Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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

1.2. Problem Statement

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

1.3 Source/Useful Links

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

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

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

1.4. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

Source : https://www.kaggle.com/c/malware-classification/data
For every malware, we have two files

1. .asm file (read more: https://www.reviversoft.com/file-extensions/asm)
2. .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)

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

1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

2.1.2. Example Data Point

.asm file

```
.text:00401000                                         assume es:nothing,
ss:nothing, ds:_data, fs:nothing, gs:nothing
.text:00401000 56                                      push    esi
.text:00401001 8D 44 24 08                                 lea     eax,
[esp+8]
.text:00401005 50                                      push    eax
.text:00401006 8B F1                                       mov     esi,
ecx
.text:00401008 E8 1C 1B 00 00                              call
??0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const *
const &)
.text:0040100D C7 06 08 BB 42 00                           mov
dword ptr [esi], offset off_42BB08
.text:00401013 8B C6                                       mov     eax,
esi
.text:00401015 5E                                      pop     esi
.text:00401016 C2 04 00                                    retn    4
```

```
.text:00401016                                ;
-------------------------------------------------------------------------
.text:00401019 CC CC CC CC CC CC CC                        align 10h
.text:00401020 C7 01 08 BB 42 00                           mov
dword ptr [ecx], offset off_42BB08
.text:00401026 E9 26 1C 00 00                              jmp
sub_402C51
.text:00401026                                ;
-------------------------------------------------------------------------
.text:0040102B CC CC CC CC CC                              align 10h
.text:00401030 56                                  push    esi
.text:00401031 8B F1                                   mov     esi,
ecx
.text:00401033 C7 06 08 BB 42 00                           mov
dword ptr [esi], offset off_42BB08
.text:00401039 E8 13 1C 00 00                              call
sub_402C51
.text:0040103E F6 44 24 08 01                              test
byte ptr [esp+8], 1
.text:00401043 74 09                                   jz      short
loc_40104E
.text:00401045 56                                  push    esi
.text:00401046 E8 6C 1E 00 00                              call
??3@YAXPAX@Z    ; operator delete(void *)
.text:0040104B 83 C4 04                                add     esp, 4
.text:0040104E
.text:0040104E                             loc_40104E:
; CODE XREF: .text:00401043j
.text:0040104E 8B C6                                   mov     eax,
esi
.text:00401050 5E                                  pop     esi
.text:00401051 C2 04 00                                retn    4
.text:00401051                                ;
-------------------------------------------------------------------------
```

.bytes file

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

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train and Test Dataset

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

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf

First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [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
from sklearn.preprocessing import MinMaxScaler
```

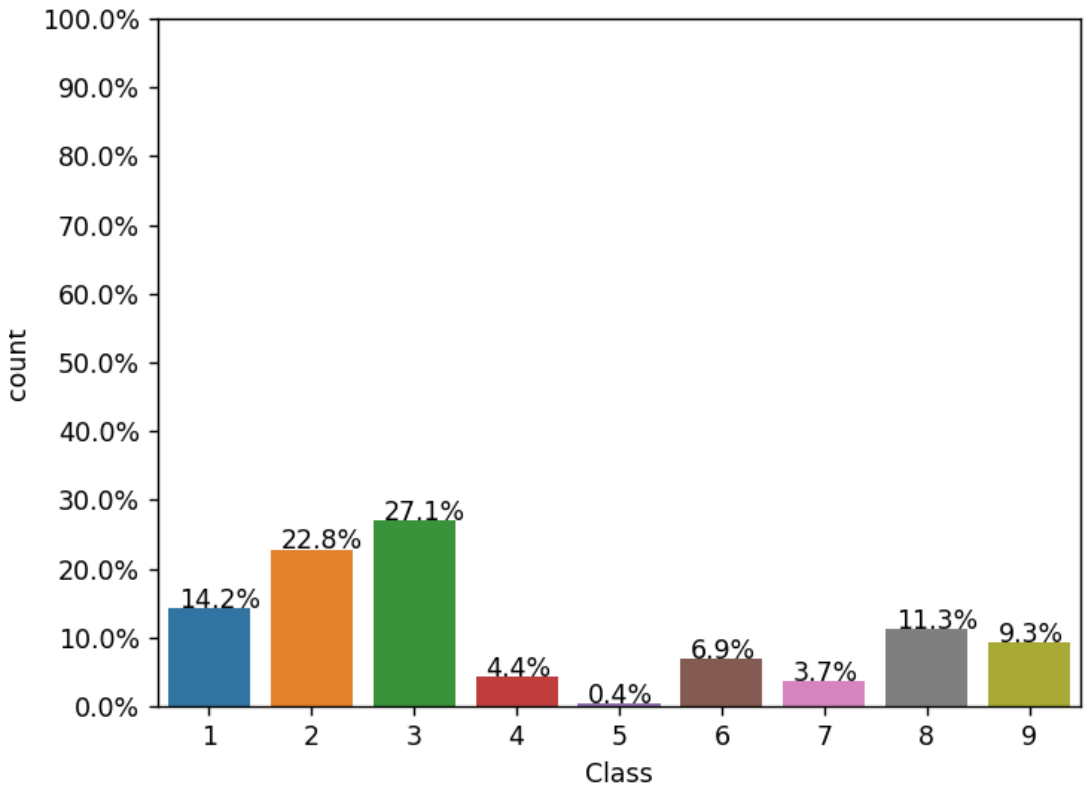
In [ ]:

```
#separating byte files and asm files

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

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

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

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

3.1. Distribution of malware classes in whole data set

In [2]:

```
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

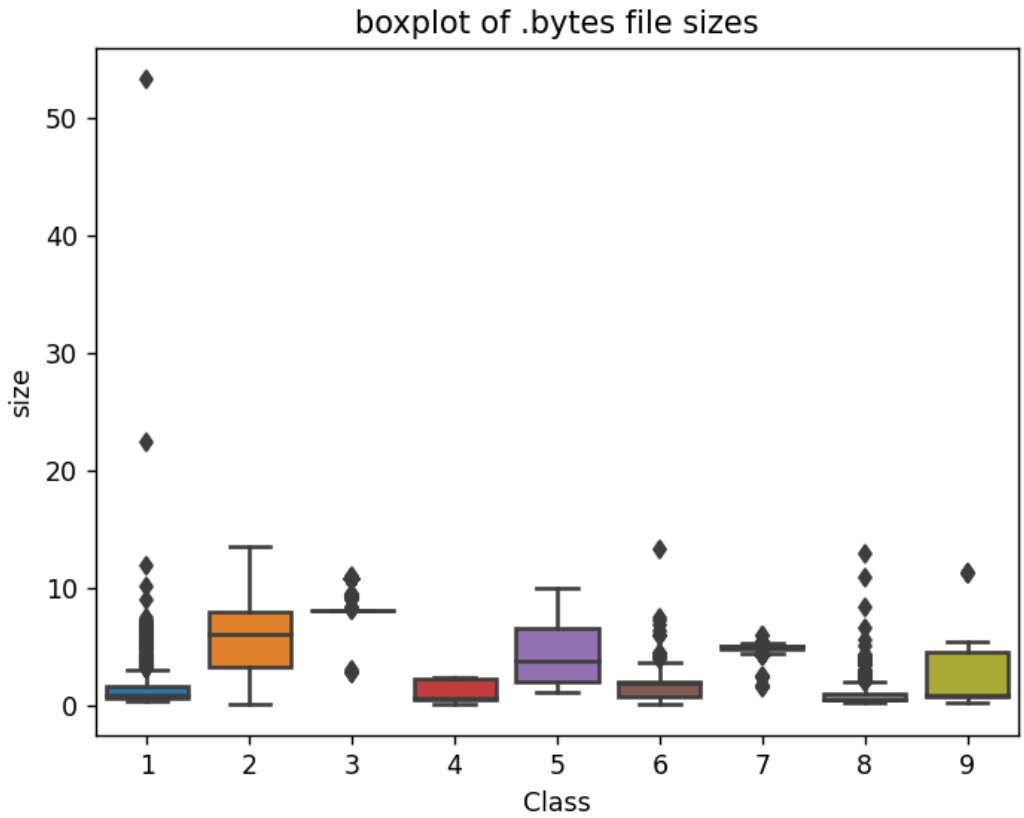
In [3]:
```
#file sizes of byte files


files=os.listdir('byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507,
st_dev=3561571700, st_nlink=1, st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522,
st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com
/python/os_stat.htm
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e
the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':cla
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 [22]:

```
#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 [ ]:

```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000

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

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



#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,
byte_feature_file.write("\n")
for file in files:
    filenames2.append(file)
```

```
                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 [ ]:

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

Out[ ]:

	ID	0	1	2	3	4	5	6	7	8	...	f7	f8
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804	3687
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451	6536

2 rows × 258 columns

In [ ]:

```
data_size_byte.head(2)
```

Out[ ]:

	ID	size	Class
0	01azqd4InC7m9JpocGv5	4.234863	9
1	01IsoiSMh5gxyDYTl4CB	5.538818	2

In [ ]:
```
byte_features_with_size = byte_features.merge(data_size_byte,
on='ID')
byte_features_with_size.to_csv("result_with_size.csv")
byte_features_with_size.head(2)
```

Out[ ]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa
0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	3101	3211
1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	439	281

2 rows × 260 columns

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

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

Out[ ]:

	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006

2 rows × 260 columns

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

Out[ ]:

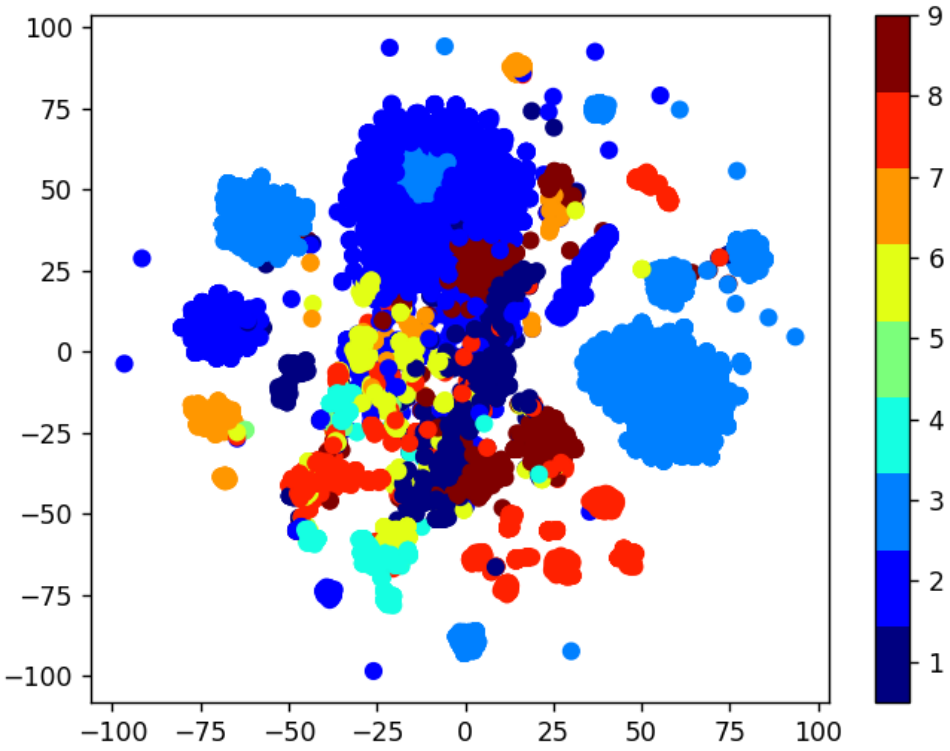
	ID	0	1	2	3	4	5	6
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.00
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.00

