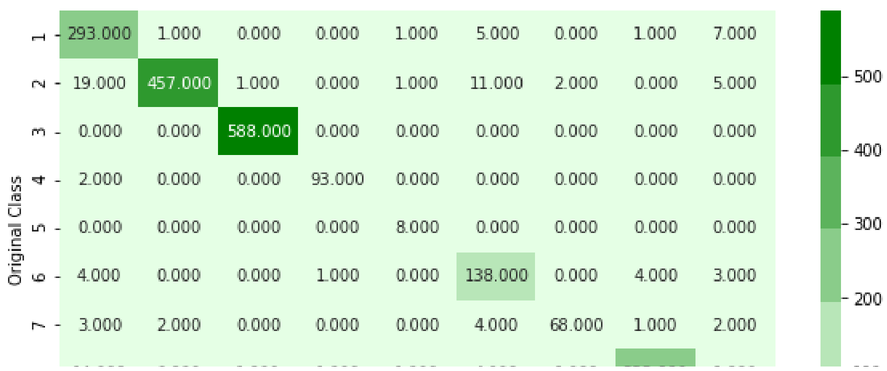
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

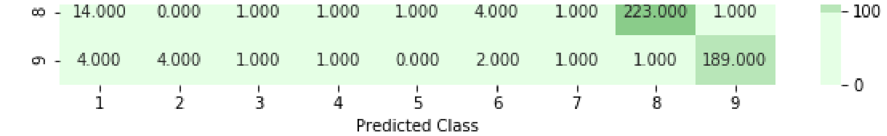
```
best_alpha = np.argmin(cv_log_error_array)

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_lo
ss(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, p
redict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for k =  1 is 0.26953571002174403
log_loss for k =  3 is 0.25241110434291286
log_loss for k =  5 is 0.2598651031372791
log_loss for k =  7 is 0.2809598930600912
log_loss for k =  9 is 0.29568483392687506
log_loss for k =  11 is 0.3109352182240991
log_loss for k =  13 is 0.32459327230406676
```

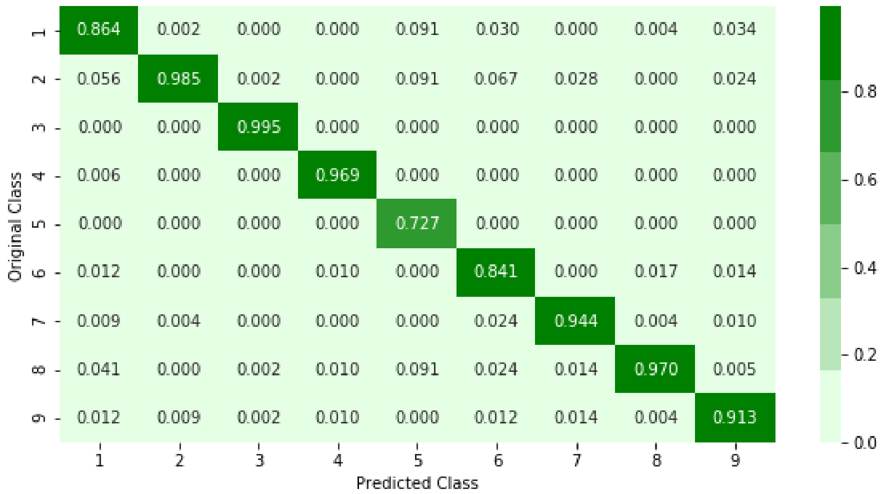



```
For values of best alpha =  3 The train log loss is: 0.124923851213006
For values of best alpha =  3 The cross validation log loss is: 0.25241110434291286
For values of best alpha =  3 The test log loss is: 0.22032851341727705
Number of misclassified points  5.381784728610856
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

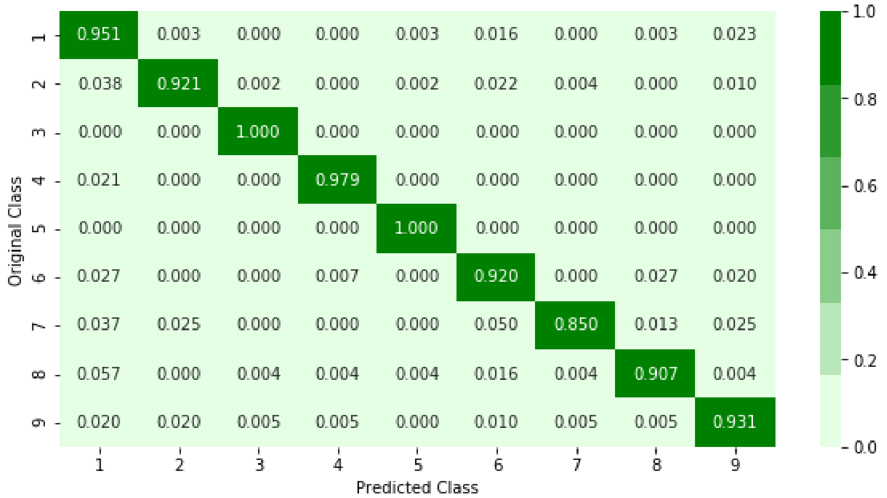





```
-------------------------------------------------- Precision matrix ------------------------------------------------
```




```
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.]
```

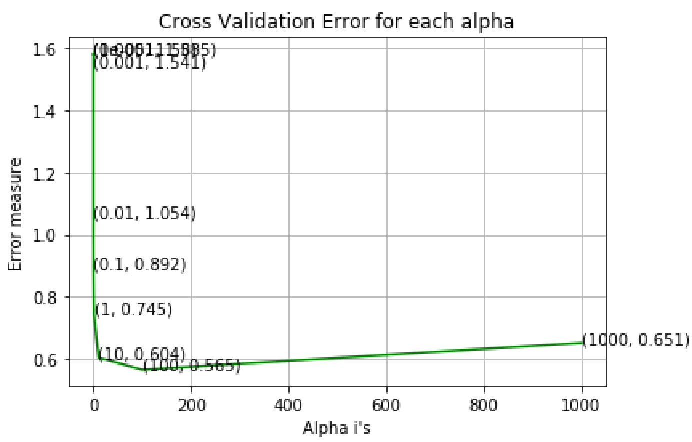
4.1.3. Logistic Regression

In [55]:

```
# 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.
```

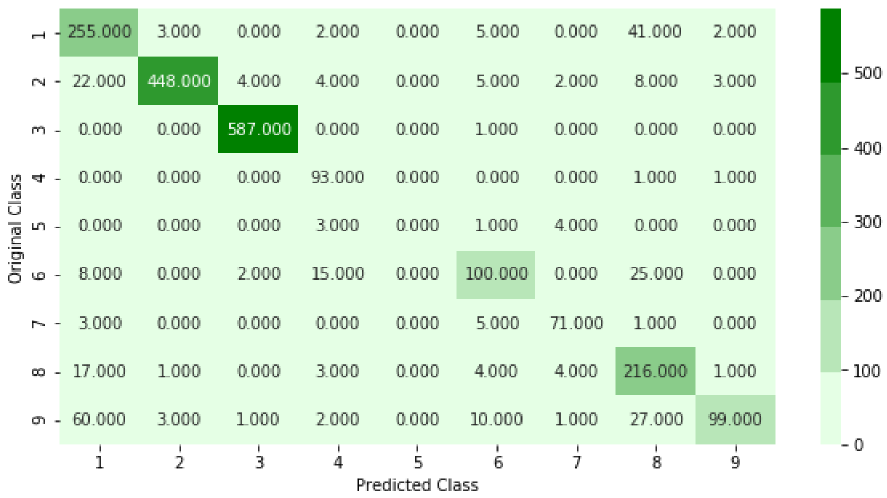
```python
#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------

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

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  1e-05 is 1.580380287612864
log_loss for c =  0.0001 is 1.5846375424548838
log_loss for c =  0.001 is 1.5413607933803408
log_loss for c =  0.01 is 1.0541268303097064
log_loss for c =  0.1 is 0.8921235339271195
log_loss for c =  1 is 0.745031041393818
log_loss for c =  10 is 0.6040037626555247
log_loss for c =  100 is 0.5646925748221535
log_loss for c =  1000 is 0.6509275991878499
```

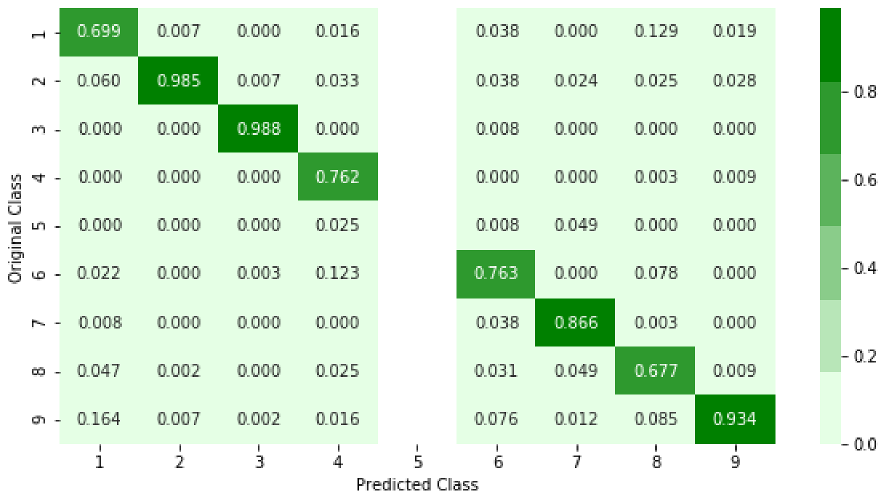



```
log loss for train data 0.5044199758666117
log loss for cv data 0.5646925748221535
```

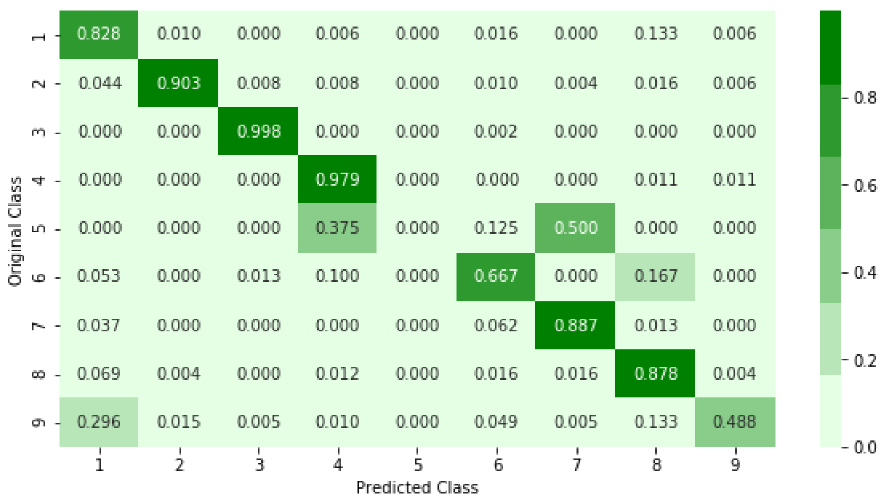
```
log loss for test data 0.5415682647192315
Number of misclassified points  14.029438822447101
-------------------------------------------------- Confusion matrix --------------------------------
----------------
```




```
-------------------------------------------------- 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.]
```

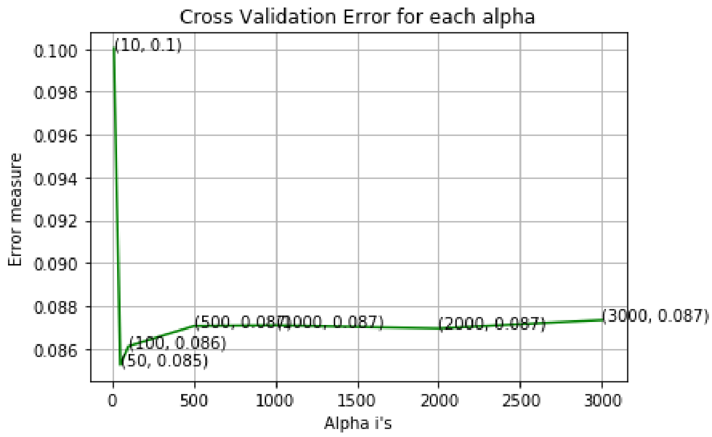
4.1.4. Random Forest Classifier

In [56]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_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-fores
t-and-their-construction-2/
# --------------------------------

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

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


best_alpha = np.argmin(cv_log_error_array)

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_lo
ss(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, p
redict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.1000315162034714
```

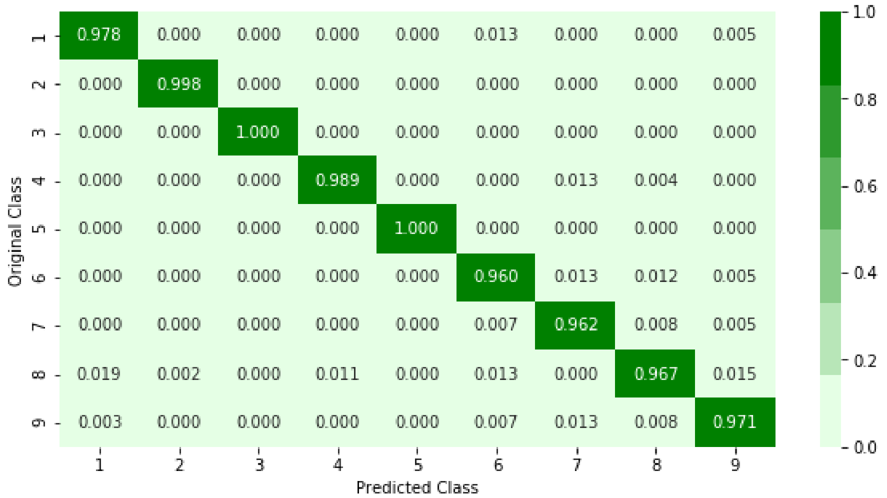
```
log_loss for c =  50 is 0.08523657478245217
log_loss for c =  100 is 0.08611140205729609
log_loss for c =  500 is 0.08705346720614689
log_loss for c =  1000 is 0.08708681392443493
log_loss for c =  2000 is 0.08694242817420156
log_loss for c =  3000 is 0.08733226441761843
```




```
For values of best alpha =  50 The train log loss is: 0.03010758441022165
For values of best alpha =  50 The cross validation log loss is: 0.08523657478245217
For values of best alpha =  50 The test log loss is: 0.08155079362840542
Number of misclassified points  1.4719411223551058
-------------------------------------------------- Confusion matrix -------------------------------
----------------
```