	ID	0	1	2	3	4	5	6	
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.00
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.00
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.00

3.2.4 Multivariate Analysis

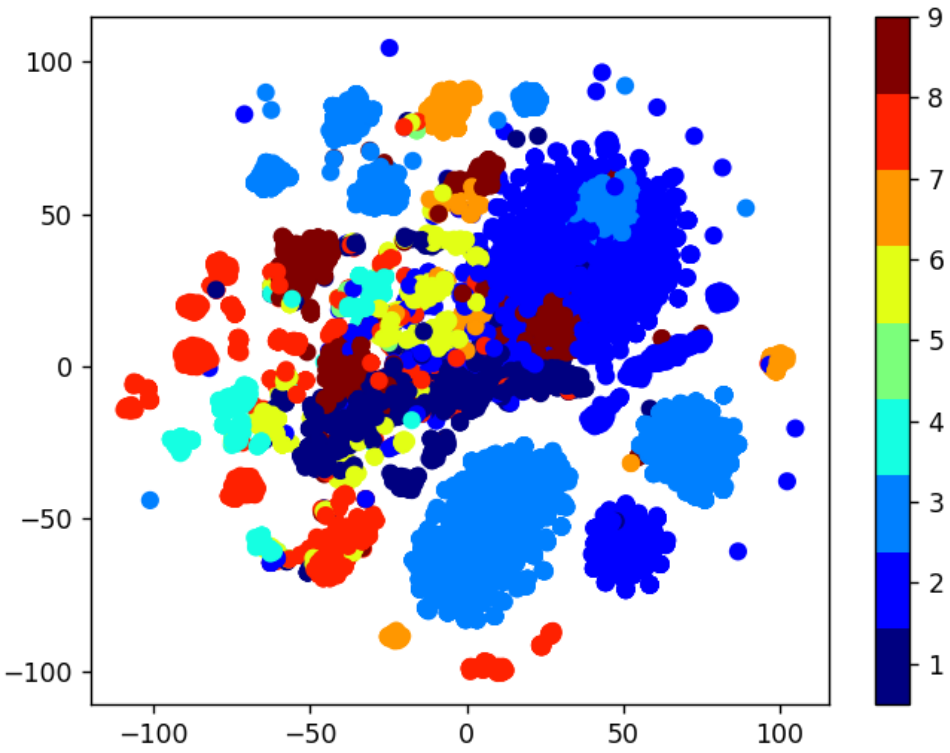
In [ ]:

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



In [ ]:

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



Train Test split

In [ ]:

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

In [ ]:

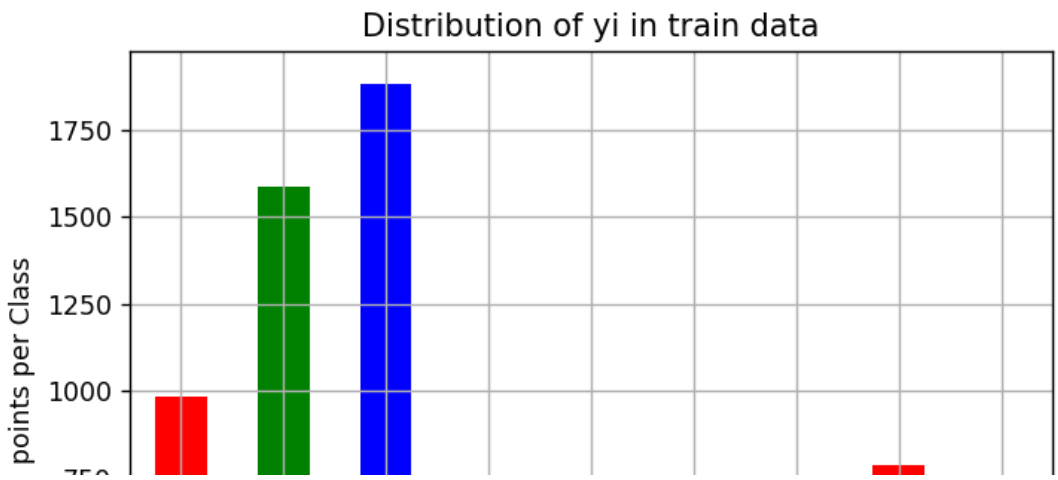
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:',
X_cv.shape[0])
```

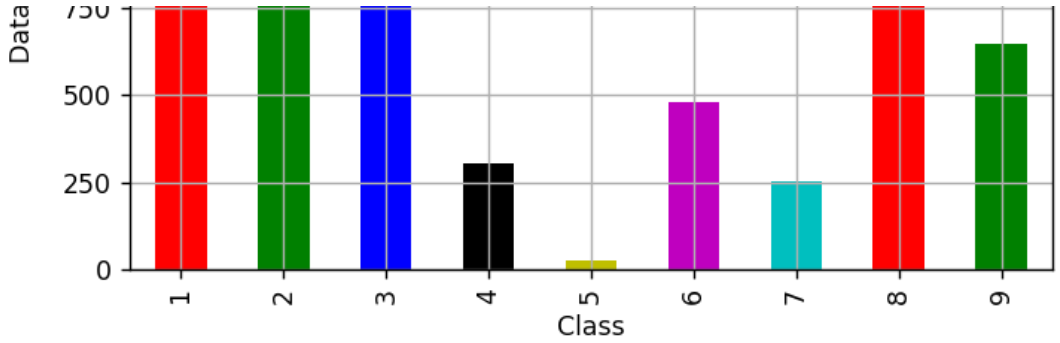
```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [ ]:

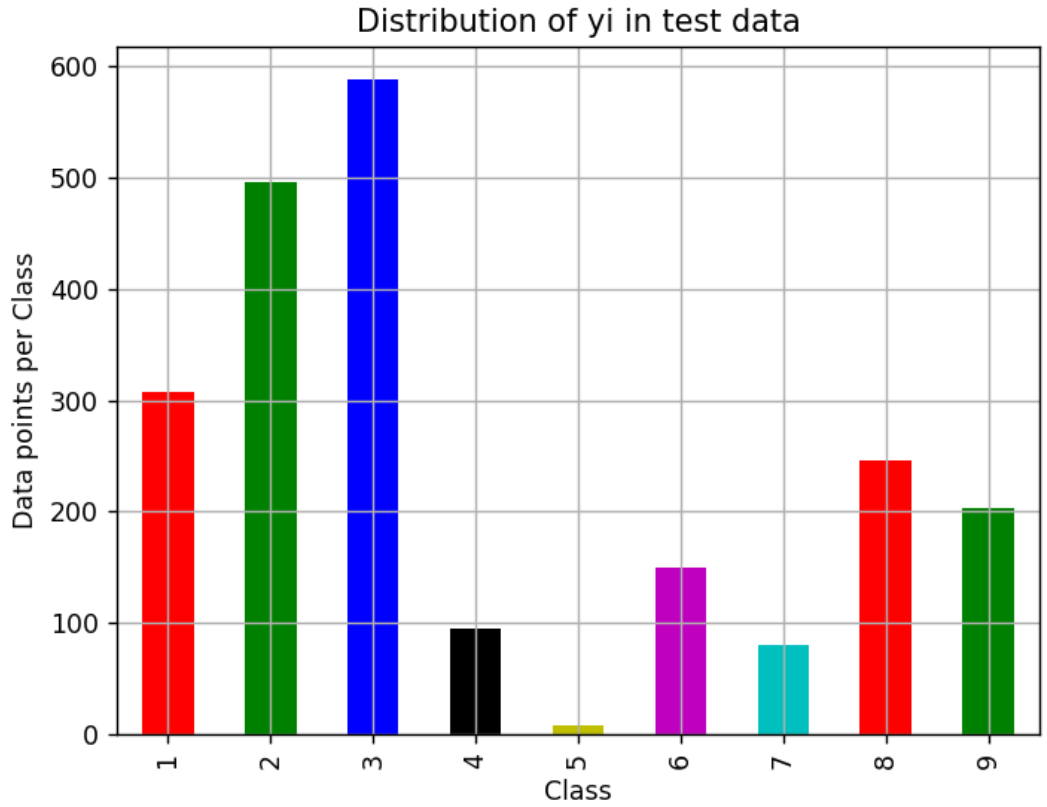
```
# it returns a dict, keys as class labels and values as the number
of data points in that class
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

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

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


print('-'*80)
my_colors = '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
```

```
np.round((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),
'%)')
```



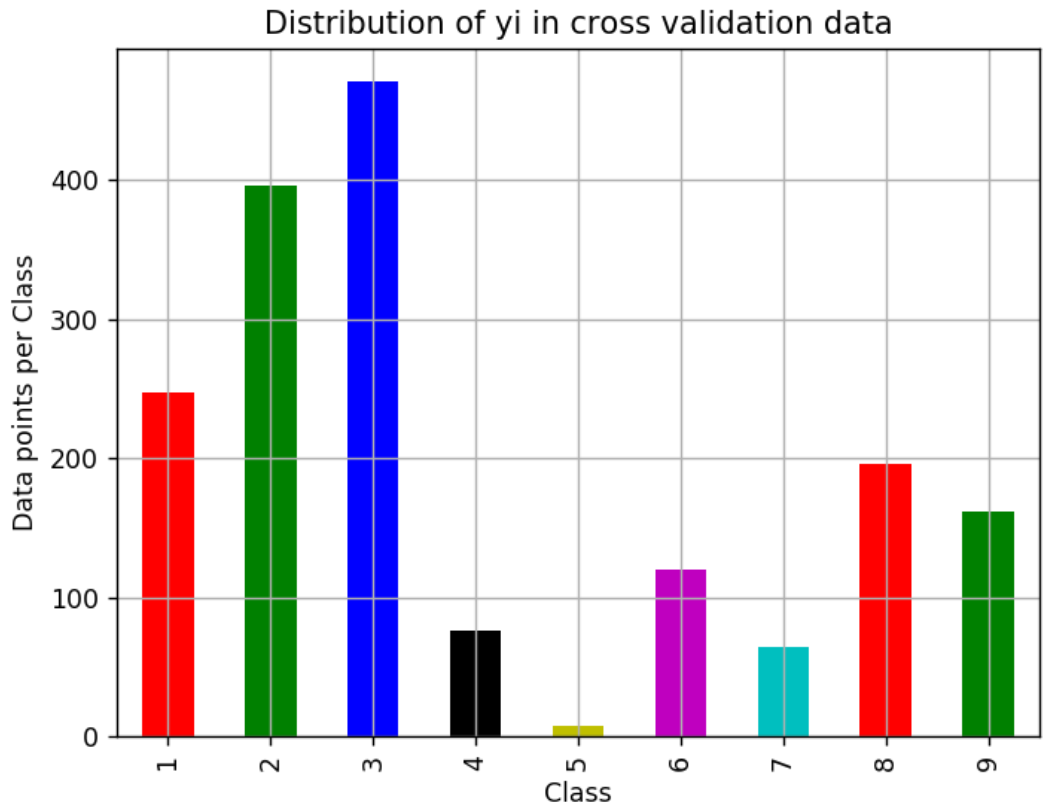



```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 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 [4]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-
np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points
of class i are predicted class j

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

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
```

```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

4. Machine Learning Models

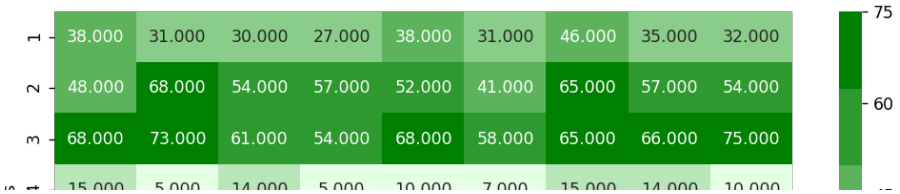
4.1. Machine Leaning Models on bytes files

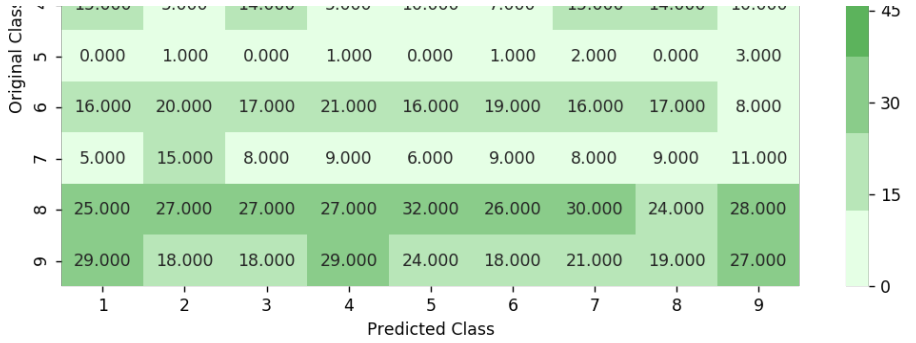
4.1.1. Random Model

In [ ]:

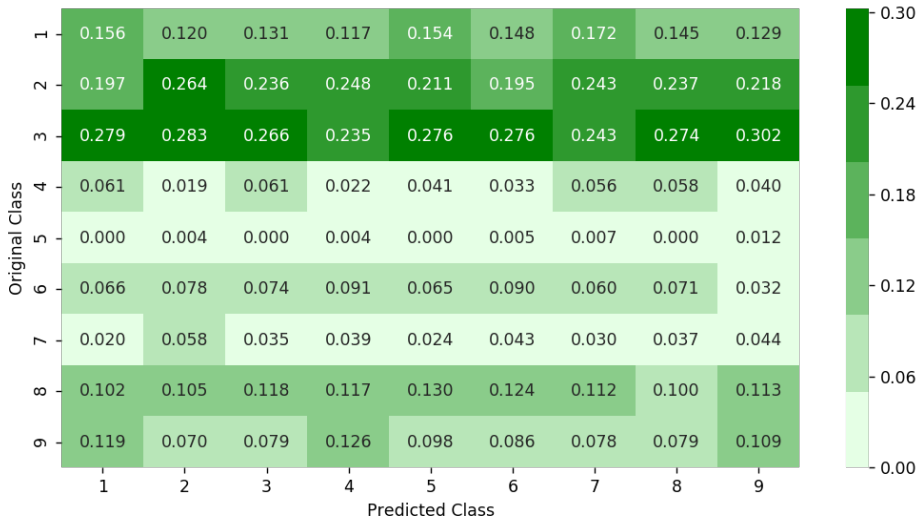
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the
numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

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



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

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

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



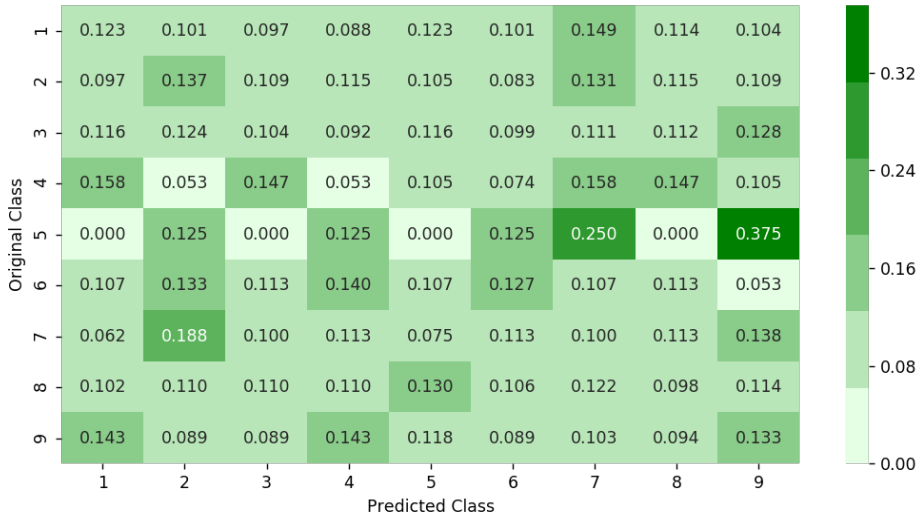


---------- Precision matrix ----------



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

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



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

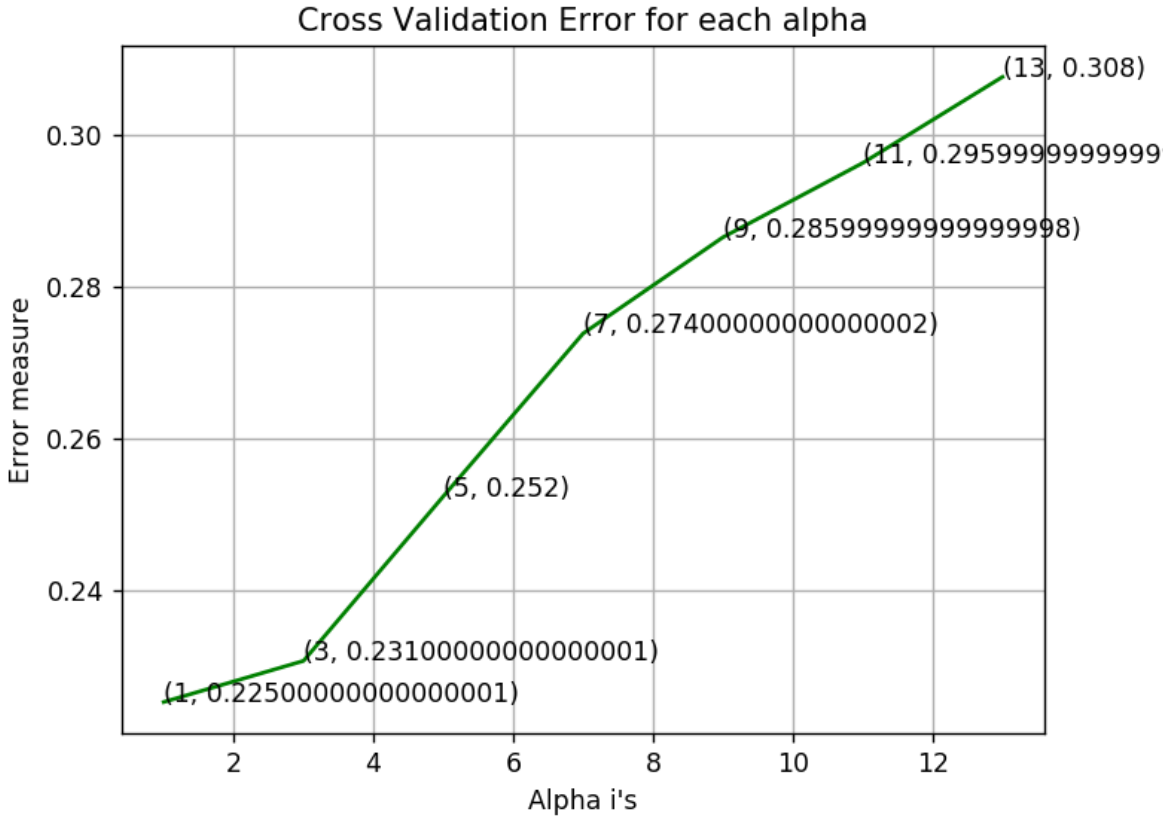
4.1.2. K Nearest Neighbour Classification

In [ ]:

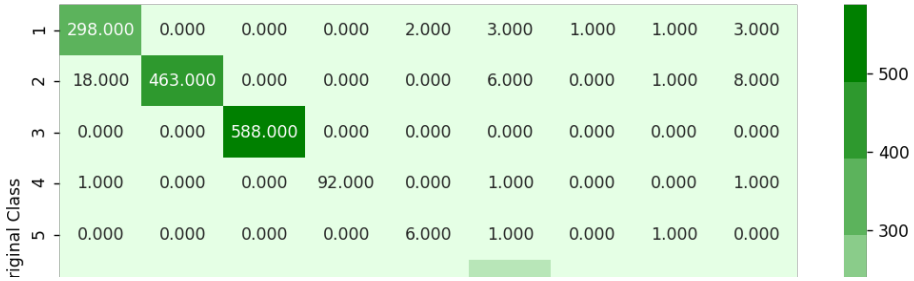
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated
/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform',
algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

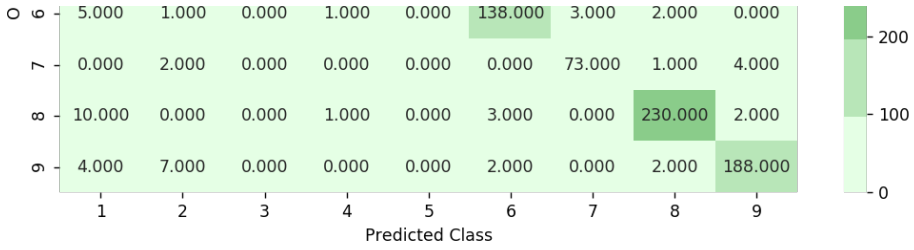
```
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y,
labels=k_cfl.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

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

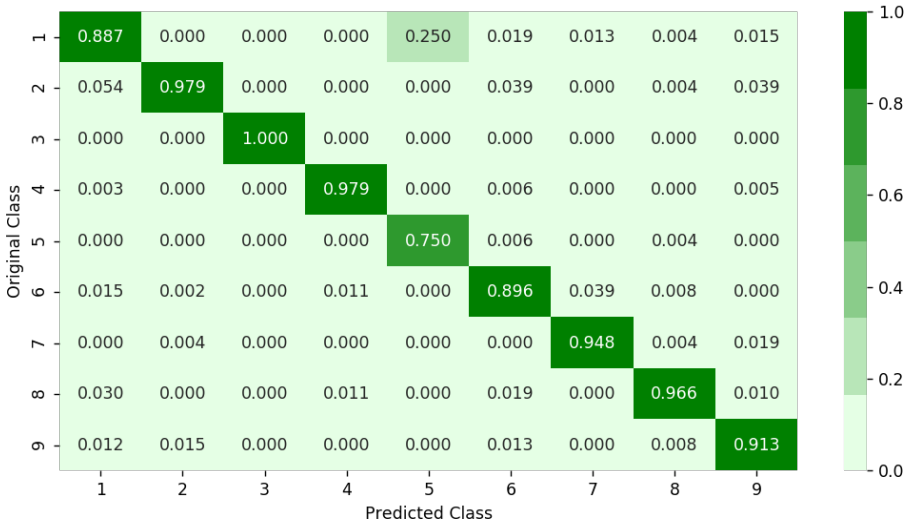



```
For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.225386237304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
------------------------------------------------------ Confusion matrix
-----------------------------------------------------
```