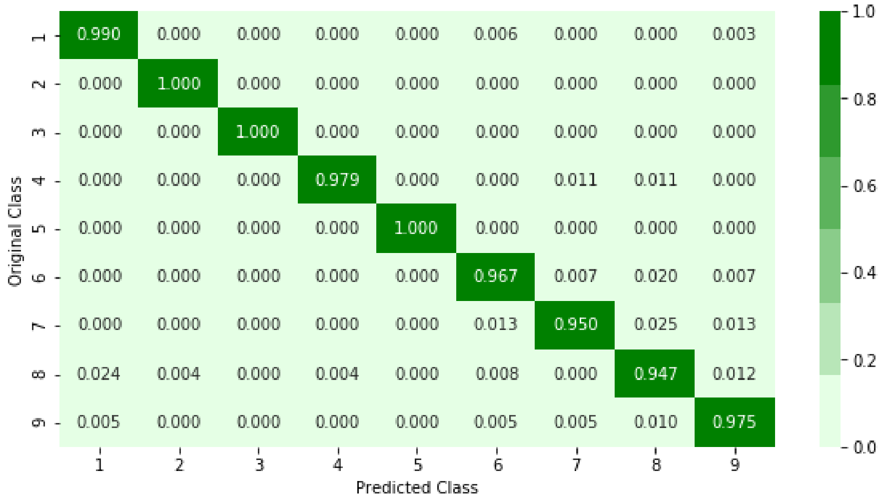
	1	2	3	4	5	6	7	8	9
1	305.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000	1.000
2	0.000	496.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	588.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	93.000	0.000	0.000	1.000	1.000	0.000
5	0.000	0.000	0.000	0.000	8.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	145.000	1.000	3.000	1.000
7	0.000	0.000	0.000	0.000	0.000	1.000	76.000	2.000	1.000
8	6.000	1.000	0.000	1.000	0.000	2.000	0.000	233.000	3.000
9	1.000	0.000	0.000	0.000	0.000	1.000	1.000	2.000	198.000

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



	1	2	3	4	5	6	7	8	9
1	0.978	0.000	0.000	0.000	0.000	0.013	0.000	0.000	0.005
2	0.000	0.998	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.989	0.000	0.000	0.013	0.004	0.000
5	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.960	0.013	0.012	0.005
7	0.000	0.000	0.000	0.000	0.000	0.007	0.962	0.008	0.005
8	0.019	0.002	0.000	0.011	0.000	0.013	0.000	0.967	0.015
9	0.003	0.000	0.000	0.000	0.000	0.007	0.013	0.008	0.971

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



	1	2	3	4	5	6	7	8	9
1	0.990	0.000	0.000	0.000	0.000	0.006	0.000	0.000	0.003
2	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.979	0.000	0.000	0.011	0.011	0.000
5	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.967	0.007	0.020	0.007
7	0.000	0.000	0.000	0.000	0.000	0.013	0.950	0.025	0.013
8	0.024	0.004	0.000	0.004	0.000	0.008	0.000	0.947	0.012
9	0.005	0.000	0.000	0.000	0.000	0.005	0.005	0.010	0.975

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

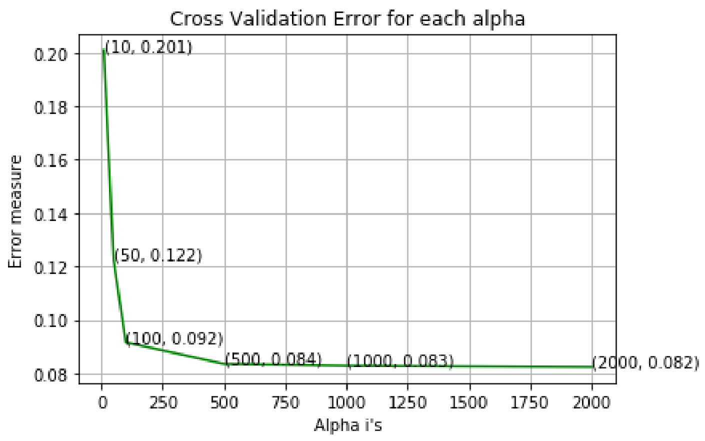
4.1.5. XgBoost Classification

In [57]:

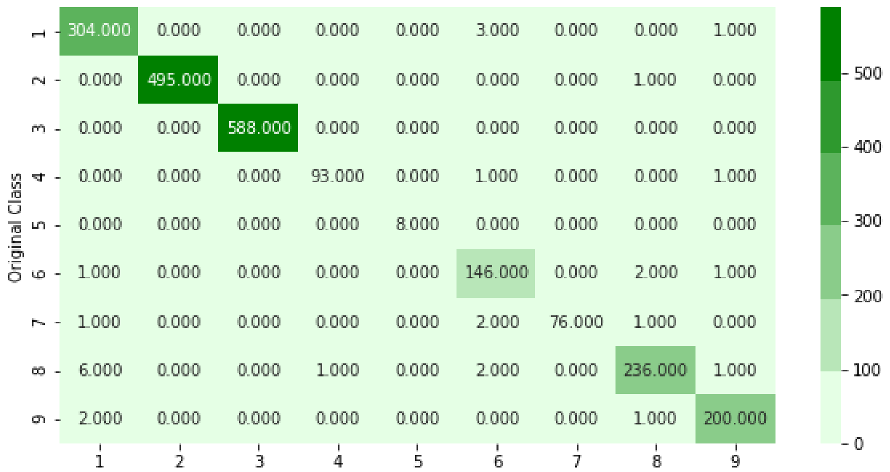
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# 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, verbo
se=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 no
t 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-en
sembles/
# -----------------------

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_lo
ss(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, p
redict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.20093744193721913
log_loss for c =  50 is 0.12244928417825396
log_loss for c =  100 is 0.09158545674673776
log_loss for c =  500 is 0.08357614645754222
log_loss for c =  1000 is 0.08286779907905514
log_loss for c =  2000 is 0.08238022931330571
```

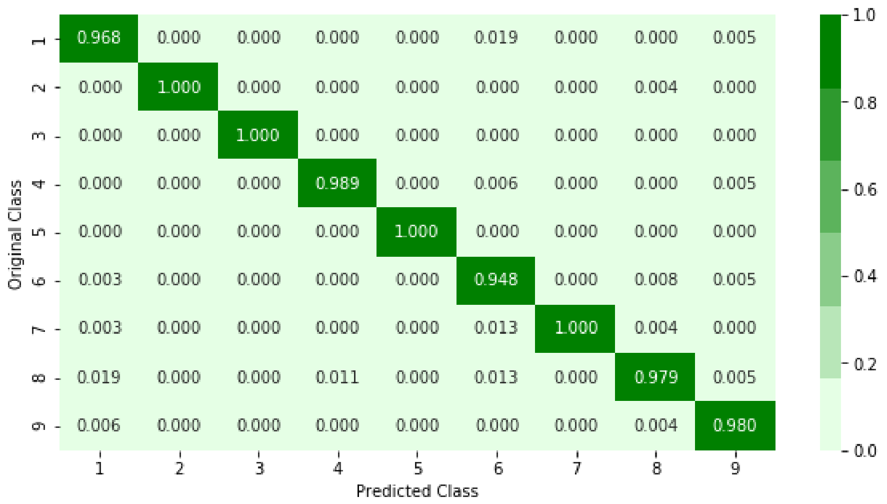



```
For values of best alpha =  2000 The train log loss is: 0.02403062392225225
For values of best alpha =  2000 The cross validation log loss is: 0.08238022931330571
For values of best alpha =  2000 The test log loss is: 0.06634036300246719
Number of misclassified points  1.2879484820607177
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

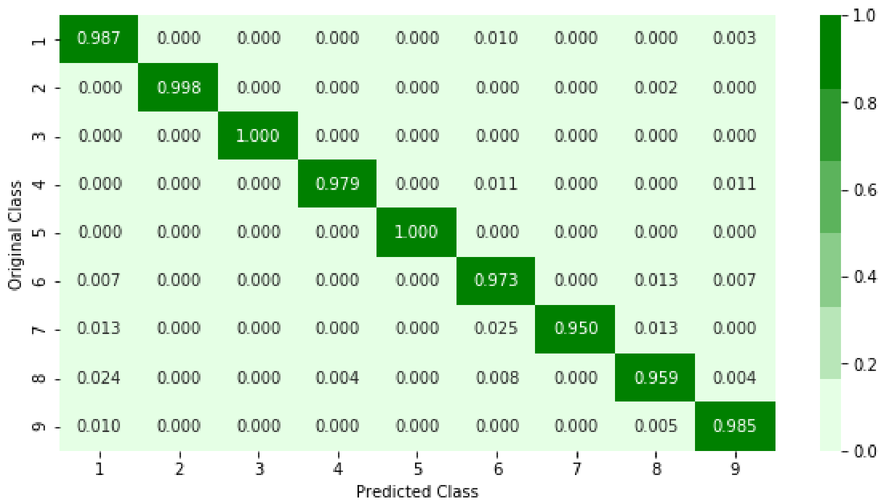


Predicted Class

```
-------------------------------------------------- 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 [58]:

```python
# 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)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  7.7min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 19.2min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 38.1min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 52.7min remaining:  5.9min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 62.4min finished
```

Out[58]:

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

In [59]:

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

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

In [60]:

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

# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.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, verbo
se=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 no
t 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-en
sembles/
# -----------------------

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))
```

```
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 0.023804923889745738
cv loss 0.08175097446620581
test loss 0.06762258334979357
```

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 a
ll the cores that are present in our computer.


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

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

4.2.1 Feature extraction from asm files

- **To extract the unigram features from the .asm files we need to process ~150GB of data**
- **We will provide you the output file of these two cells, which you can directly use it**

In [11]:

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

Out[11]:

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

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [12]:

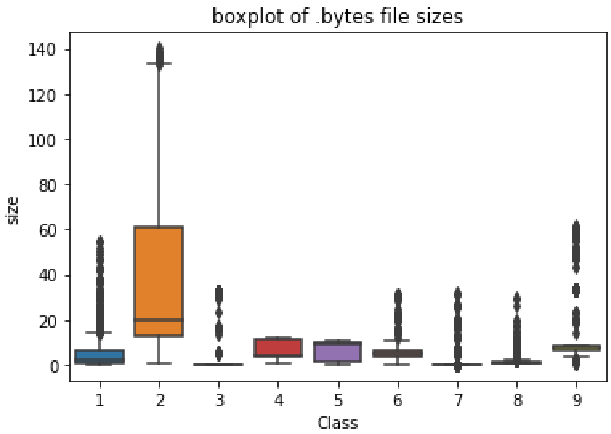
```
#file sizes of byte files
asmFiles_dir = r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\asmFiles'
files=os.listdir(asmFiles_dir)
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1,
st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat(asmFiles_dir+'\\'+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
```

4.2.1.2 Distribution of .asm file sizes

In [63]:

```
#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 [13]:

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

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

Out[13]:

Out[13]:

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

5 rows × 54 columns

In [14]:

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

Out[14]:

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

5 rows × 54 columns

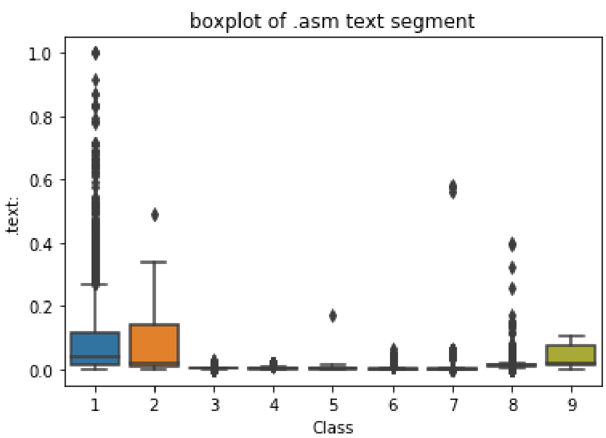
In [15]:

```
result_asm.to_csv('asmFeatures.csv')
```

4.2.2 Univariate analysis on asm file features

In [66]:

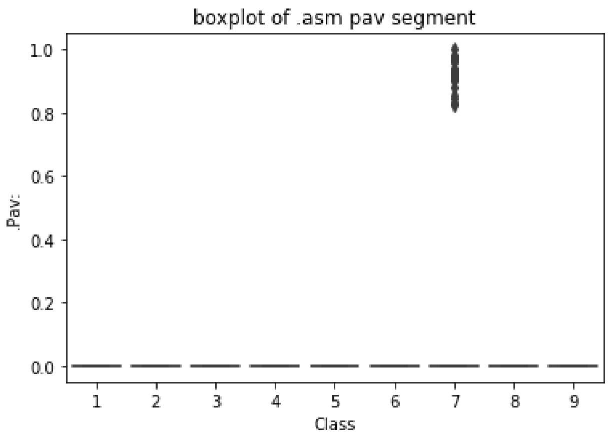
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```




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

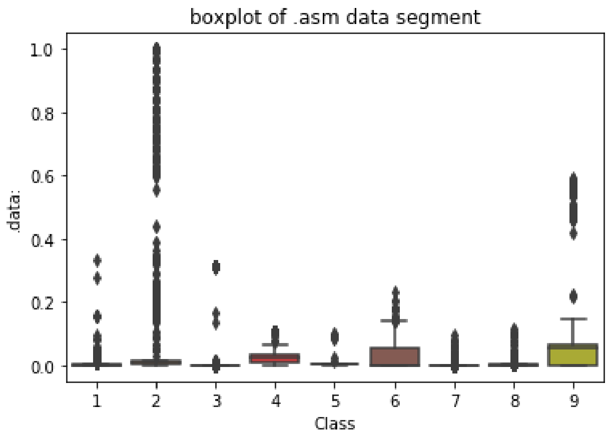
In [67]:

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



In [68]:

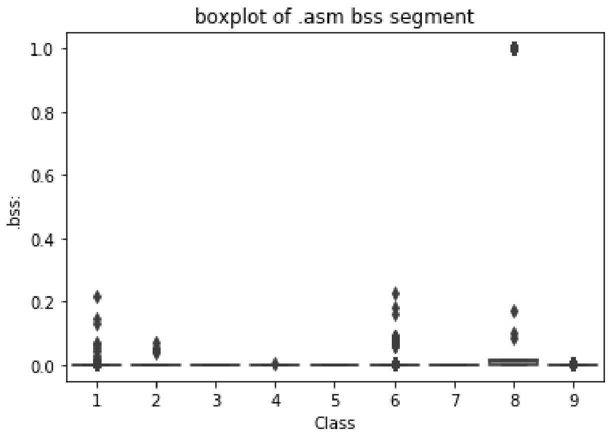
```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```



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

In [69]:

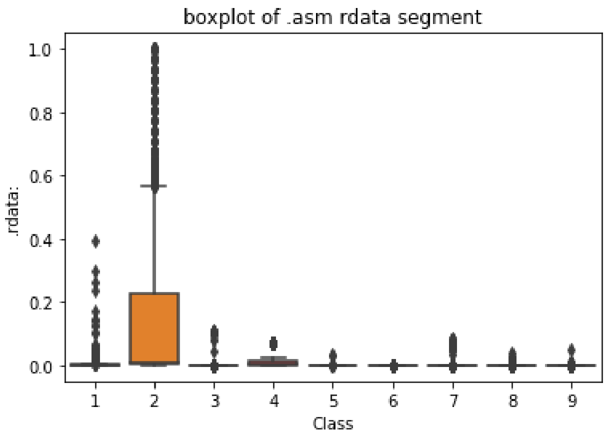
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



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

In [70]:

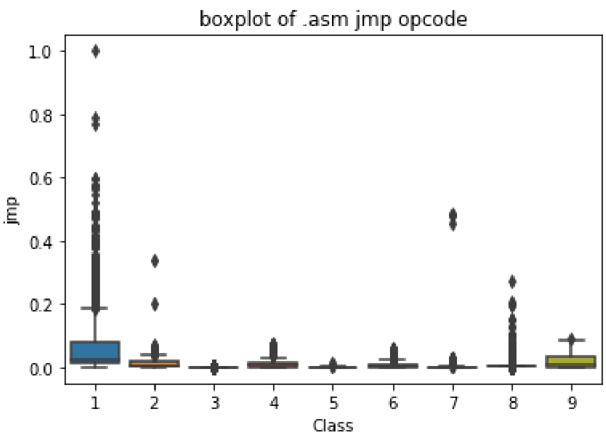
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

In [71]:

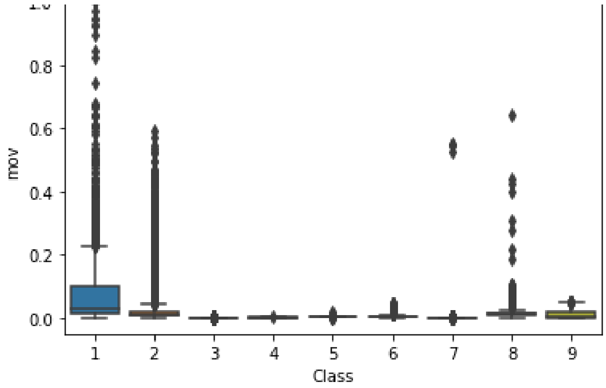
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



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

In [72]:

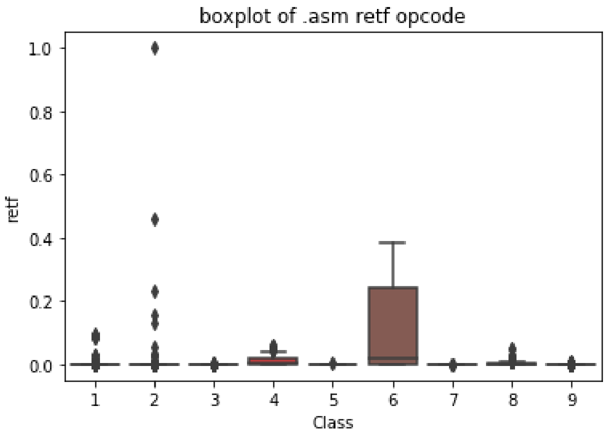
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```



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

In [73]:

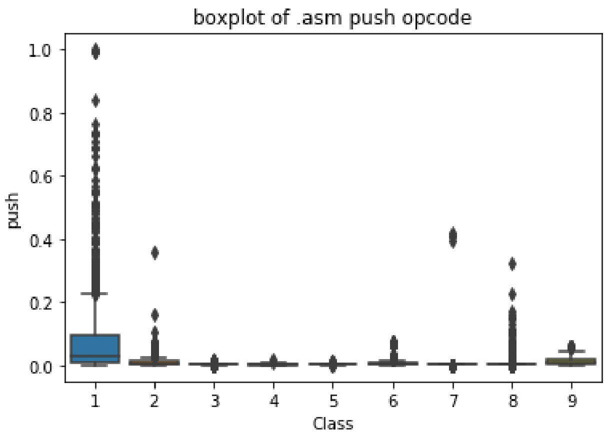
```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```



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

In [74]:

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

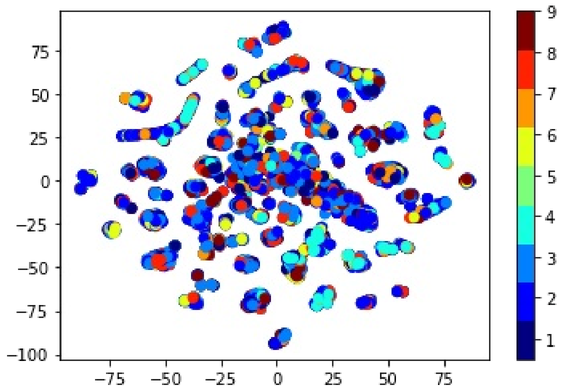


plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

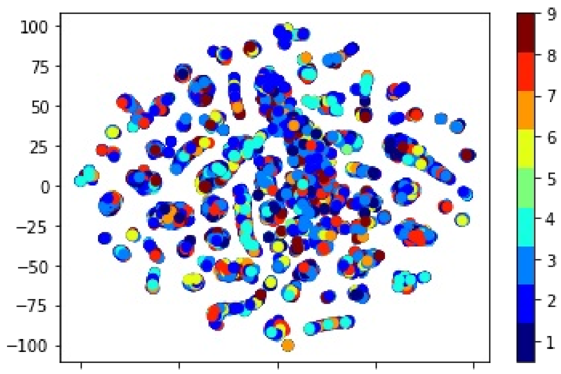
In [75]:

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

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



In [76]:

```
# 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 tho
se features
# the plot looks very messy

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



−100 −50 0 50 100

```
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 [19]:

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

In [20]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,tes
t_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 [21]:

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

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

```
.dll       False
std::      False
:dword     False
edx        False
esi        False
eax        False
ebx        False
ecx        False
edi        False
ebp        False
esp        False
eip        False
size       False
dtype: bool
```

4.4. Machine Learning models on features of .asm files

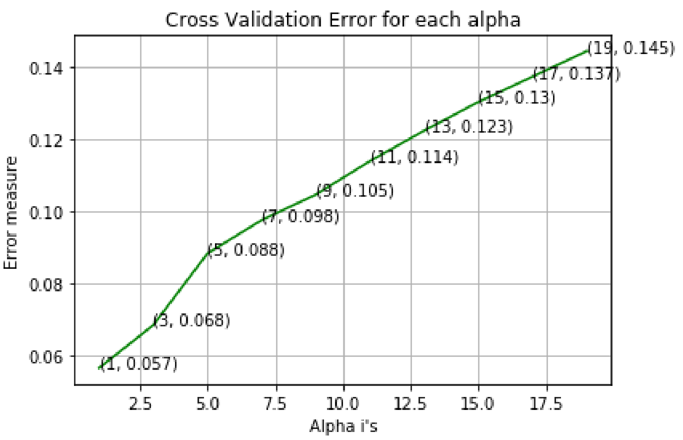
4.4.1 K-Nearest Neigbors

In [80]:

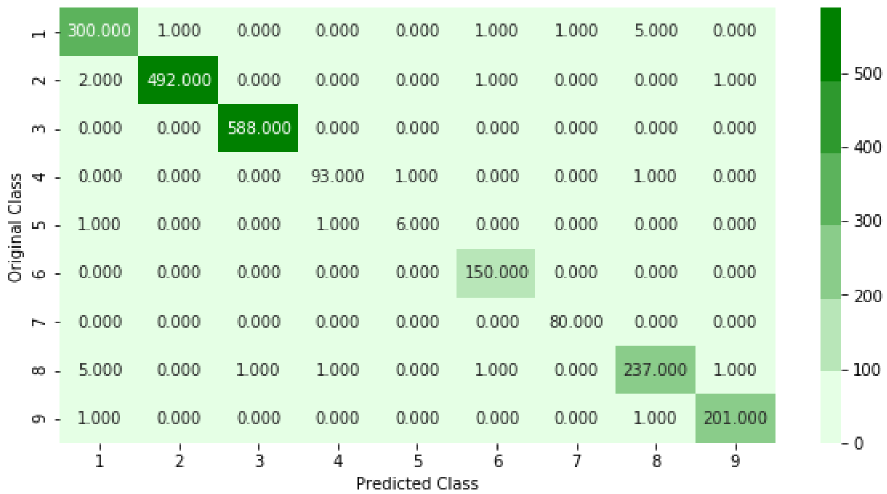
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

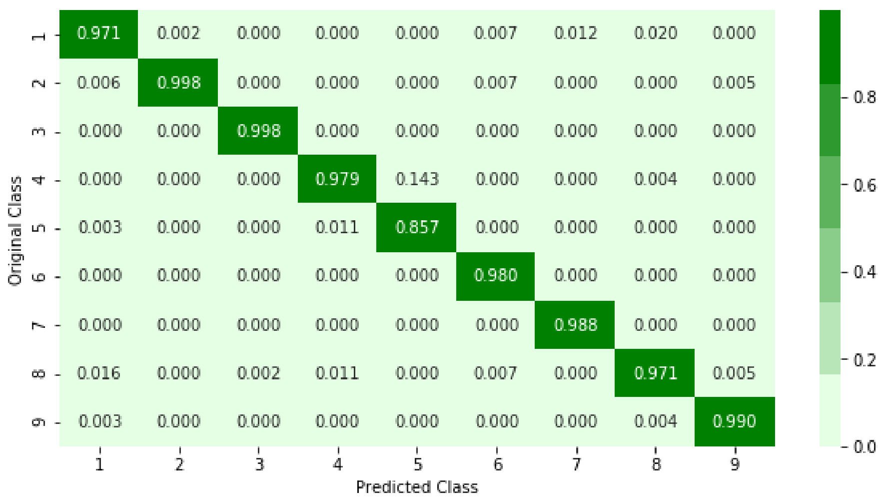
```
log_loss for k =  1 is 0.0565592329378145
log_loss for k =  3 is 0.0684782244419754
log_loss for k =  5 is 0.08836709946643373
log_loss for k =  7 is 0.09763839608888916
log_loss for k =  9 is 0.1047187910415532
log_loss for k =  11 is 0.11403313836469549
log_loss for k =  13 is 0.12250470014645061
log_loss for k =  15 is 0.13043368043867534
log_loss for k =  17 is 0.13746052242074216
log_loss for k =  19 is 0.14454649140816156
```



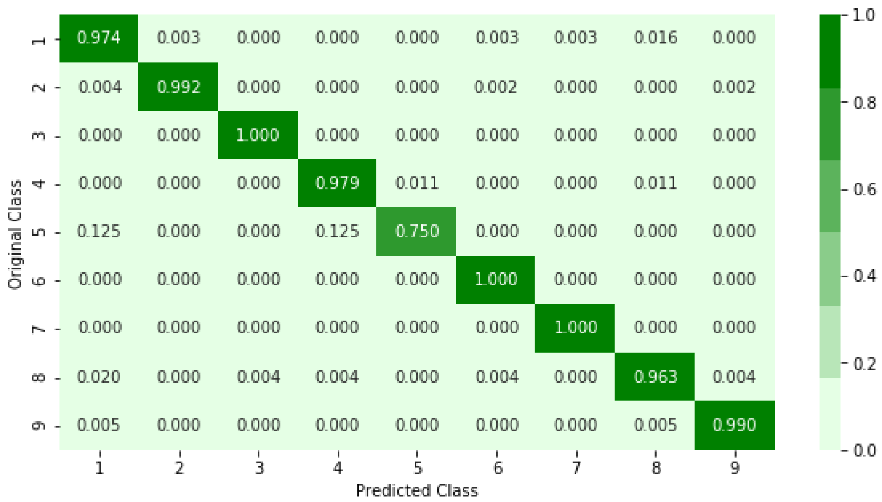

```
log loss for train data 0.028117982660904615
log loss for cv data 0.0565592329378145
log loss for test data 0.07839497751954067
Number of misclassified points  1.2419503219871204
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```



```
-------------------------------------------------- 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.]
```