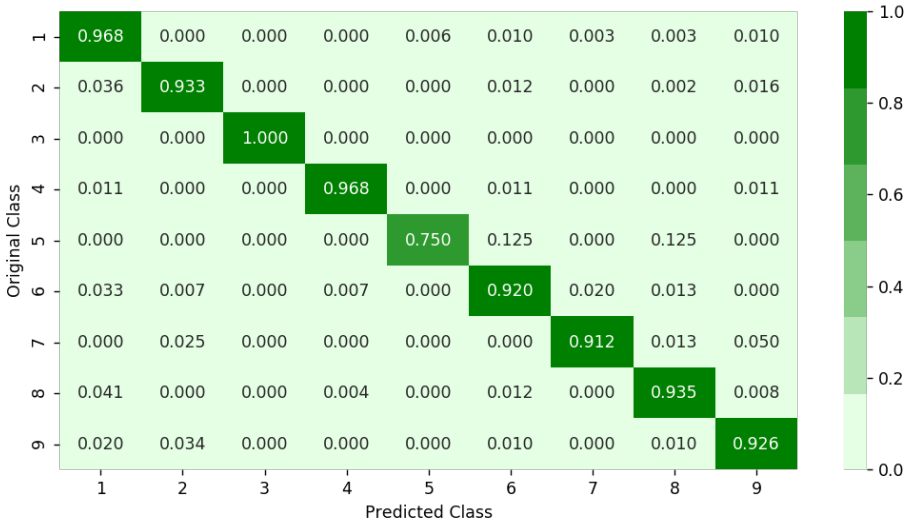





```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```




```
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.]
```

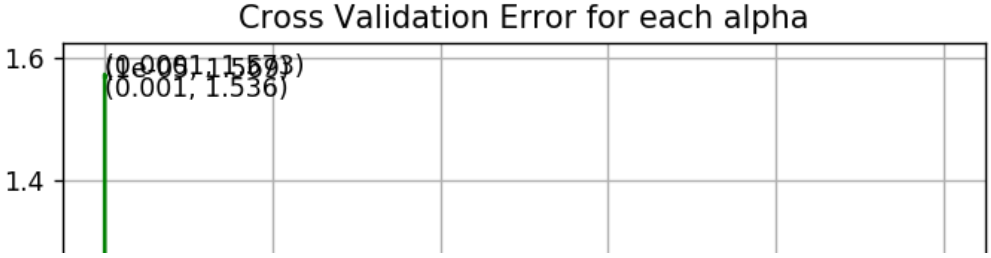
4.1.3. Logistic Regression

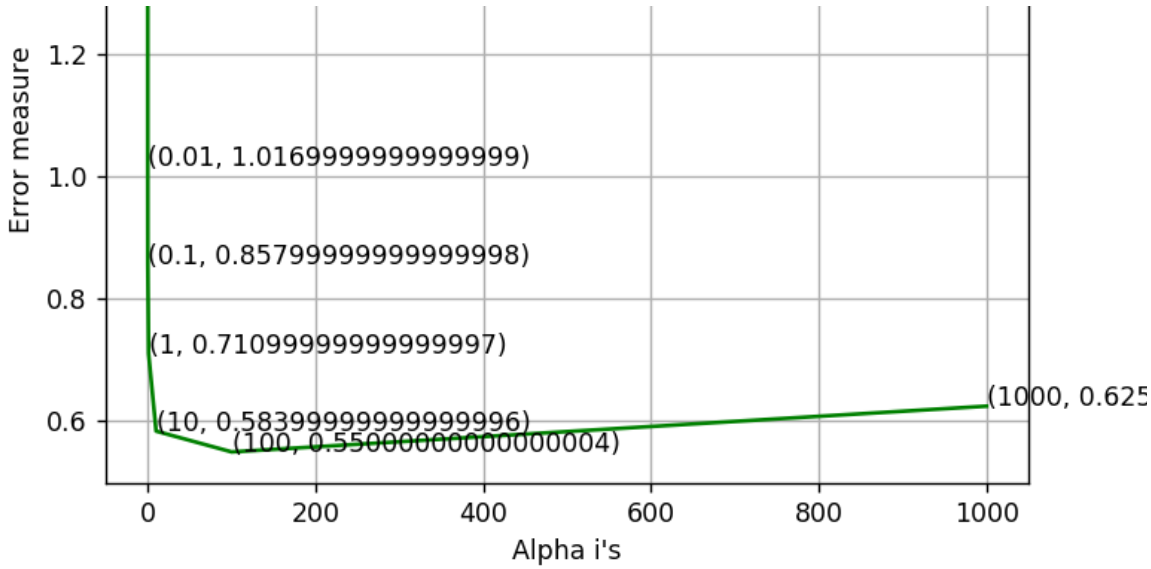
In [ ]:

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

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

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

alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:

logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced'
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y,
labels=logisticR.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
```

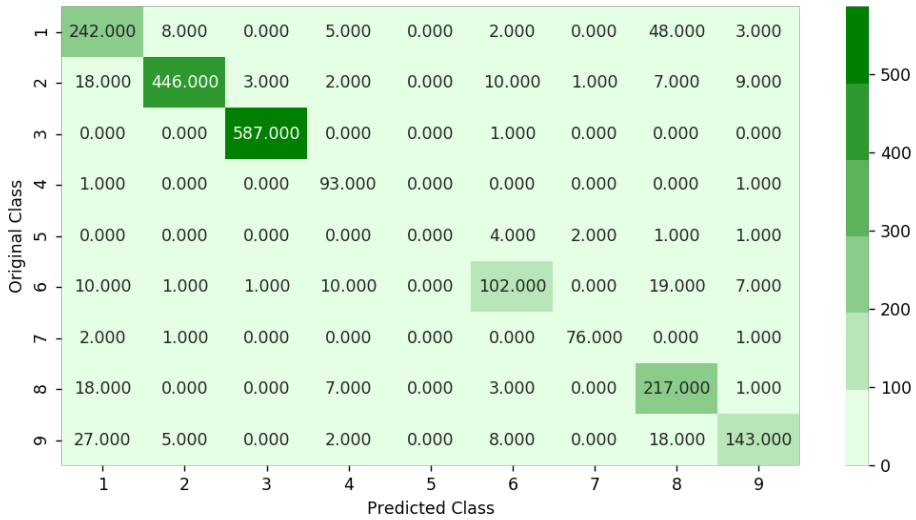
```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_w
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 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =  100 is 0.549929846589
log_loss for c =  1000 is 0.624746769121
```

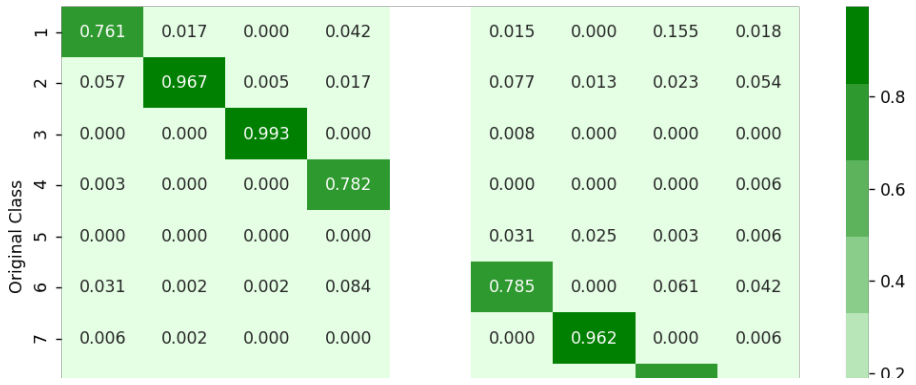


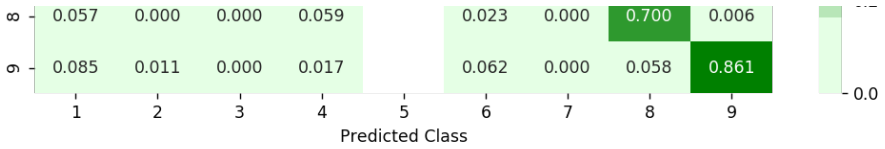



```
log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```

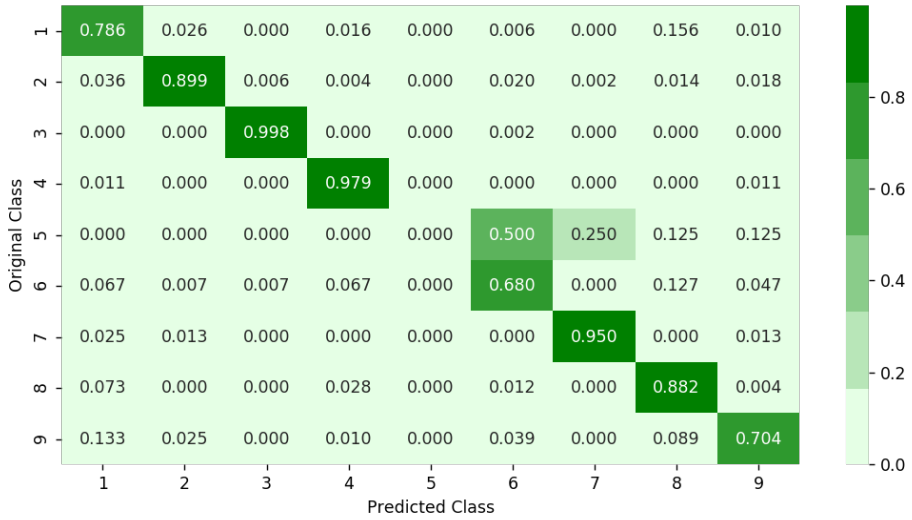



```
---------------------------------------------------- Precision matrix
-----------------------------------------------------
```






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




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

4.1.4. Random Forest Classifier

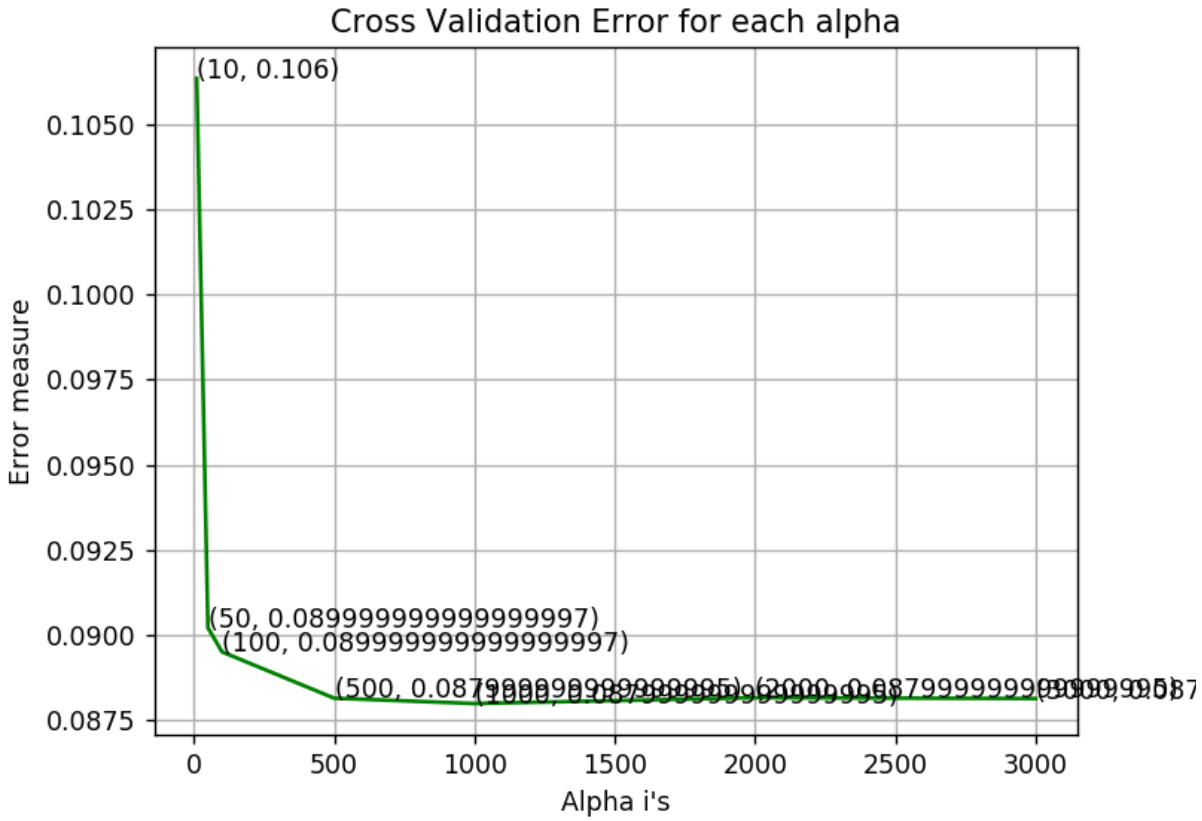
In [ ]:

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

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

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

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

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

r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
```