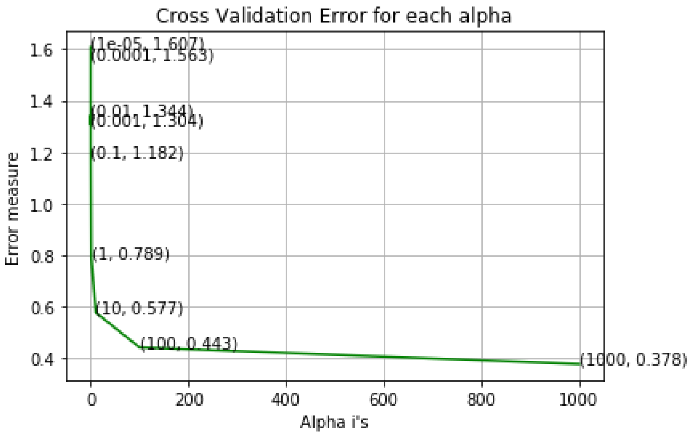
4.4.2 Logistic Regression

In [81]:

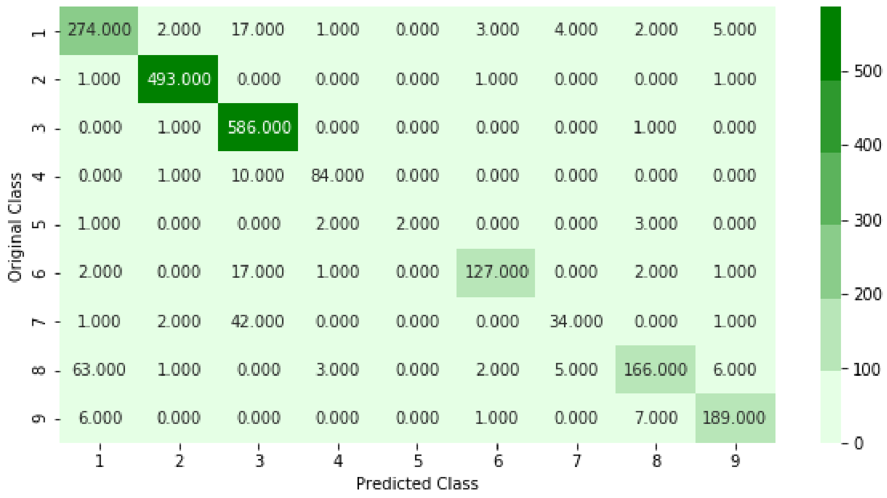
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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=1
e-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))
```

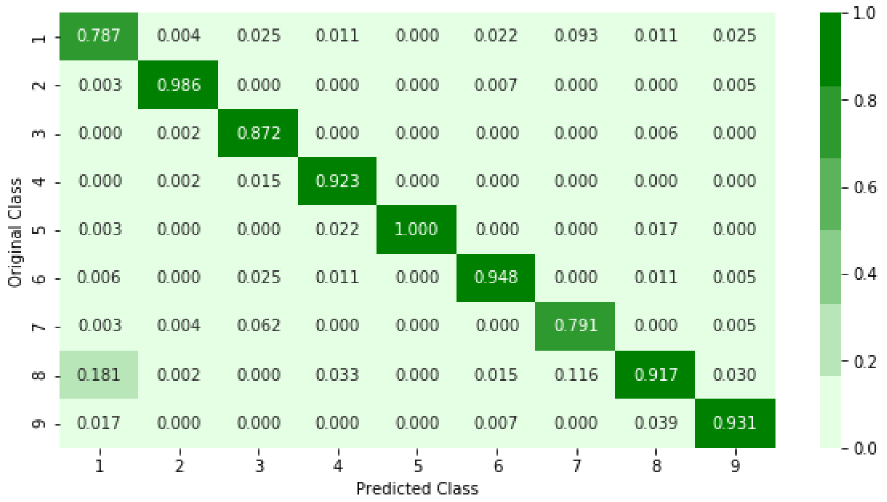
```
log_loss for c =  1e-05 is 1.6069702508942914
log_loss for c =  0.0001 is 1.5632898224114213
log_loss for c =  0.001 is 1.3044990345111
log_loss for c =  0.01 is 1.3440398120308317
log_loss for c =  0.1 is 1.1820878465469766
log_loss for c =  1 is 0.788612629295657
log_loss for c =  10 is 0.5773638185870443
log_loss for c =  100 is 0.44258192751883385
log_loss for c =  1000 is 0.37811608444374034
```



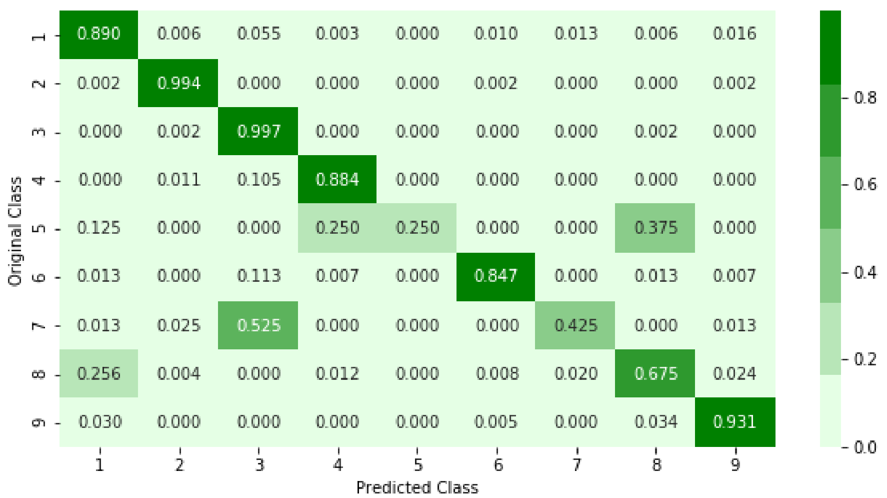

```
log loss for train data 0.3492784628579213
log loss for cv data 0.37811608444374034
log loss for test data 0.3747357754031491
Number of misclassified points  10.073597056117755
-------------------------------------------------- Confusion matrix ------------------------------
```




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




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




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

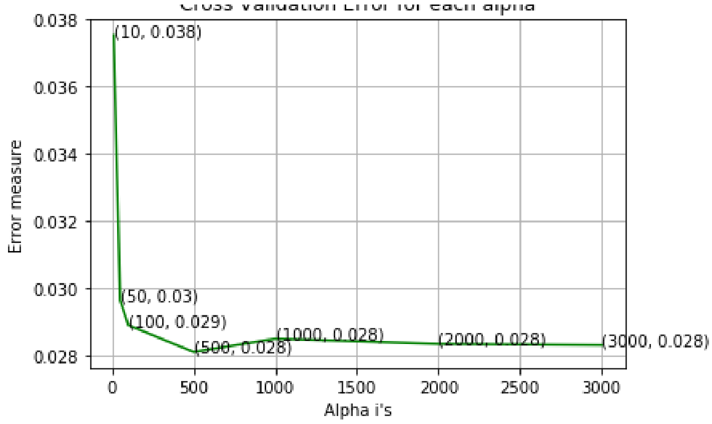
4.4.3 Random Forest Classifier

In [82]:

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

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)

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

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

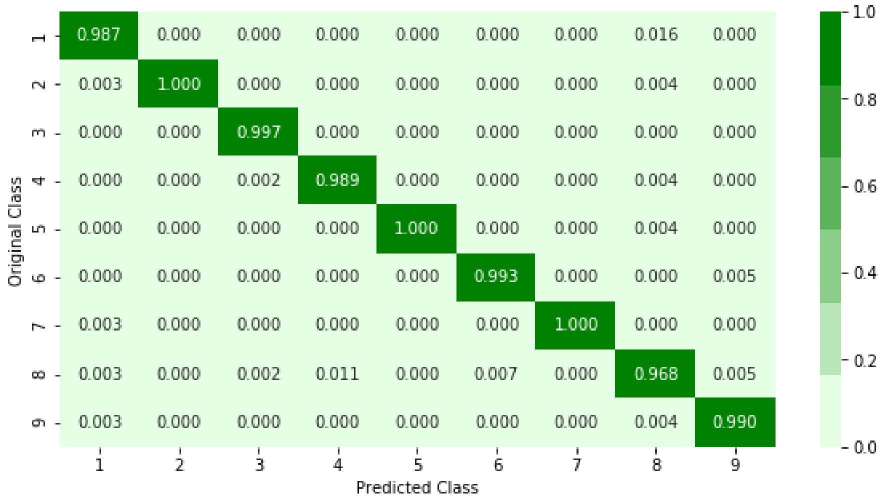
```
log_loss for c =  10 is 0.037534860440192164
log_loss for c =  50 is 0.029639055626223078
log_loss for c =  100 is 0.02889974259916546
log_loss for c =  500 is 0.028101889950011717
log_loss for c =  1000 is 0.0284973295613931
log_loss for c =  2000 is 0.0283425072456327
log_loss for c =  3000 is 0.028308425235938173
```

Cross Validation Error for each alpha

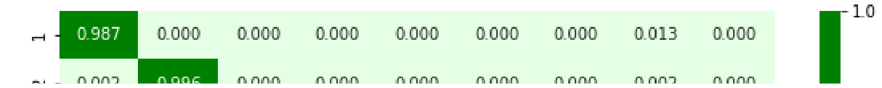


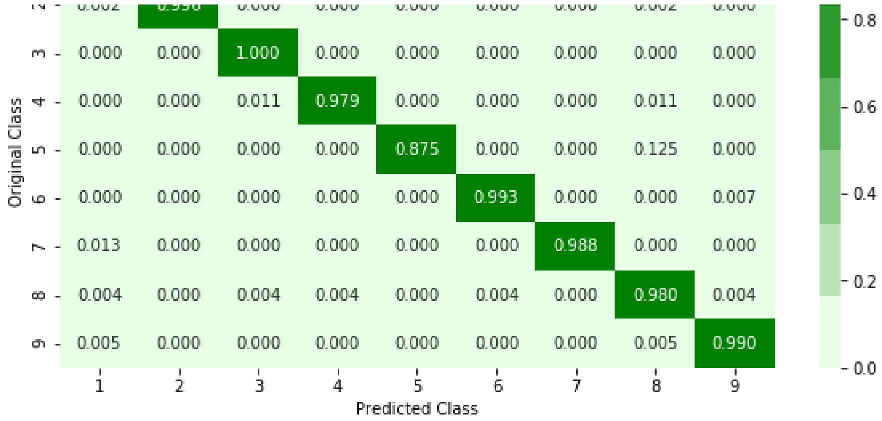

```
log loss for train data 0.01401275289735877
log loss for cv data 0.028101889950011717
log loss for test data 0.03406187189263935
Number of misclassified points  0.8279668813247469
-------------------------------------------------- Confusion matrix -------------------------------
----------------
```


```
-------------------------------------------------- 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.]
```

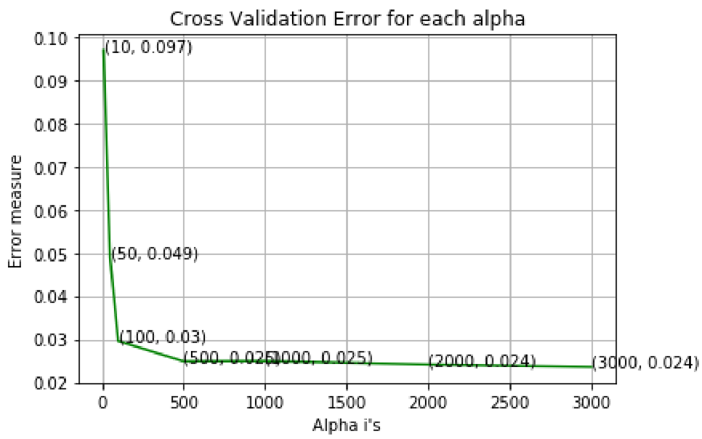
4.4.4 XgBoost Classifier

In [83]:

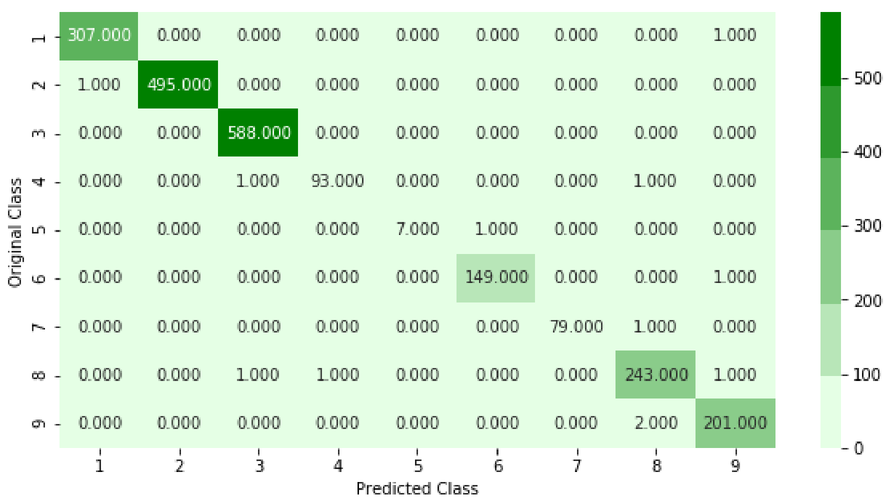
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# 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, verbo
se=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 no
t 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-en
sembles/
# -----------------------

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=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))
```

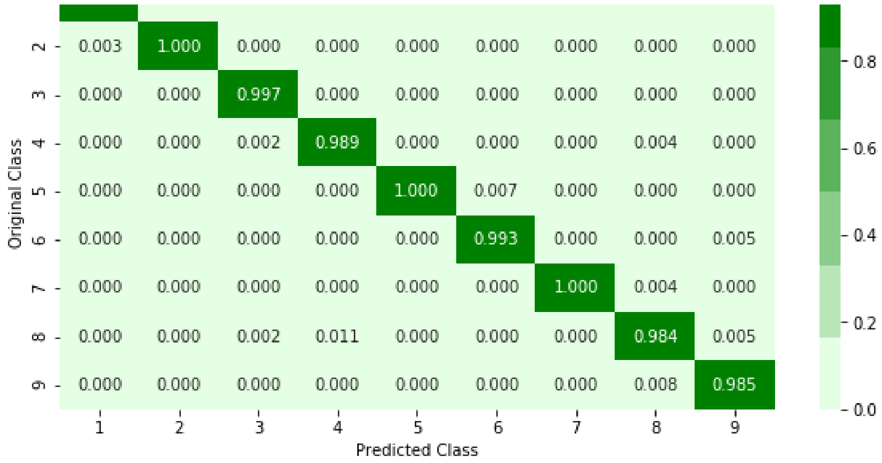
```
log_loss for c =  10 is 0.09687367359857597
log_loss for c =  50 is 0.04889395487009157
log_loss for c =  100 is 0.029586700275233423
log_loss for c =  500 is 0.024916238927858315
log_loss for c =  1000 is 0.024981772319136415
log_loss for c =  2000 is 0.024161614707979114
log_loss for c =  3000 is 0.023592159620269226
```



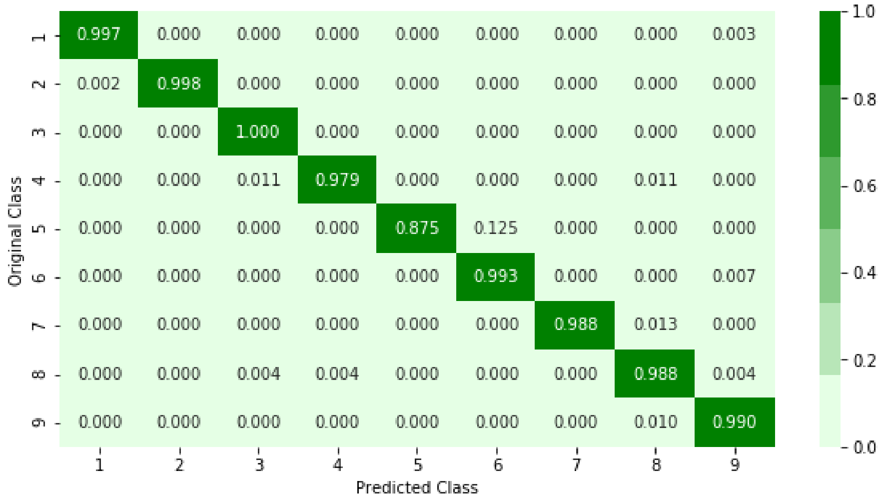

```
For values of best alpha =  3000 The train log loss is: 0.011655401136386943
For values of best alpha =  3000 The cross validation log loss is: 0.023592159620269226
For values of best alpha =  3000 The test log loss is: 0.036899819791654756
Number of misclassified points  0.5519779208831647
-------------------------------------------------- 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 [84]:

```
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)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  8.5min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 10.5min remaining:  1.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 11.0min finished
```

Out[84]:

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

In [85]:

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

```
{'subsample': 0.3, 'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.15, 'colsample_bytree':
1}
```

In [86]:

```
# 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, verbo
se=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 no
t 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-en
sembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5,max_dept
h=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 0.012450566428266408
cv loss 0.0248884582822612
test loss 0.03319751955642889
```

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

4.5.1. Merging both asm and byte file features

In [16]:

```
result = pd.read_csv('byteFeatures.csv')
```

In [17]:

```
result.head()
```

Out[17]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	f9
0	0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	...	0.013560
1	1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	...	0.001920
2	2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	...	0.009804
3	3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	...	0.002121
4	4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	...	0.001530

5 rows × 261 columns

In [18]:

```
result_asm = pd.read_csv('asmFeatures.csv')
```

In [19]:

```
result_asm.head()
```

Out[19]:

	Unnamed: 0	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	...	esi	ea
0	0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	...	0.000746	0.00030
1	1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	...	0.000328	0.00096
2	2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	...	0.000475	0.00020
3	3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	...	0.000090	0.00028
4	4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	...	0.000102	0.00036

5 rows × 55 columns

In [20]:

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

```
(10868, 261)
(10868, 55)
```

In [21]:

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

Out[21]:

	Unnamed: 0_x	0	1	2	3	4	5	6	7	8	...	edx	esi	e
0	0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	...	0.015418	0.025875	0.025
1	1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	...	0.004961	0.012316	0.007
2	2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	...	0.000095	0.006181	0.000
3	3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	...	0.000343	0.000746	0.000
4	4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	...	0.000343	0.013875	0.000

5 rows × 309 columns

In [22]:

```
result_x.to_csv('asm_byte_features.csv')
```

In [23]:

```
result_y.to_csv('class_labels.csv')
```

In [24]:

```
result_x.shape
```

Out[24]:

```
(10868, 309)
```

In [25]:

```
result_y.shape
```

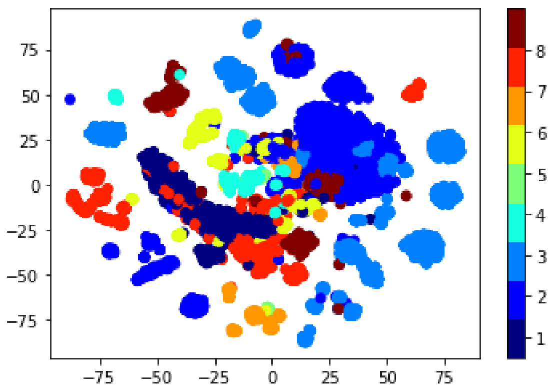
Out[25]:

```
(10868,)
```

4.5.2. Multivariate Analysis on final fearures

In [91]:

```
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 [92]:

```
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)
```

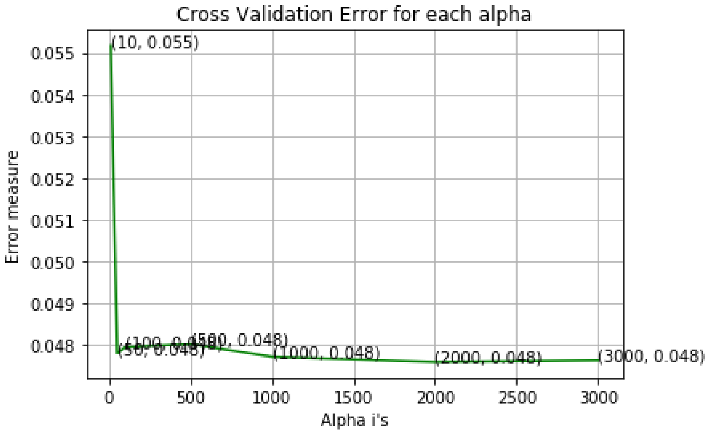
4.5.4. Random Forest Classifier on final features

In [93]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_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-fores
t-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_lo
ss(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 0.05516454747620572
log_loss for c =  50 is 0.04781040080855114
log_loss for c =  100 is 0.047954913272203865
log_loss for c =  500 is 0.04803211461983113
log_loss for c =  1000 is 0.04772607973092137
log_loss for c =  2000 is 0.047598569256819295
log_loss for c =  3000 is 0.04764087875292368
```




```
For values of best alpha =  2000 The train log loss is: 0.016541211459021366
For values of best alpha =  2000 The cross validation log loss is: 0.047598569256819295
For values of best alpha =  2000 The test log loss is: 0.051243853612484846
```

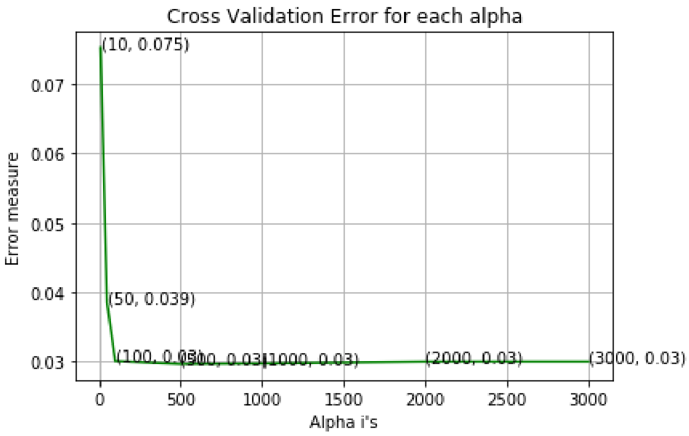
4.5.5. XgBoost Classifier on final features

In [94]:

```
# 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, verbo
se=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 no
t 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-en
sembles/
# -----------------------

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_lo
ss(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 0.07523461567814871
log_loss for c =  50 is 0.03852076516629391
log_loss for c =  100 is 0.030098633603574544
log_loss for c =  500 is 0.029662288687246283
log_loss for c =  1000 is 0.029746979570145826
log_loss for c =  2000 is 0.030015343470325487
log_loss for c =  3000 is 0.030015133379280776
```




```
For values of best alpha =  500 The train log loss is: 0.012713297985148074
For values of best alpha =  500 The cross validation log loss is: 0.030015133379280776
For values of best alpha =  500 The test log loss is: 0.040164014756712385
```

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

In [95]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
```

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

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  9.7min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 28.0min remaining:  3.1min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 33.1min finished
```

Out[95]:

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

In [96]:

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

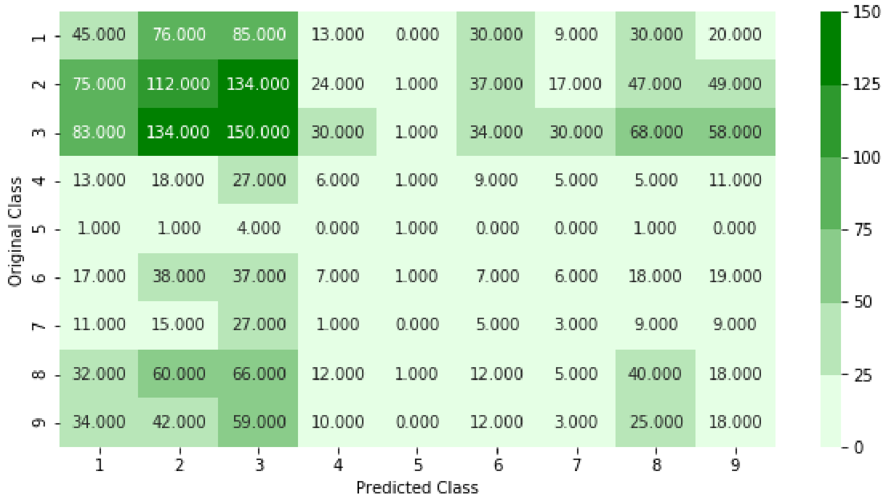
```
{'subsample': 1, 'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.2, 'colsample_bytree': 1}
```

In [97]:

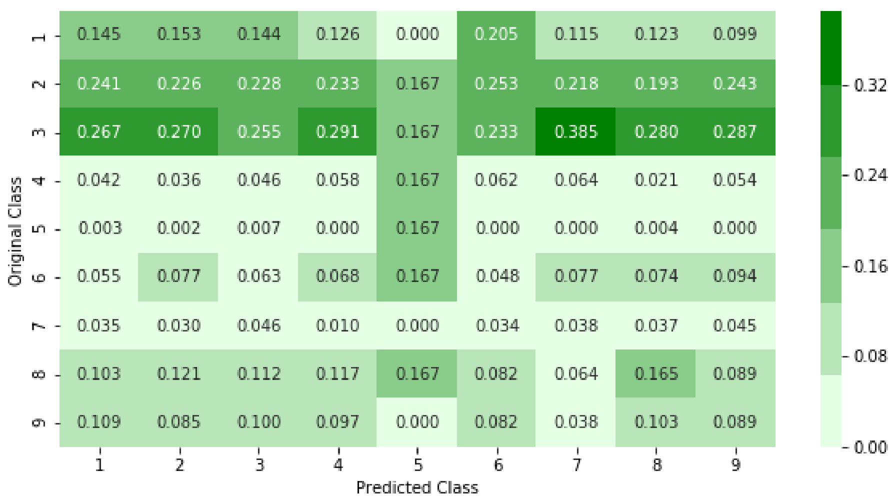
```
# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', 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, verbo
se=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 no
t 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-en
sembles/
# -----------------------
```

```
x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=0.5,subsampl
e=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_lo
ss(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))
```

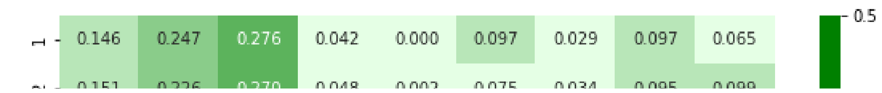
```
For values of best alpha =  500 The train log loss is: 0.01334929780943573
For values of best alpha =  500 The cross validation log loss is: 0.03561046834864658
For values of best alpha =  500 The test log loss is: 0.045008050447680366
Number of misclassified points  82.42870285188593
---------------------------------------------------- Confusion matrix --------------------------------
----------------
```

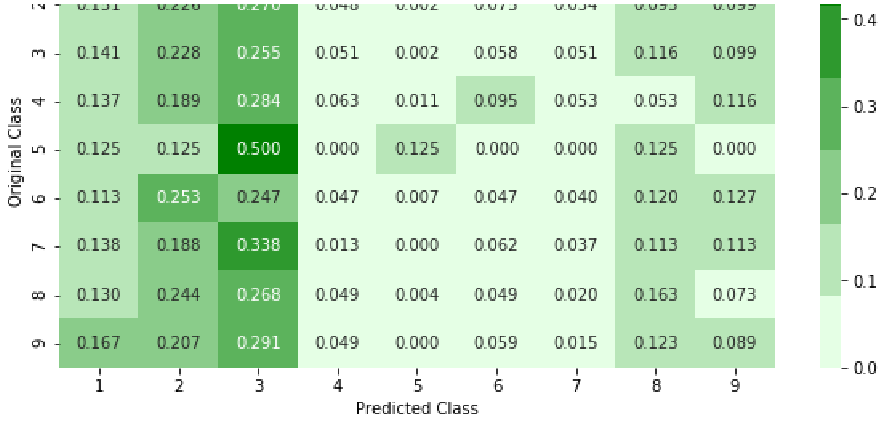