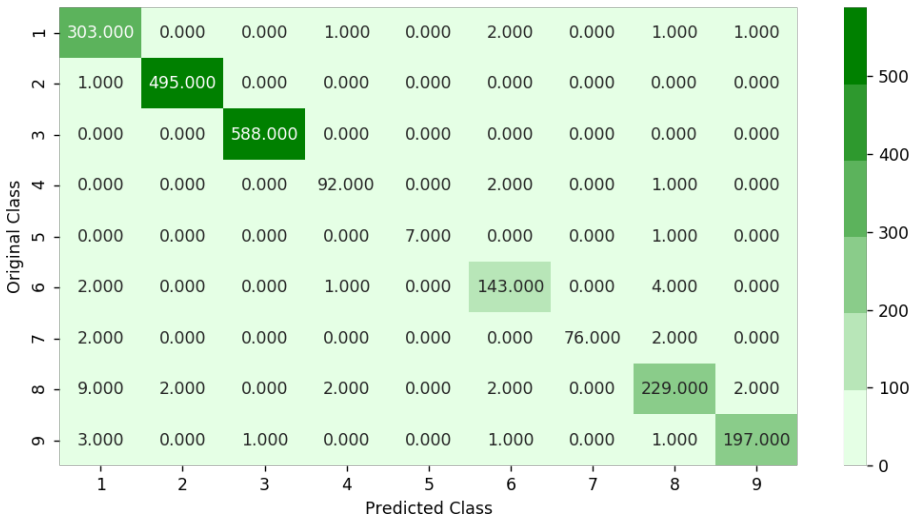
```
best_alpha = np.argmin(cv_log_error_array)

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_st
1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
log_loss for c =  10 is 0.106357709164
```

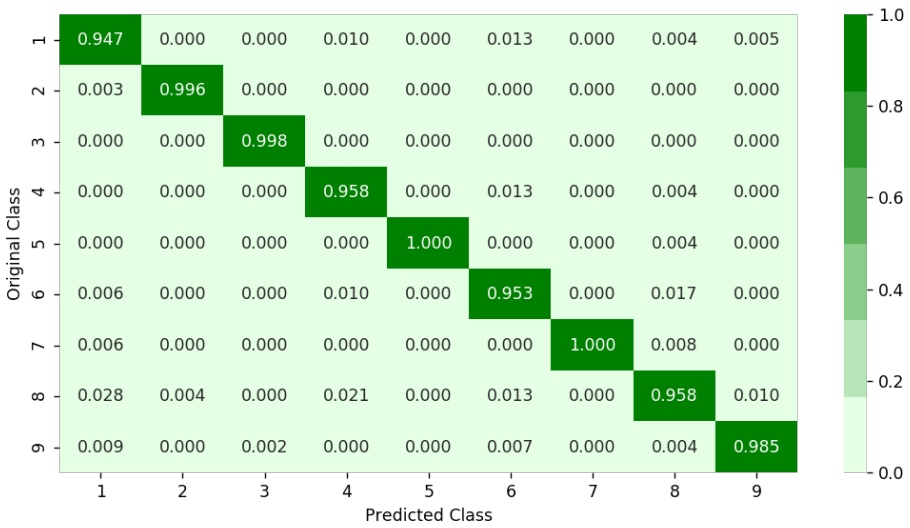
```
log_loss for c =  50 is 0.0902124124145
log_loss for c =  100 is 0.0895043339776
log_loss for c =  500 is 0.0881420869288
log_loss for c =  1000 is 0.0879849524621
log_loss for c =  2000 is 0.0881566647295
```



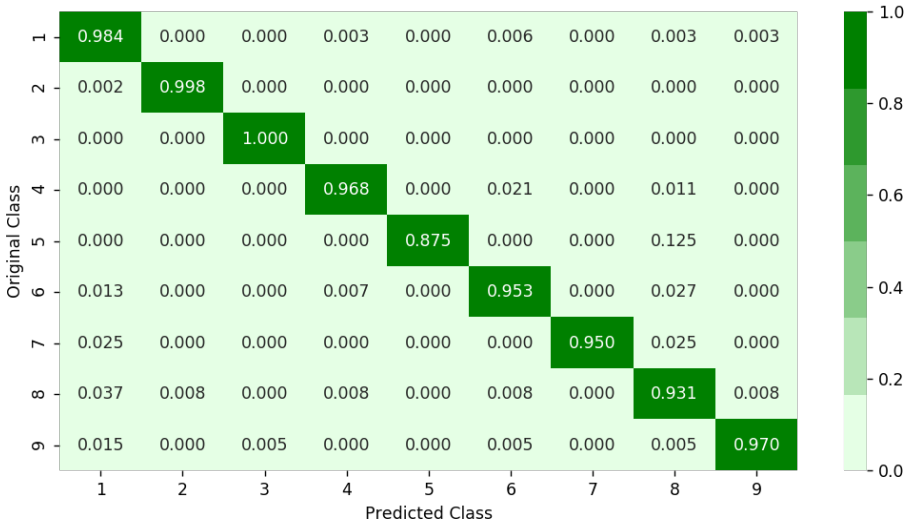

```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.087984952
4621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```



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




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




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

4.1.5. XgBoost Classification

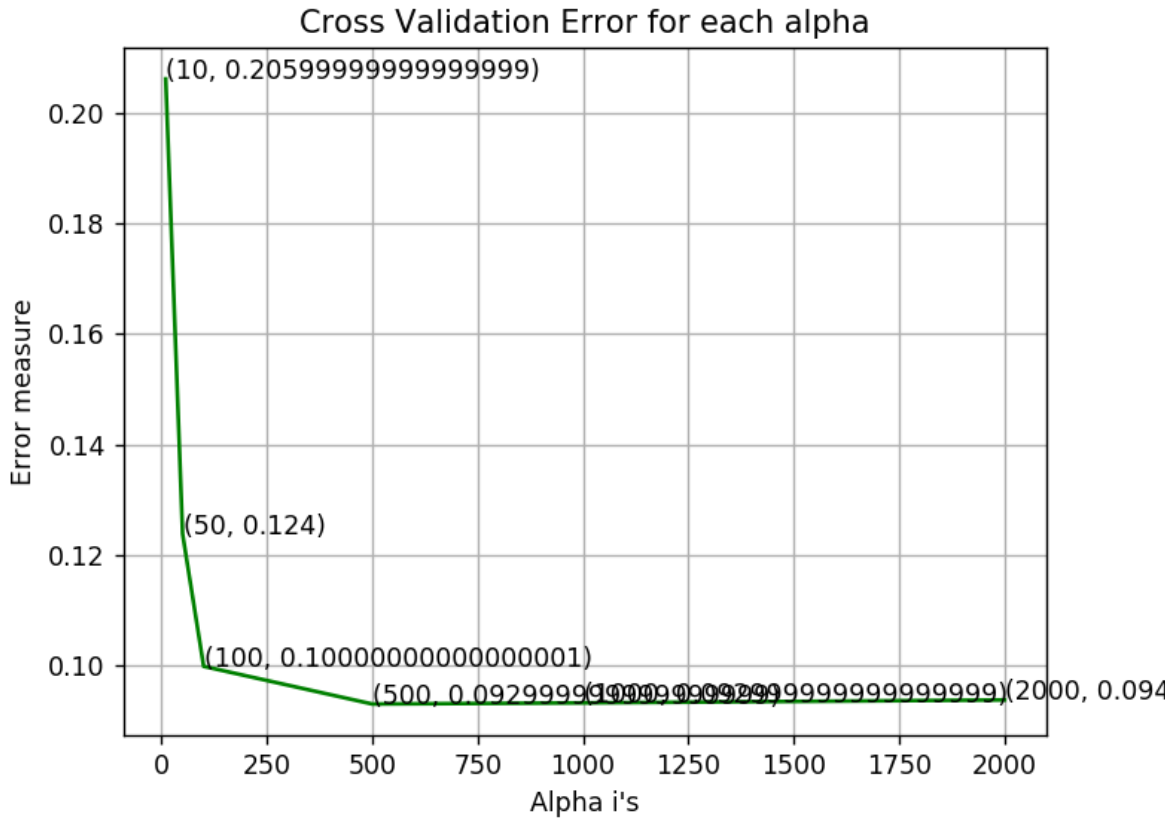
In [ ]:

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

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None,
early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with
data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-
course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-
course-online/lessons/what-are-ensembles/
# ------------------------

alpha=[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)
```

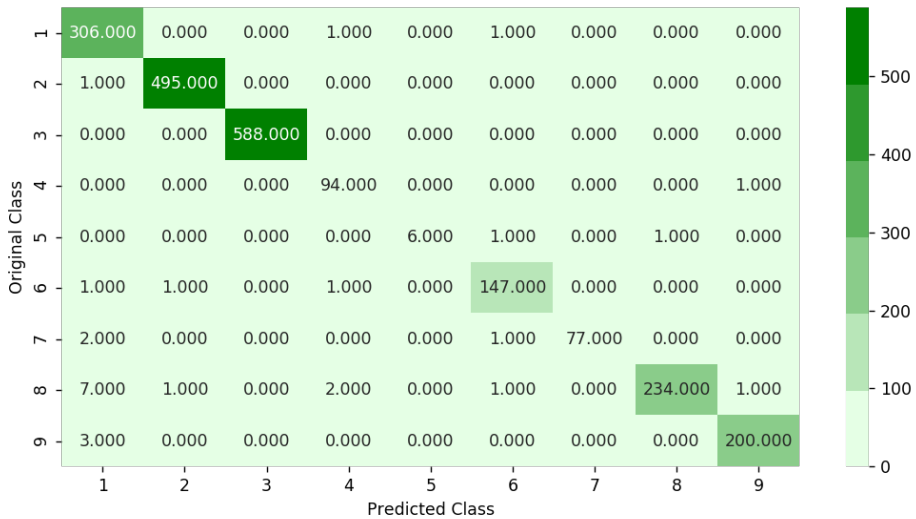
```
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  10 is 0.20615980494
log_loss for c =  50 is 0.123888382365
log_loss for c =  100 is 0.099919437112
```

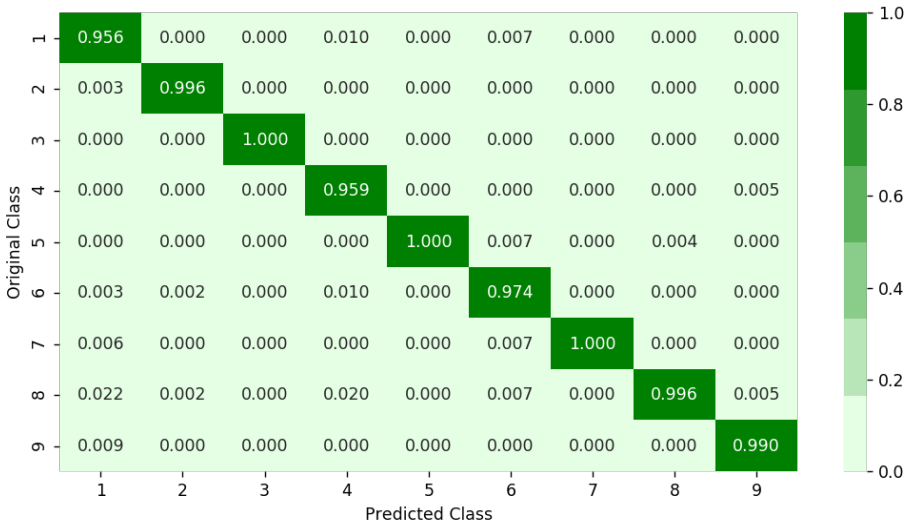
```
log_loss for c =  500 is 0.0931035681289
log_loss for c =  1000 is 0.0933084876012
log_loss for c =  2000 is 0.0938395690309
```