```
---------------------------------------------------- Precision matrix --------------------------------
----------------
```




```
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.]
```

New features on byte files

In [3]:

```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import scipy
import os
from sklearn.preprocessing import normalize
```

In [4]:

```
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,2(
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,
4,45,46,47,48,49,4a,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,(
,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
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,
b,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,c
,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
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

bi-grams

In [5]:

```
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[5]:

```
66049
```

In [16]:

```
byte_bigram_vocab[:5]
```

Out[16]:

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

In [18]:

```
if not os.path.isfile('bytebigrams_norm.npz'):

    byteFilesPath = r"F:\AAIC\Assignments\17 MicrosoftMalwareDetection\MicrosoftMalware\byteFiles"
```

```
    byteFilesPath = r"E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\byteFiles"
    vect_bigram = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
    byte_bigram_data = []
    for i, file in tqdm(enumerate(os.listdir(byteFilesPath))):
        f = open(byteFilesPath+'/' + file)
        byte_bigram_data.append(scipy.sparse.csr_matrix(vect_bigram.fit_transform([f.read().replace
('\n', ' ').lower()])))
        f.close()
    byte_bigrams = scipy.sparse.vstack(byte_bigram_data)
    byte_bigram_vect = normalize(byte_bigrams, axis = 0)
    scipy.sparse.save_npz('bytebigrams_norm.npz', byte_bigrams_vect)
else:
    byte_bigrams_vect = scipy.sparse.load_npz('bytebigrams_norm.npz')
```

tri-grams

In [ ]:

```
byte_trigram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for bigram in byte_bigram_vocab:
        byte_trigram_vocab.append(v+' '+bigram)

len(byte_trigram_vocab)
```

In [ ]:

```
if not os.path.isfile('byte_trigrams_norm.npz'):

    byteFilesPath = r"E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\byteFiles"
    vect_trigram = CountVectorizer(lowercase=False,ngram_range=(3,3), vocabulary=byte_trigram_vocab
,max_df=100)
    byte_trigram_data = []
    for i, file in tqdm(enumerate(os.listdir(byteFilesPath))):
        f = open(byteFilesPath+'/' + file)
        byte_trigram_data.append(scipy.sparse.csr_matrix(vect_trigram.fit_transform([f.read().repla
ce('\n', ' ').lower()])))
        f.close()
    byte_trigrams = scipy.sparse.vstack(byte_trigram_data)
    byte_trigram_vect = normalize(byte_trigrams, axis = 0)
    scipy.sparse.save_npz('byte_trigrams_norm.npz', byte_trigram_vect)
else:
    byte_trigram_vect = scipy.sparse.load_npz('byte_trigrams_norm.npz')
```

tetra-grams

In [ ]:

```
byte_tetragram_vocab = []
for i, v in enumerate(byte_vocab.split(',')):
    for trigram in byte_trigram_vocab:
        byte_tetragram_vocab.append(v+' '+trigram)

len(byte_tetragram_vocab)
```

In [ ]:

```
if not os.path.isfile('byte_tetragrams_norm.npz'):

    byteFilesPath = r"E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\byteFiles"
    vect_tetragram = CountVectorizer(lowercase=False,ngram_range=(4,4),
vocabulary=byte_tetragram_vocab,max_df=100)
    byte_tetragram_data = []
    for i, file in tqdm(enumerate(os.listdir(byteFilesPath))):
        f = open(byteFilesPath+'/' + file)
        byte_tetragram_data.append(scipy.sparse.csr_matrix(vect_tetragram.fit_transform([f.read().r
eplace('\n', ' ').lower()])))
        f.close()
    byte_tetragrams = scipy.sparse.vstack(byte_tetragram_data)
    byte_tetragram_vect = normalize(byte_tetragrams, axis = 0)
    scipy.sparse.save_npz('byte_tetragrams_norm.npz', byte_tetragram_vect)
```

```
    scipy.sparse.save_npz('byte_tetragrams_norm.npz', byte_tetragram_vect)
else:
    byte_tetragram_vect = scipy.sparse.load_npz('byte_tetragrams_norm.npz')
```

New features on asm files

In [6]:

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

In [ ]:

```
import codecs

if not os.path.isfile("opcode_file.txt"):

    op_file = open("opcode_file.txt", "w+")
    asmFilesDir = r"E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\asmFiles"
    for asmfile in os.listdir(asmFilesDir):
        opcode_str = ""
        with codecs.open(asmFilesDir+'/' + asmfile, encoding='cp1252', errors ='replace') as file:
            for line in file:
                sents = line.rstrip().split()
                for li in sents:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
```

In [ ]:

```
raw_opcode = open('opcode_file.txt').read().split('\n')
```

bi-grams

In [7]:

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

In [ ]:

```
len(asm_bigrams)
```

In [ ]:

```
asm_bigrams[:5]
```

In [ ]:

```
if not os.path.isfile('asmbigrams.npz'):

    vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_bigrams)
    asm_bigrams_data = []

    for i in range(10868):
        asm_bigrams_data.append(scipy.sparse.csr_matrix(vect.transform([raw_opcode[i]])))

    asm_bigrams_vect = scipy.sparse.vstack(asm_bigrams_data)
    scipy.sparse.save_npz('asmbigrams.npz', asm_bigrams_vect)
else:
    asm_bigrams_vect = scipy.sparse.load_npz('asmbigrams.npz')
```

In [ ]:

In [ ]:

```
from sklearn.preprocessing import normalize
asm_bigrams_vect = normalize(asm_bigrams_vect, axis = 0)
```

In [ ]:

```
scipy.sparse.save_npz('asmbigrams_norm.npz',asm_bigrams_vect)
```

tri-grams

In [8]:

```
asm_trigrams = []
for bigram in asm_bigrams:
    for j in range(0,len(opcodes)):
        asm_trigrams.append(bigram+' '+opcodes[j])
```

In [ ]:

```
len(asm_trigrams)
```

In [ ]:

```
asm_trigrams[:5]
```

In [ ]:

```
if not os.path.isfile('asmtrigrams.npz'):

    vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_trigrams,max_df=100)
    asm_trigrams_data = []

    for i in range(10868):
        asm_trigrams_data.append(scipy.sparse.csr_matrix(vect.transform([raw_opcode[i]])))

    asm_trigrams_vect = scipy.sparse.vstack(asm_trigrams_data)
    scipy.sparse.save_npz('asmtrigrams.npz', asm_trigrams_vect)
else:
    asm_trigrams_vect = scipy.sparse.load_npz('asmtrigrams.npz')
```

In [ ]:

```
from sklearn.preprocessing import normalize
asm_trigrams_vect = normalize(asm_trigrams_vect, axis = 0)
```

In [ ]:

```
scipy.sparse.save_npz('asmtrigrams_norm.npz',asm_trigrams_vect)
```

tetra-grams

In [9]:

```
asm_tetragrams = []
for trigram in asm_trigrams:
    for j in range(0,len(opcodes)):
        asm_tetragrams.append(trigram+' '+opcodes[j])
```

In [70]:

```
len(asm_tetragrams)
```

Out[70]:

```
456976
```

In [71]:

```
asm_tetragrams[:5]
```

Out[71]:

```
['jmp jmp jmp jmp',
 'jmp jmp jmp mov',
 'jmp jmp jmp retf',
 'jmp jmp jmp push',
 'jmp jmp jmp pop']
```

In [ ]:

```
if not os.path.isfile('asmtetragrams.npz'):

    vect = CountVectorizer(ngram_range=(4, 4), vocabulary = asm_tetragrams,max_df=100)
    asm_tetragrams_data = []

    for i in range(10868):
        asm_tetragrams_data.append(scipy.sparse.csr_matrix(vect.transform([raw_opcode[i]])))

    asm_tetragrams_vect = scipy.sparse.vstack(asm_tetragrams_data)
    scipy.sparse.save_npz('asmtetragrams.npz', asm_tetragrams_vect)
else:
    asm_tetragrams_vect = scipy.sparse.load_npz('asmtetragrams.npz')
```

In [ ]:

```
from sklearn.preprocessing import normalize
asm_tetragrams_vect = normalize(asm_tetragrams_vect, axis = 0)
```

In [ ]:

```
scipy.sparse.save_npz('asmtetragrams_norm.npz',asm_tetragrams_vect)
```

Image Feature Extraction From ASM files

In [10]:

```
import os
import numpy as np
import math
import array
import time as tm
import numpy as np
import scipy as sp
import pandas as pd
```

In [11]:

```
def read_image(filename):
    '''
    Read image data
    '''
    print(filename)
    f = open(filename,'rb')
    ln = os.path.getsize(filename) # length of file in bytes
    width = 256
    rem = ln%width
    a = array.array("B") # uint8 array
    a.fromfile(f,ln-rem)
    f.close()
    g = np.reshape(a,(int(len(a)/width), width))
    g = np.uint8(g)
    g = np.resize(g, (1000,))
    return list(g)
```

In [9]:

```
def extract_asm_image_features(tfiles):
    '''
    Extract image features from the asm files
    '''
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)
    print(ftot)
    # Generate feature file csv
    pid = os.getpid()
    print(pid)
    feature_file = str(pid) + '-image-features-asm.csv'

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in range(1000)]
        fw.writerow(column_names)
        for idx, fname in tqdm(enumerate(asm_files)):
            file_id = fname.split('.')[0]
            image_data =
read_image(r'E:/AAIC/Assignments/17.MicrosoftMalwareDetection/MicrosoftMalware/asmFiles/'+ fname)
            outrows.append([file_id] + image_data)

            # Print progress
            if (idx+1) % 100 == 0:
                print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

        # Write remaining files
        if len(outrows) > 0:
            fw.writerows(outrows)
            outrows = []
```

In [ ]:

```
start_time = tm.time()
tfiles = os.listdir(r'E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\asmFiles')
extract_asm_image_features(tfiles)
print("Elapsed time: {:.2f} hours.".format((tm.time() - start_time)/3600.0))
```

In [12]:

```
imageFeatures = pd.read_csv('image-features-asm.csv')
imageFeatures.shape
```

Out[12]:

```
(10868, 1001)
```

In [13]:

```
imageFeatures.head()
```

Out[13]:

	filename	ASM_0	ASM_1	ASM_2	ASM_3	ASM_4	ASM_5	ASM_6	ASM_7	ASM_8	...	ASM_990	ASM_991	ASM
0	01azqd4InC7m9JpocGv5	72	69	65	68	69	82	58	48	48	...	116	101	
1	01IsoiSMh5gxyDYTl4CB	46	116	101	120	116	58	48	48	52	...	10	46	
2	01jsnpXSAlgw6aPeDxrU	72	69	65	68	69	82	58	48	48	...	116	101	
3	01kcPWA9K2BOxQeS5Rju	72	69	65	68	69	82	58	49	48	...	71	77	
4	01SuzwMJEIXsK7A8dQbl	72	69	65	68	69	82	58	48	48	...	116	101	

5 rows × 1001 columns

In [37]:

```
result_x = result_x.drop(['Unnamed: 0_x'],axis=1)
```

In [14]:

```
imageFeatures = imageFeatures.drop(['filename'],axis=1)
```

In [114]:

```
final_data = pd.concat([result_x,imageFeatures], axis = 1, join = 'inner')
```

Important Feature Selection Using Random Forest

In [15]:

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

In [16]:

```
asmFileNames = []
asmFilesDir = r"E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\asmFiles"
for asmFile in os.listdir(asmFilesDir):
    head,tail = os.path.split(asmFile)
    asmFileNames.append(tail.split('.')[0])
```

In [17]:

```
byteFileNames = []
byteFilesDir = r"E:\AAIC\Assignments\17.MicrosoftMalwareDetection\MicrosoftMalware\byteFiles"
for byteFile in os.listdir(byteFilesDir):
    head,tail = os.path.split(byteFile)
    byteFileNames.append(tail.split('.')[0])
```

In [18]:

```
numberOfFeatures = 15000
topFeatures = 200
```

In [21]:

```
result_y = pd.read_csv('class_labels.csv')
```

In [26]:

```
result_y = result_y.drop(['Unnamed: 0'],axis=1)
```

Top 200 Byte Bi-Gram Features

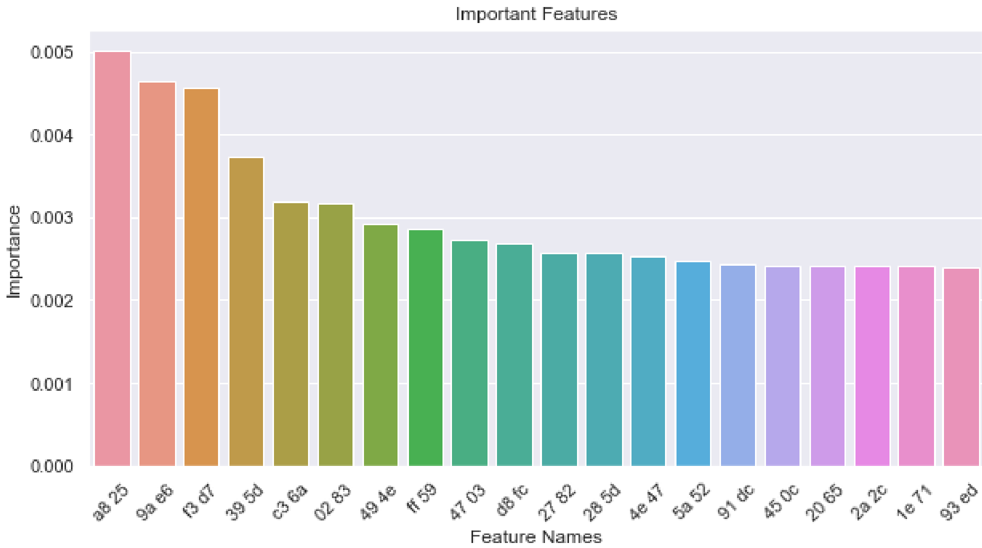
In [19]:

```
byte_bigram_vect = scipy.sparse.load_npz('bytebigrams_norm.npz')
```

```
byte_bigram_vect = scipy.sparse.load_npz('bytebigrams_norm.npz')
```

In [27]:

```
top_200_byte_bigram_features_index = imp_features(byte_bigram_vect,byte_bigram_vocab,topFeatures)
```



In [28]:

```
top_200_feat_byte_bigram_data = np.zeros((10868, 0))
for i in top_200_byte_bigram_features_index:
    sliced = byte_bigram_vect[:, i].todense()
    top_200_feat_byte_bigram_data = np.hstack([top_200_feat_byte_bigram_data, sliced])
```

In [29]:

```
byte_bigram_df = pd.DataFrame(top_200_feat_byte_bigram_data,columns = np.take(byte_bigram_vocab,
top_200_byte_bigram_features_index))
```

In [30]:

```
byte_bigram_df.head()
```

Out[30]:

	a8 25	9a e6	f3 d7	39 5d	c3 6a	02 83	49 4e	ff 59	47 03	d8 fc	...	56 56	f6 45	8b 4
0	0.003512	0.003650	0.004069	0.000148	0.000567	0.000270	0.000959	0.000386	0.000199	0.002237	...	0.000637	0.001995	0.00069
1	0.000000	0.000000	0.001017	0.000935	0.002552	0.000517	0.000213	0.008937	0.000288	0.001627	...	0.000163	0.003989	0.00230
2	0.003512	0.009126	0.005086	0.000197	0.000992	0.000472	0.001811	0.012477	0.000044	0.001424	...	0.000523	0.002992	0.00230
3	0.001317	0.000913	0.001017	0.000098	0.000567	0.000112	0.001704	0.001893	0.000000	0.000610	...	0.000033	0.000997	0.00046
4	0.002195	0.000913	0.002034	0.000000	0.000000	0.000045	0.000000	0.001507	0.000000	0.000000	...	0.000000	0.000997	0.00017

5 rows × 200 columns

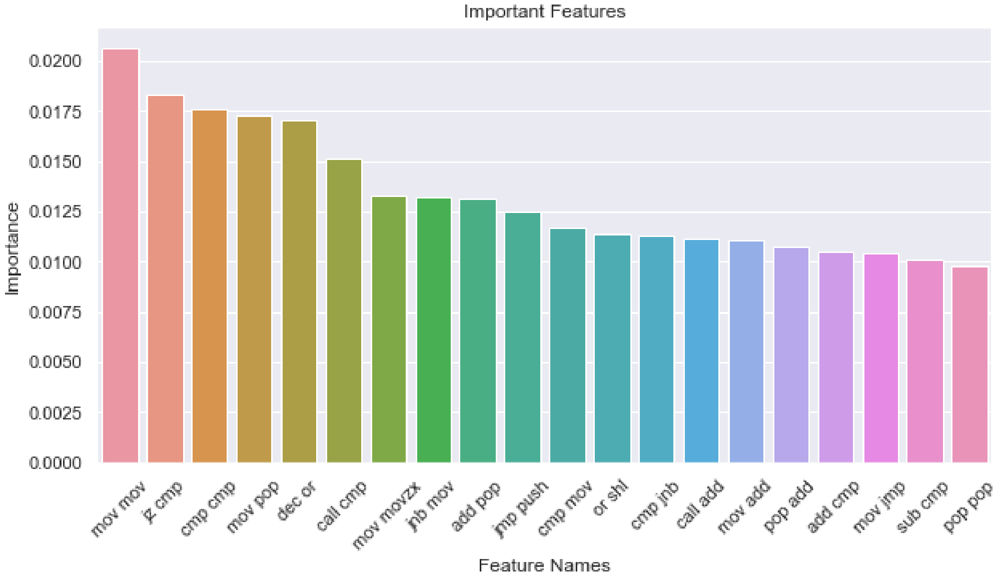
Top 200 ASM Bi-Gram Features

In [31]:

```
asm_bigrams_vect = scipy.sparse.load_npz('asmbigrams_norm.npz')
```

In [32]:

```
top_200_asm_bigram_features_index = imp_features(asm_bigrams_vect,asm_bigrams,topFeatures)
```



In [33]:

```
top_200_feat_asm_bigram_data = np.zeros((10868, 0))
for i in top_200_asm_bigram_features_index:
    sliced = asm_bigrams_vect[:, i].todense()
    top_200_feat_asm_bigram_data = np.hstack([top_200_feat_asm_bigram_data, sliced])
```

In [34]:

```
asm_bigram_df = pd.DataFrame(top_200_feat_asm_bigram_data,columns = np.take(asm_bigrams,
top_200_asm_bigram_features_index))
```

In [35]:

```
asm_bigram_df.head()
```

Out[35]:

	mov mov	jz cmp	cmp cmp	mov pop	dec or	call cmp	mov movzx	jnb mov	add pop	jmp push	...	sub pop	call movzx	shr or
0	0.002736	0.000417	0.000922	0.004581	0.000000	0.008045	0.000000	0.000000	0.001096	0.00042	...	0.021503	0.000000	0.000000
1	0.001812	0.000209	0.000503	0.001497	0.001152	0.000380	0.002776	0.003595	0.000274	0.00021	...	0.000000	0.002508	0.000272
2	0.000000	0.000522	0.002432	0.000000	0.000000	0.000326	0.000084	0.000000	0.000000	0.00000	...	0.000000	0.010660	0.000000
3	0.000039	0.000104	0.000084	0.000091	0.000576	0.000109	0.000000	0.000599	0.000000	0.00007	...	0.000000	0.000000	0.000000
4	0.001898	0.000104	0.000168	0.000499	0.000576	0.000054	0.001093	0.004494	0.000183	0.00028	...	0.000000	0.000000	0.000000

5 rows × 200 columns

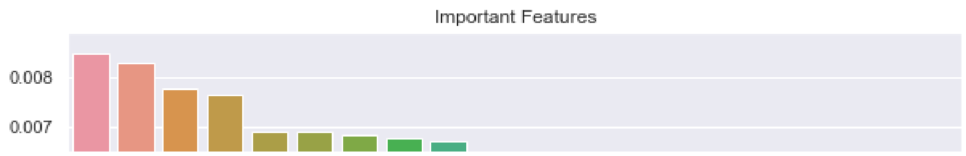
Top 200 ASM Tri-Gram Features

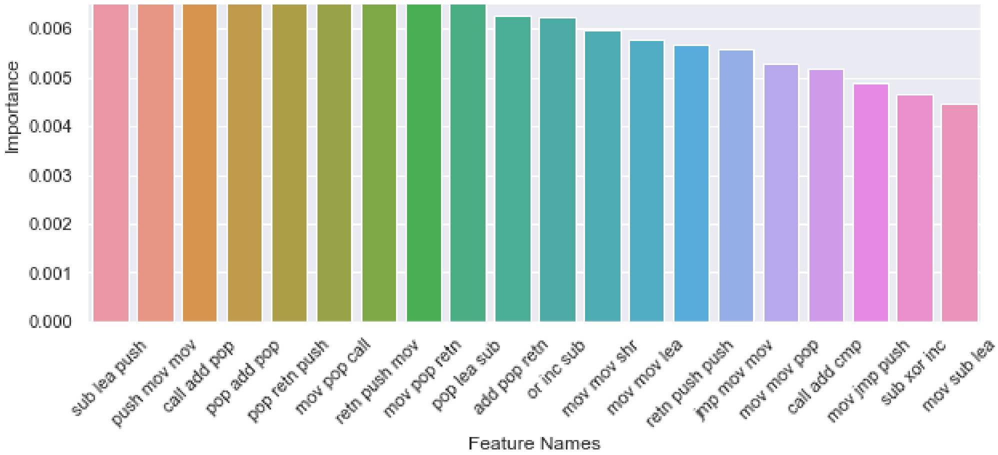
In [36]:

```
asm_trigrams_vect = scipy.sparse.load_npz('asmtrigrams_norm.npz')
```

In [37]:

```
top_200_asm_trigram_features_index = imp_features(asm_trigrams_vect,asm_trigrams,topFeatures)
```





In [38]:

```
top_200_feat_asm_trigram_data = np.zeros((10868, 0))
for i in top_200_asm_trigram_features_index:
    sliced = asm_trigrams_vect[:, i].todense()
    top_200_feat_asm_trigram_data = np.hstack([top_200_feat_asm_trigram_data, sliced])
```

In [39]:

```
asm_trigram_df = pd.DataFrame(top_200_feat_asm_trigram_data,columns = np.take(asm_trigrams,
top_200_asm_trigram_features_index))
```

In [40]:

```
asm_trigram_df.head()
```

Out[40]:

	sub lea push	push mov mov	call add pop	pop add pop	pop retn push	mov pop call	retn push mov	mov pop retn	pop lea sub	add pop retn	...	jz mov sub	mov retn push	shl mov mov
0	0.001130	0.004141	0.000000	0.003551	0.001802	0.0	0.001129	0.004439	0.000000	0.00084	...	0.013931	0.005650	0.010062
1	0.000000	0.001042	0.000251	0.000000	0.001073	0.0	0.000497	0.001141	0.000000	0.00028	...	0.000000	0.000892	0.003833
2	0.007907	0.000000	0.000000	0.000000	0.000129	0.0	0.001129	0.000000	0.004873	0.00000	...	0.000000	0.000000	0.000000
3	0.000000	0.000055	0.000000	0.000000	0.000000	0.0	0.000090	0.000190	0.000000	0.00000	...	0.000000	0.000000	0.000000
4	0.001130	0.000137	0.000000	0.000000	0.000815	0.0	0.000858	0.001268	0.000000	0.00000	...	0.010061	0.000000	0.013895

5 rows × 200 columns

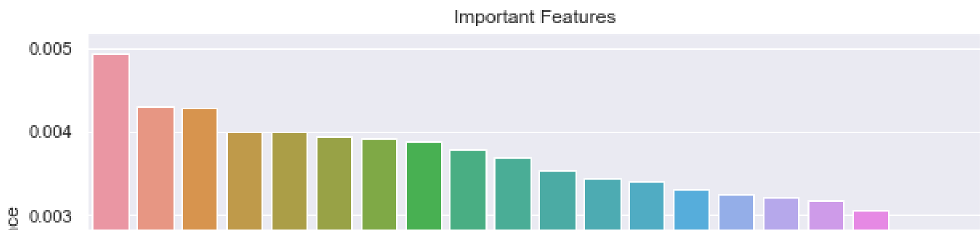
Top 200 ASM Tetra-Gram Features

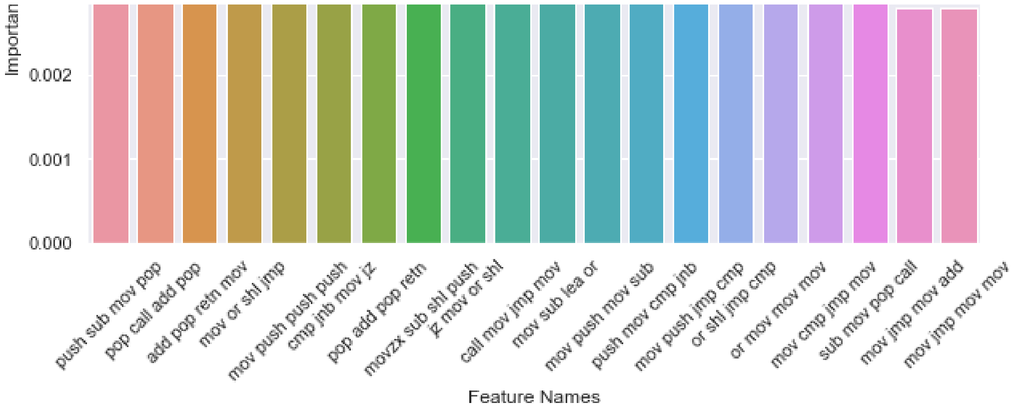
In [41]:

```
asm_tetragrams_vect = scipy.sparse.load_npz('asmtetragrams_norm.npz')
```

In [42]:

```
top_200_asm_tetragram_features_index = imp_features(asm_tetragrams_vect,asm_tetragrams,topFeatures
)
```





In [43]:

```
top_200_feat_asm_tetragram_data = np.zeros((10868, 0))
for i in top_200_asm_tetragram_features_index:
    sliced = asm_tetragrams_vect[:, i].todense()
    top_200_feat_asm_tetragram_data = np.hstack([top_200_feat_asm_tetragram_data, sliced])
```

In [44]:

```
asm_tetragram_df = pd.DataFrame(top_200_feat_asm_tetragram_data,columns = np.take(asm_tetragrams, t
op_200_asm_tetragram_features_index))
```

In [45]:

```
asm_tetragram_df.head()
```

Out[45]:

	push sub mov pop	pop call add pop	add pop retn mov	mov or shl jmp	mov push push push	cmp jnb mov jz	pop add pop retn	movzx sub shl push	jz mov or shl	call mov jmp mov	...	sub lea cmp jz	mov call mov add	mov jmp mov cmp	jz push call mov	mov jmp mov push
0	0.0	0.0	0.000854	0.0	0.012679	0.000000	0.000725	0.0	0.0	0.005826	...	0.0	0.000659	0.008246	0.007820	0.002245
1	0.0	0.0	0.000000	0.0	0.000171	0.011106	0.000000	0.0	0.0	0.000380	...	0.0	0.001977	0.004398	0.000823	0.001796
2	0.0	0.0	0.000000	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.000000	...	0.0	0.000000	0.000000	0.000000	0.000000
3	0.0	0.0	0.000000	0.0	0.000043	0.000000	0.000000	0.0	0.0	0.000000	...	0.0	0.000000	0.000000	0.000000	0.000000
4	0.0	0.0	0.000000	0.0	0.000043	0.000000	0.000000	0.0	0.0	0.000127	...	0.0	0.000000	0.000550	0.000000	0.002245

5 rows × 200 columns

Advanced Features

byte bigrams+asm bigrams+asm trigrams+asm tetragrams+asm image features

In [46]:

```
asm_byte_unigrams = pd.read_csv('asm_byte_features.csv')
```

In [47]:

```
asm_byte_unigrams.shape
```

Out[47]:

```
(10868, 310)
```

In [48]:

```
final_data = pd.concat([asm_byte_unigrams, byte_bigram_df, asm_bigram_df, asm_trigram_df, asm_tetra
gram_df,imageFeatures], axis = 1, join = 'inner')
```

In [50]:

```
final_data = final_data.drop(['Unnamed: 0','Unnamed: 0_x'],axis=1)
```

In [52]:

```
final_data.head()
```

Out[52]:

	0	1	2	3	4	5	6	7	8	9	...	ASM_990	ASM_991	ASM_
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531	...	116	101	
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394	...	10	46	
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707	...	116	101	
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959	0.000521	...	71	77	
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376	0.000246	...	116	101	

5 rows × 2108 columns

In [53]:

```
final_data.to_csv('final_data.csv')
```

In [54]:

```
byte_bigram_df.to_csv('byte_bigram_df.csv')
asm_bigram_df.to_csv('asm_bigram_df.csv')
asm_trigram_df.to_csv('asm_trigram_df.csv')
asm_tetragram_df.to_csv('asm_tetragram_df.csv')
```

In [55]:

```
final_data.shape
```

Out[55]:

```
(10868, 2108)
```

In [56]:

```
x_train_final, x_test_final, y_train_final, y_test_final = train_test_split(final_data, result_y, s
tratify = result_y, test_size = 0.20)
x_trn_final, x_cv_final, y_trn_final, y_cv_final = train_test_split(x_train_final, y_train_final, s
tratify = y_train_final, test_size = 0.25)
```

In [57]:

```
x_trn_final.shape
```

Out[57]:

```
(6520, 2108)
```

In [58]:

```
x_cv_final.shape
```

Out[58]:

```
(2174, 2108)
```

In [59]:

```
x_test_final.shape
```

Out[59]:

```
(2174, 2108)
```

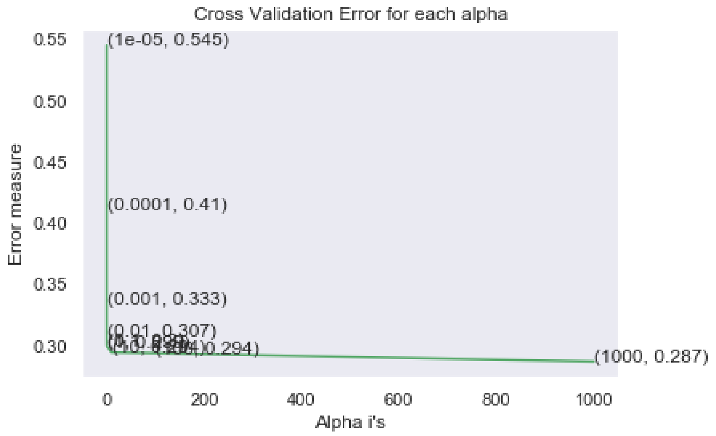
Machine Learning Models on Byte features + ASM features + Advanced features

Logistic Regression

In [60]:

```
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_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(x_trn_final,y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, 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()
```

```
log_loss for c =  1e-05 is 0.5448628494597765
log_loss for c =  0.0001 is 0.4099540415727637
log_loss for c =  0.001 is 0.33307261225467955
log_loss for c =  0.01 is 0.30718494451783646
log_loss for c =  0.1 is 0.29987024068314233
log_loss for c =  1 is 0.2989709985845353
log_loss for c =  10 is 0.2943535683507242
log_loss for c =  100 is 0.2937031027991738
log_loss for c =  1000 is 0.28687762311871784
```



In [61]:

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

predict_y = sig_clf.predict_proba(x_trn_final)
print ('log loss for train data',(log_loss(y_trn_final, predict_y, labels=logisticR.classes_, eps=1
e-15)))
predict_y = sig_clf.predict_proba(x_cv_final)
print ('log loss for cv data',(log_loss(y_cv_final, predict_y, labels=logisticR.classes_, eps=1e-15
)))
predict_y = sig_clf.predict_proba(x_test_final)
print ('log loss for test data',(log_loss(y_test_final, predict_y, labels=logisticR.classes_, eps=1
e-15)))
```

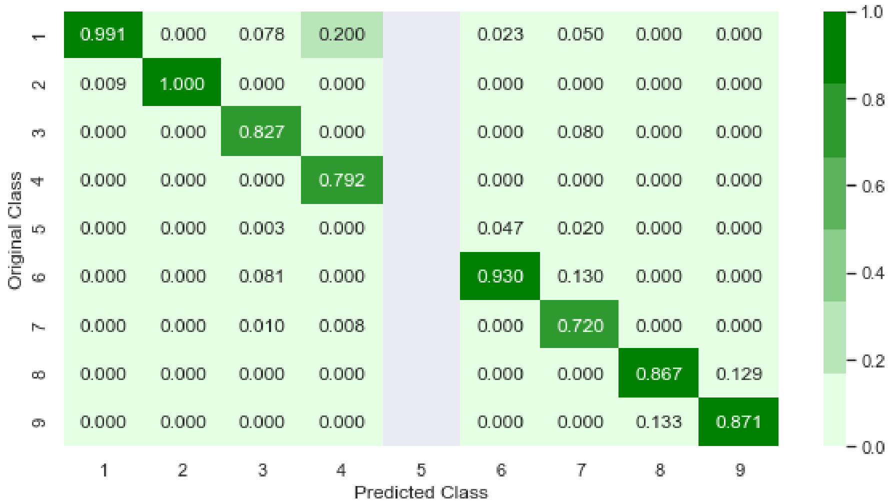
```
log loss for train data 0.28108029135292695
log loss for cv data 0.28687762311871784
log loss for test data 0.2876240529457212
```

In [62]:

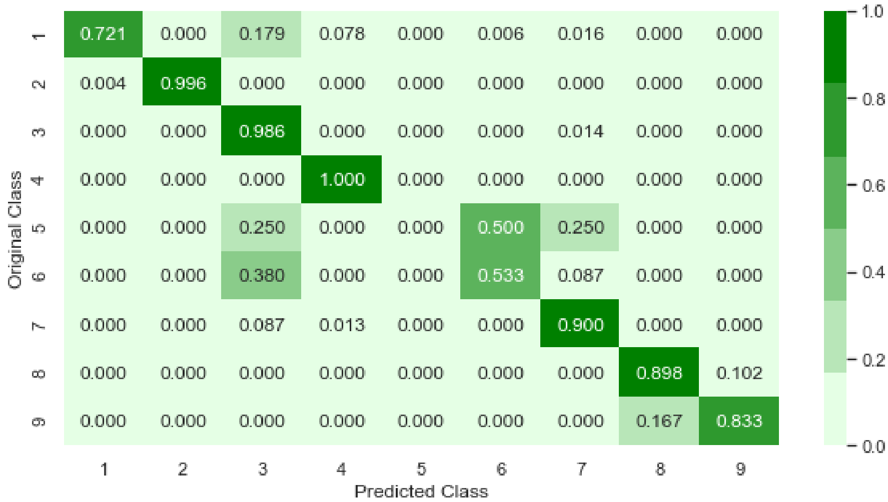
```
plot_confusion_matrix(y_test_final,sig_clf.predict(x_test_final))
```

```
Number of misclassified points  11.085556577736892
-------------------------------------------------- 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.]
```

Xg-Boost Classifier

In [63]:

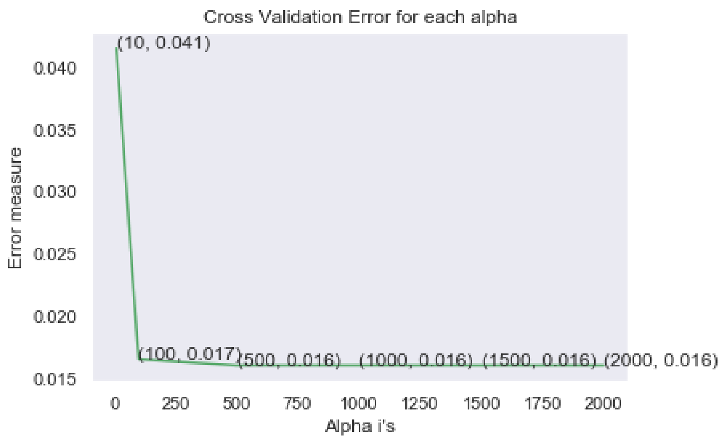
```python
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# 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, verbo
se=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 no
t 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-en
sembles/
# -----------------------

alpha=[10,100,500,1000,1500,2000]
cv_log_error_array=[]
for i in alpha:
    print('i = ',i)
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(x_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(x_trn_final, y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=x_cfl.classes_, eps=1e-15))
```

```
i =  10
i =  100
i =  500
i =  1000
i =  1500
i =  2000
```

In [64]:

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


best_alpha = np.argmin(cv_log_error_array)

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

```
log_loss for c =  10 is 0.04147468912808547
log_loss for c =  100 is 0.016585478912043684
log_loss for c =  500 is 0.016090268599055723
log_loss for c =  1000 is 0.01609494082413553
log_loss for c =  1500 is 0.016092467310550078
log_loss for c =  2000 is 0.016093250060612827
```



In [65]:

```
x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(x_trn_final,y_trn_final,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_trn_final, y_trn_final)

predict_y = sig_clf.predict_proba(x_trn_final)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_trn_final, predict_y))
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_final, predict_y))
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_final, predict_y))
```

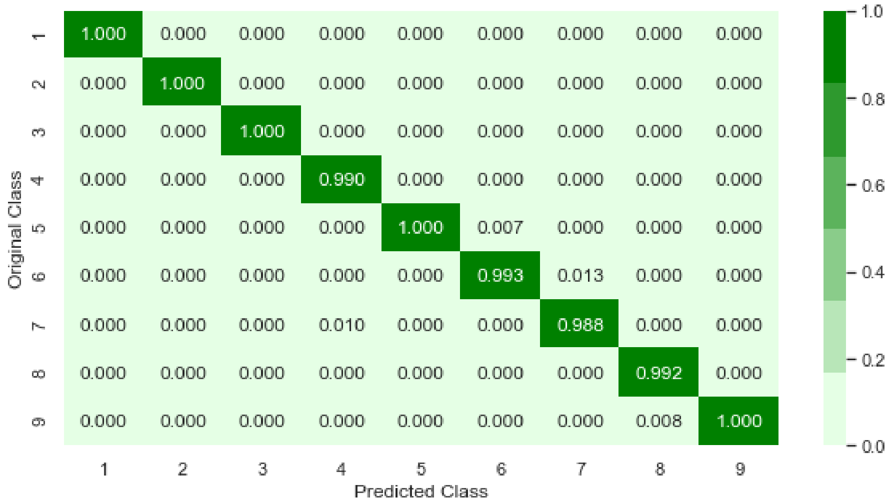
```
For values of best alpha =  500 The train log loss is: 0.008725093969523312
For values of best alpha =  500 The cross validation log loss is: 0.016090268599055723
For values of best alpha =  500 The test log loss is: 0.01376878523175999
```

In [68]:

```
plot_confusion_matrix(y_test_final,sig_clf.predict(x_test_final))
```

```
Number of misclassified points  5
-------------------------------------------------- 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.]
```

Results

In [69]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','log loss']
ptable.add_row(["random","Byte files","2.47"])
ptable.add_row(["knn","Byte files","0.22"])
ptable.add_row(["Logistic Regression","Byte files","0.54"])
ptable.add_row(["Random Forest Classifier ","Byte files","0.81"])
ptable.add_row(["XgBoost Classification","Byte files","0.06"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["knn","asmfiles","0.07"])
ptable.add_row(["Logistic Regression","asmfiles","0.37"])
ptable.add_row(["Random Forest Classifier ","asmfiles","0.34"])
ptable.add_row(["XgBoost Classification","asmfiles","0.04"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Random Forest Classifier ","Byte files+asmfiles","0.05"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles","0.04"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Logistic Regression","Byte files+asmfiles+advanced features","0.28"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles+advanced features","0.013"])
print(ptable)
```

```
+--------------------------------------------------------------------------------+
|                               Model Comparision                                |
+---------------------------+-----------------------------------------+----------+
|           Model           |                 Features                | log loss |
+---------------------------+-----------------------------------------+----------+
|           random          |                Byte files               |   2.47   |
|            knn            |                Byte files               |   0.22   |
|    Logistic Regression    |                Byte files               |   0.54   |
| Random Forest Classifier  |                Byte files               |   0.81   |
|   XgBoost Classification  |                Byte files               |   0.06   |
|                           |                                         |          |
|                           |                                         |          |
|            knn            |                 asmfiles                |   0.07   |
|    Logistic Regression    |                 asmfiles                |   0.37   |
| Random Forest Classifier  |                 asmfiles                |   0.34   |
|   XgBoost Classification  |                 asmfiles                |   0.04   |
|                           |                                         |          |
|                           |                                         |          |
| Random Forest Classifier  |           Byte files+asmfiles           |   0.05   |
|   XgBoost Classification  |           Byte files+asmfiles           |   0.04   |
|                           |                                         |          |
|                           |                                         |          |
|    Logistic Regression    | Byte files+asmfiles+advanced features |   0.28   |
|   XgBoost Classification  | Byte files+asmfiles+advanced features |  0.013   |
+---------------------------+-----------------------------------------+----------+
```

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