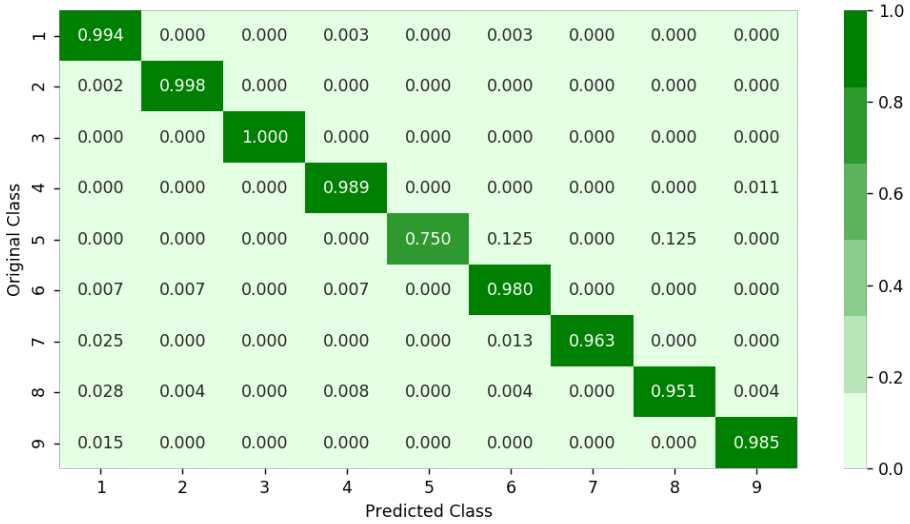

```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.0931035681
289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```




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




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




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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

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


prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[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,verbos
1,)
random_cfl1.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  9.3min remaining:  5.4m
in
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 10.1min remaining:  3.1m
in
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6m
in
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 14.2min finished
```

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

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

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

In [ ]:

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

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

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

x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05,
colsample_bytree=1, max_depth=3)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
```

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

4.2 Modeling with .asm files

```
There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features.In
parallel we can use all the cores that are present in our computer.


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

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

4.2.1 Feature extraction from asm files

To extract the unigram features from the .asm files we need to process ~150GB of data

Note: Below two cells will take lot of time (over 48 hours to complete)

We will provide you the output file of these two cells, which you can directly use it

In [ ]:

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

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

In [ ]:

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

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

    prefixes =
['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.ec
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn',
'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr',
'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    #best keywords that are taken from different blogs
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and
special registers
    #All the registers which are taken are best
    registers=
['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library
/codecs.html#codecs.ignore_errors
```

```
                # https://www.tutorialspoint.com/python3
/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and
'CODE' segments
                        if registers[i] in li and ('text' in l or
'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
```

```
['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.ec
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn',
'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr',
'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=
['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('second/'+f,encoding='cp1252',errors
='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
```

```
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()

# same as smallprocess() functions
def thirdprocess():
    prefixes =
['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.ec
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn',
'nop', 'sub', 'inc', 'dec', 'add','imul', 'xchg', 'or', 'shr',
'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
    keywords = ['.dll','std::',':dword']
    registers=
['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('third')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
```

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

```
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or
'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
```

```
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fifth/'+f,encoding='cp1252',errors
='replace') as fli:
            for lines in fli:
```

```
System
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
```

In [ ]:

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

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	ed
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	1
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	1
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	1
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	1

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

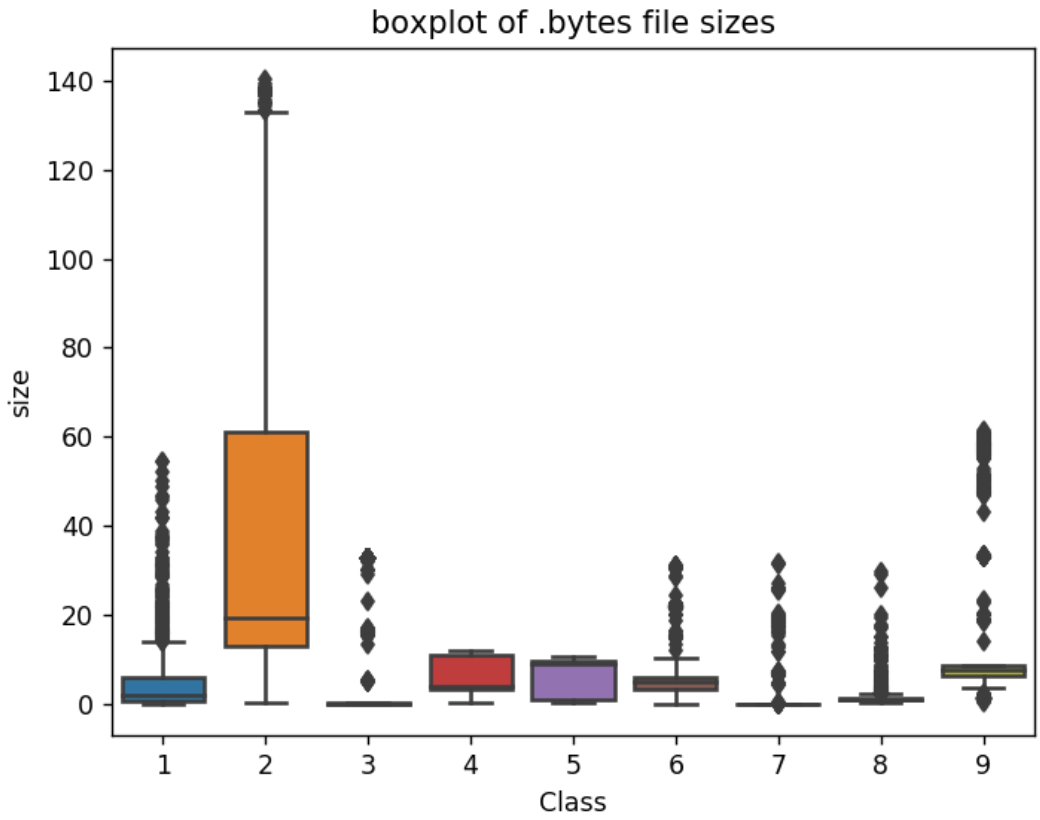
In [ ]:

```
#file sizes of byte files

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

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

4.2.1.2 Distribution of .asm file sizes

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



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

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

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9

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

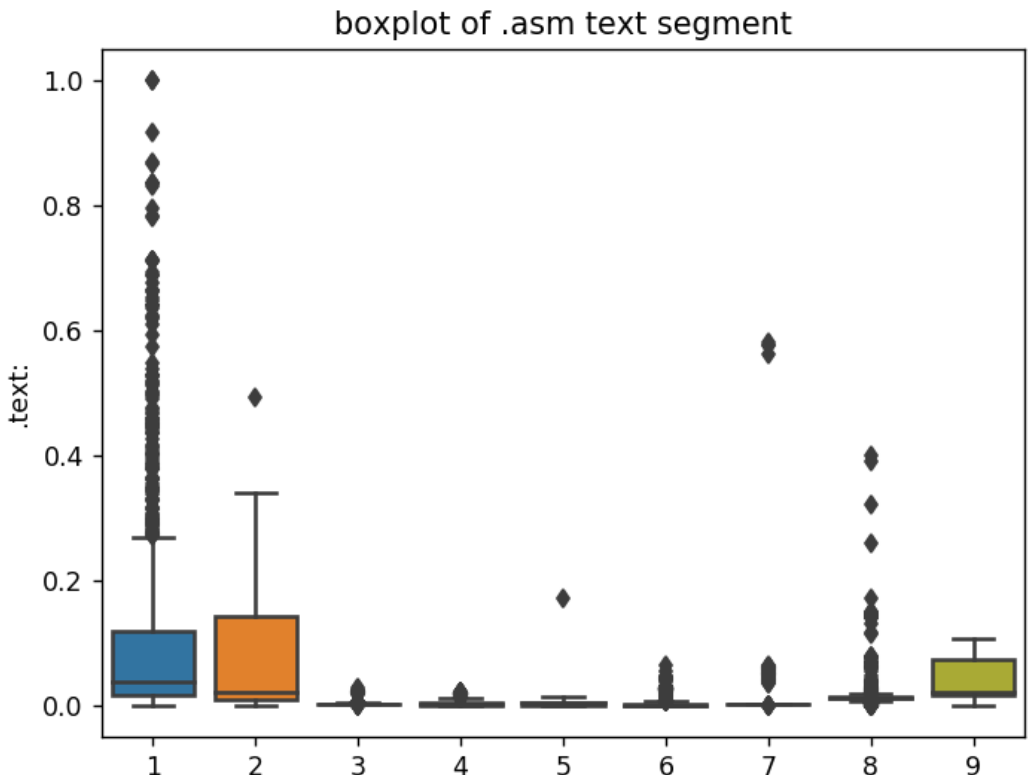
Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.0
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.0
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.0
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.0
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.0

5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

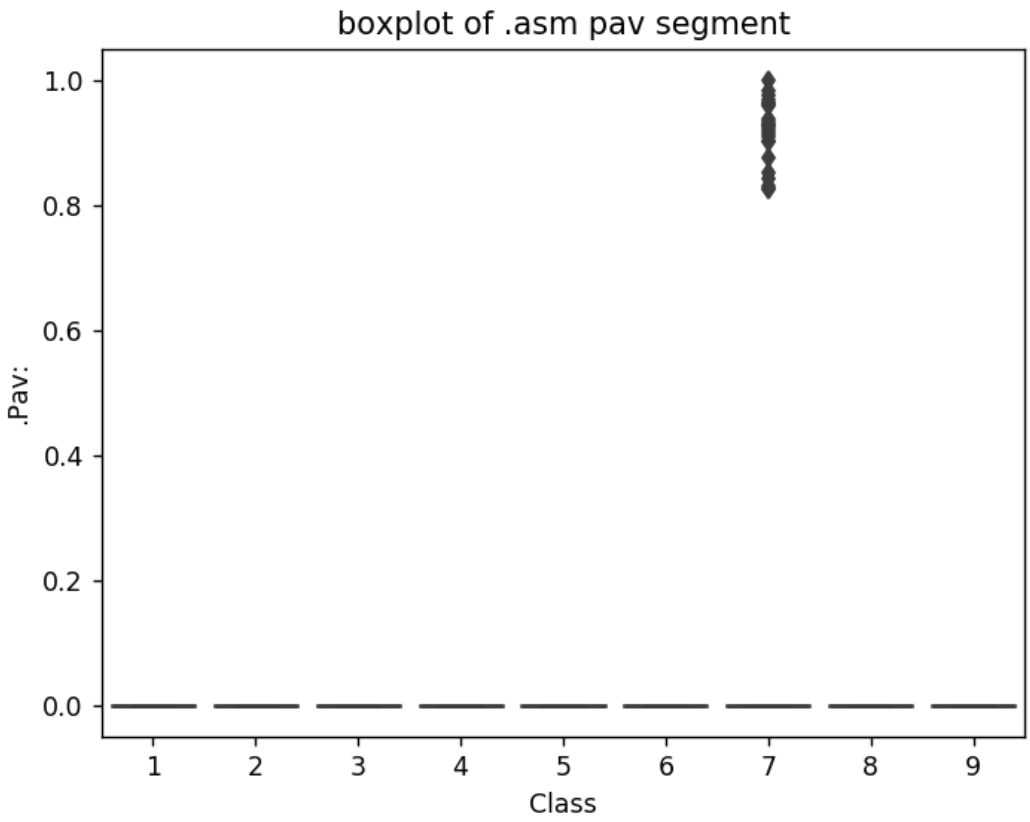
In [ ]:
```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```



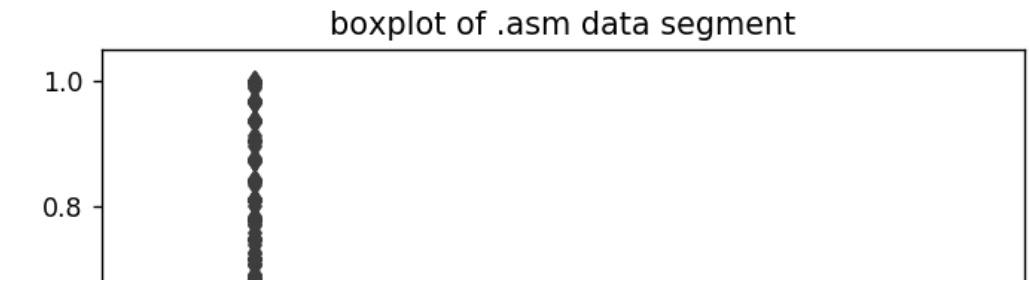
Class

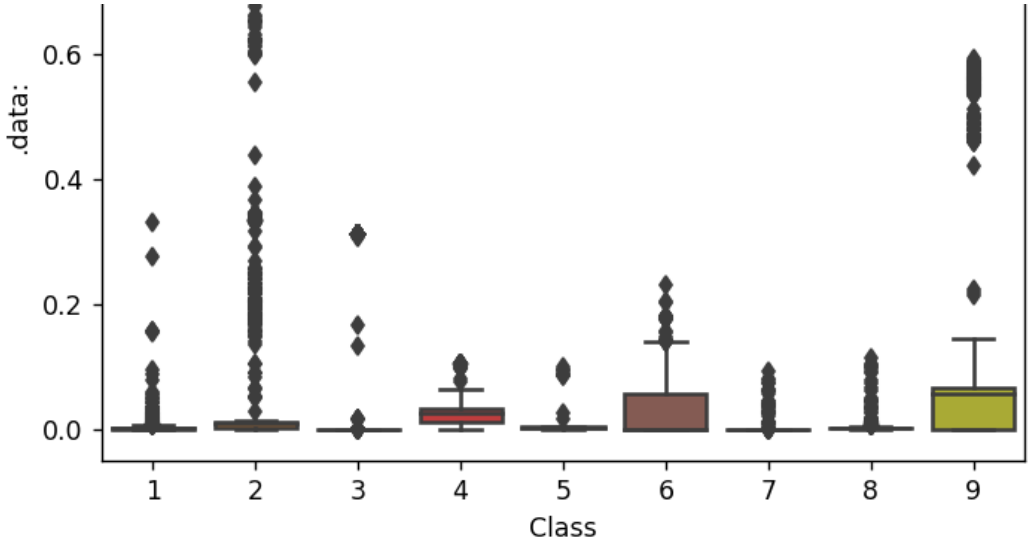
The plot is between Text and class
Class 1,2 and 9 can be easly separated

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



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

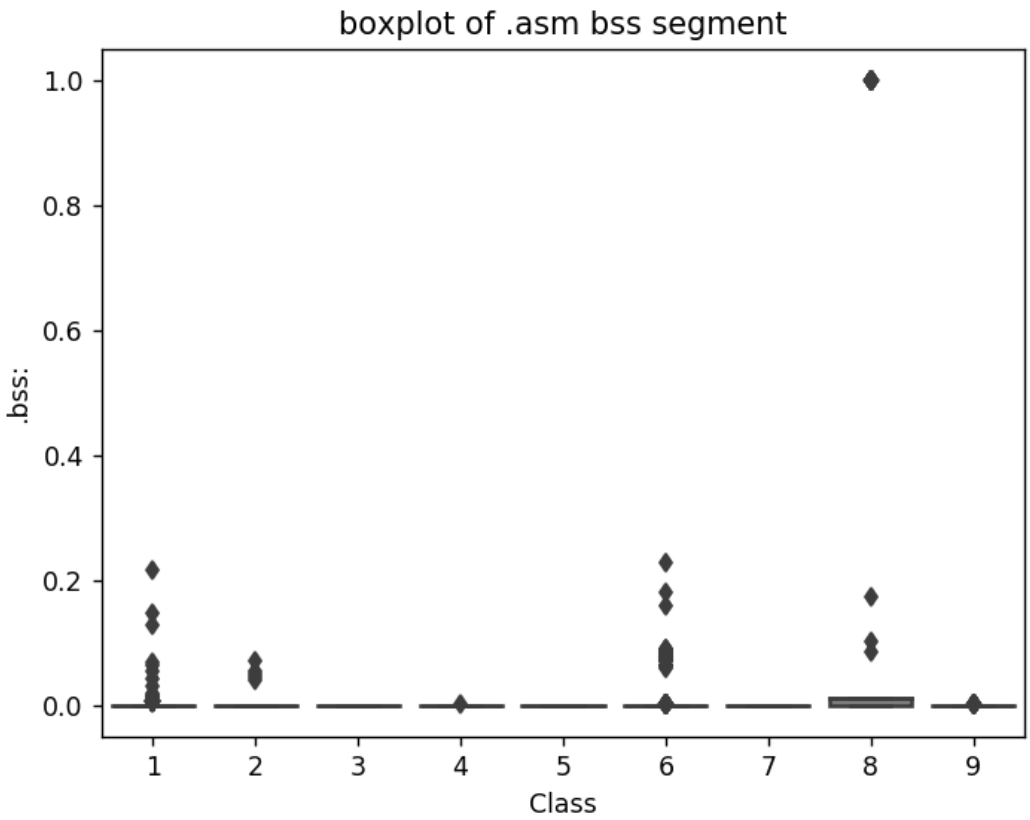




The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

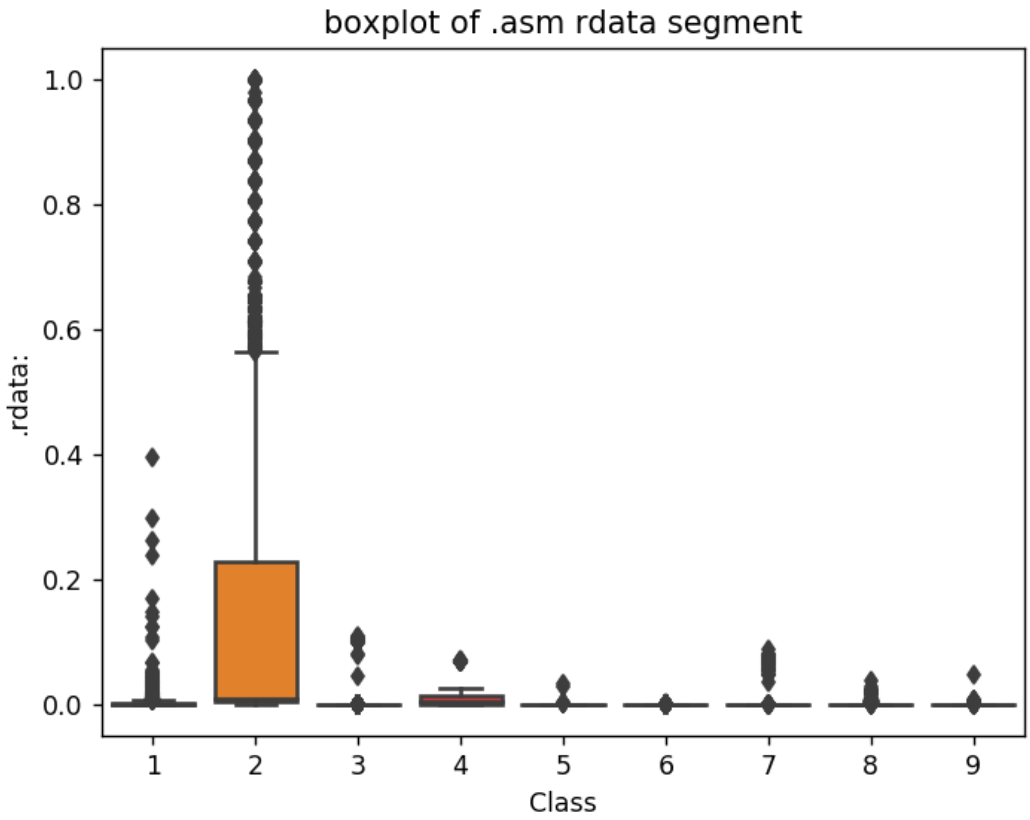
In [ ]:

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



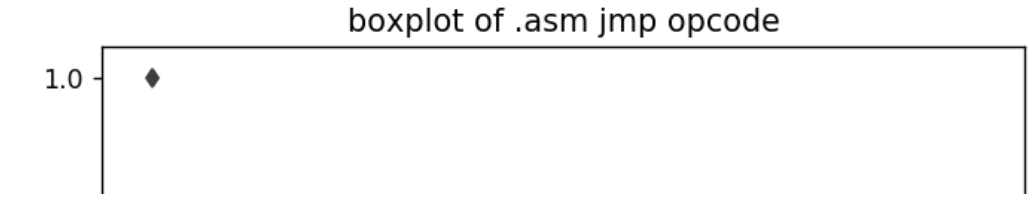
plot between bss segment and class label
very less number of files are having bss segment

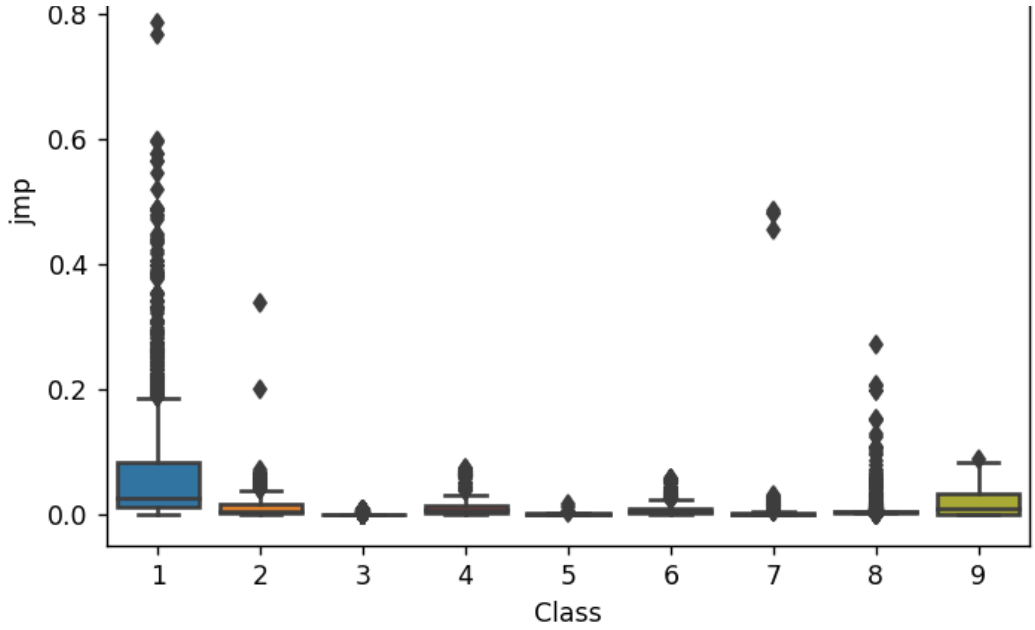
In [ ]:
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



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

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

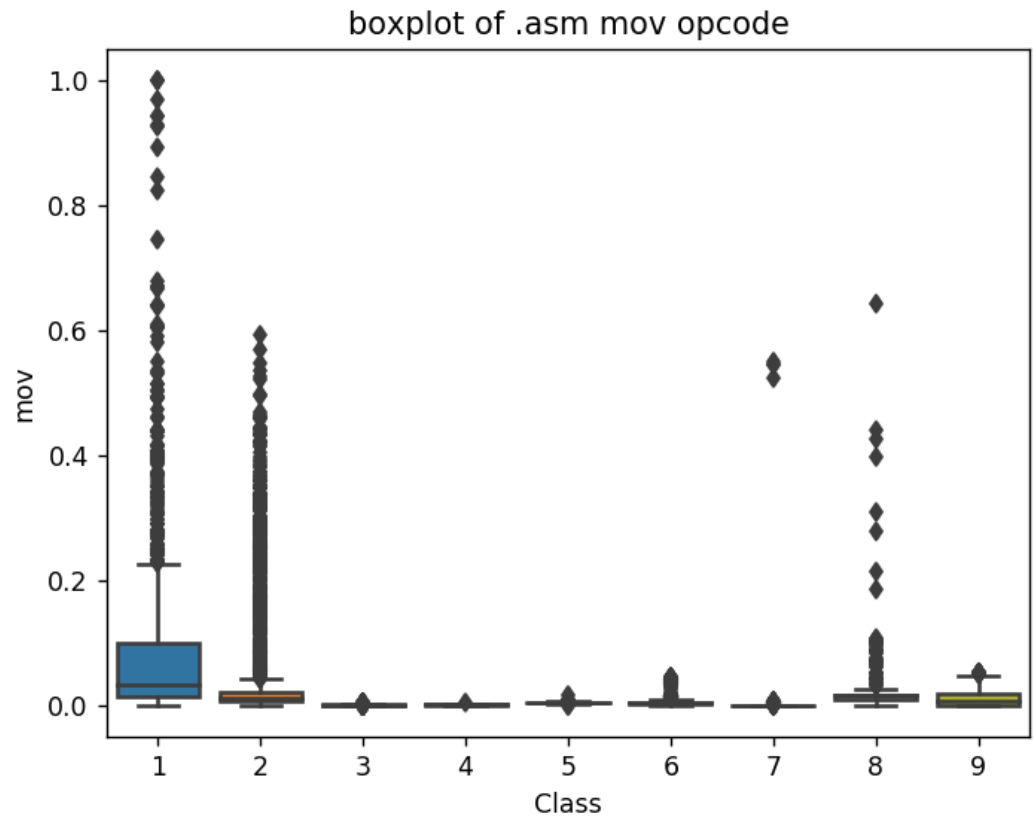




plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

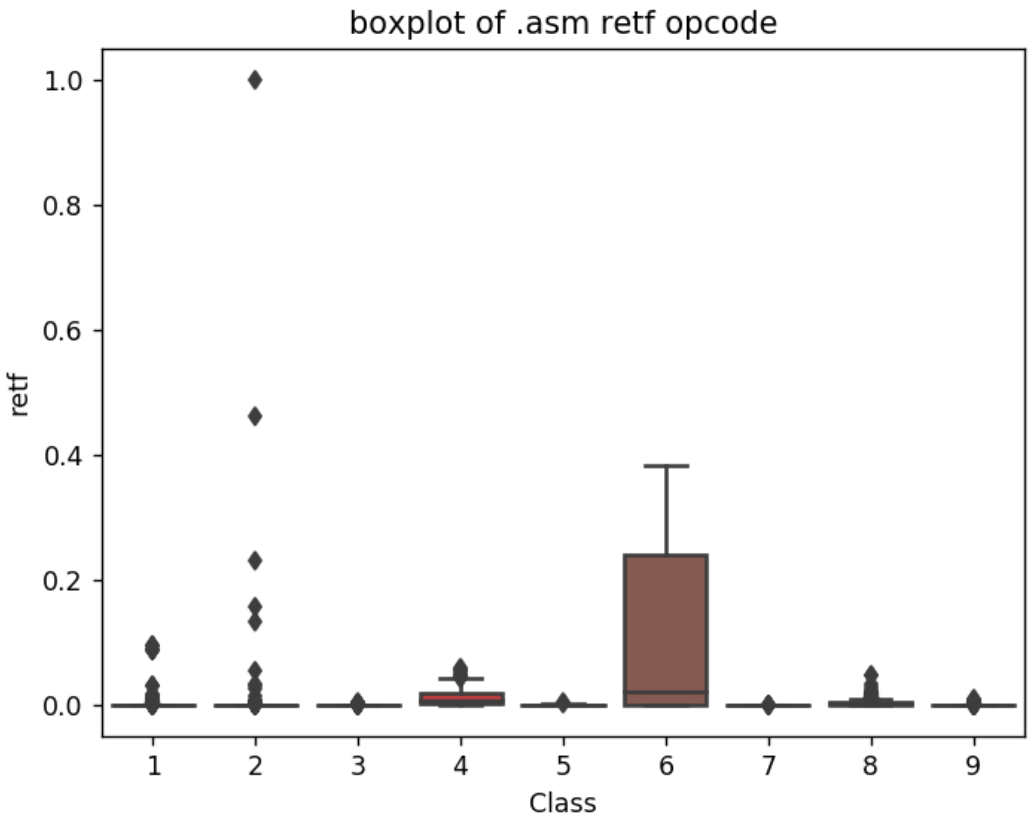
In [ ]:

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



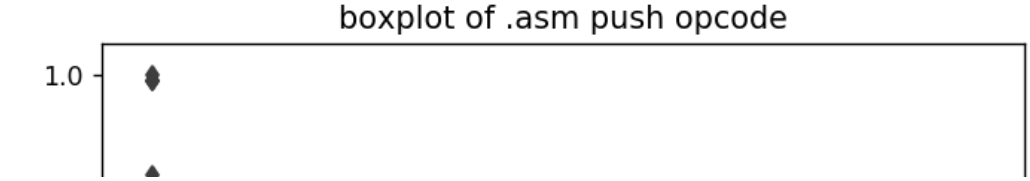
plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

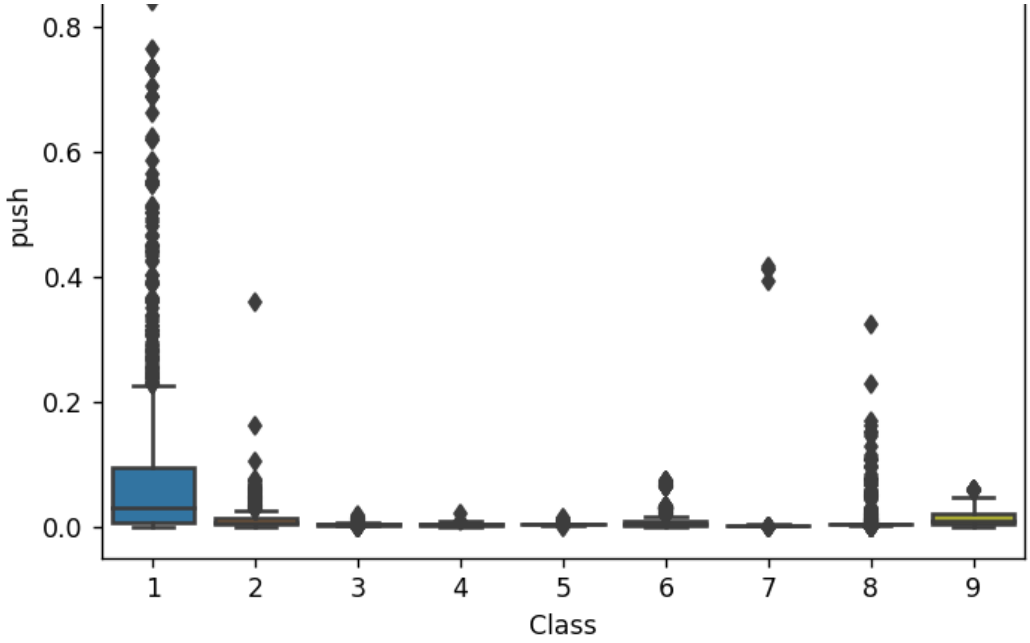
In [ ]:
```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```



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

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



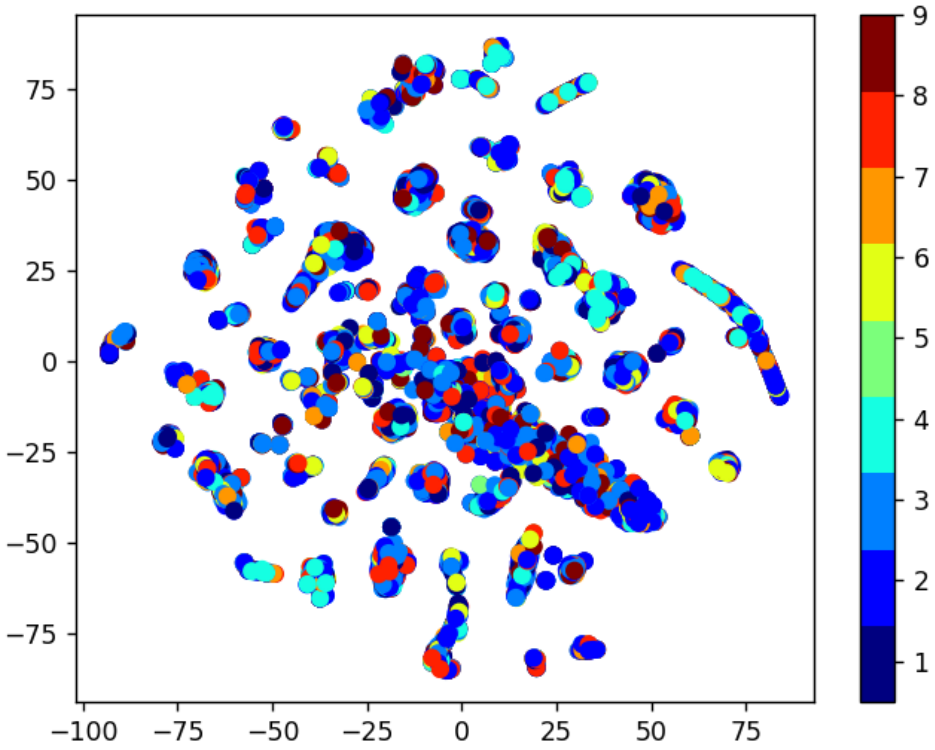



```
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency
1000
```

4.2.2 Multivariate Analysis on .asm file features

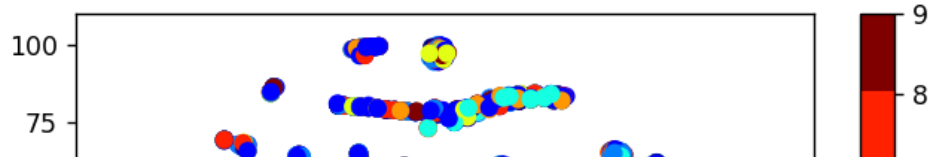
In [ ]:

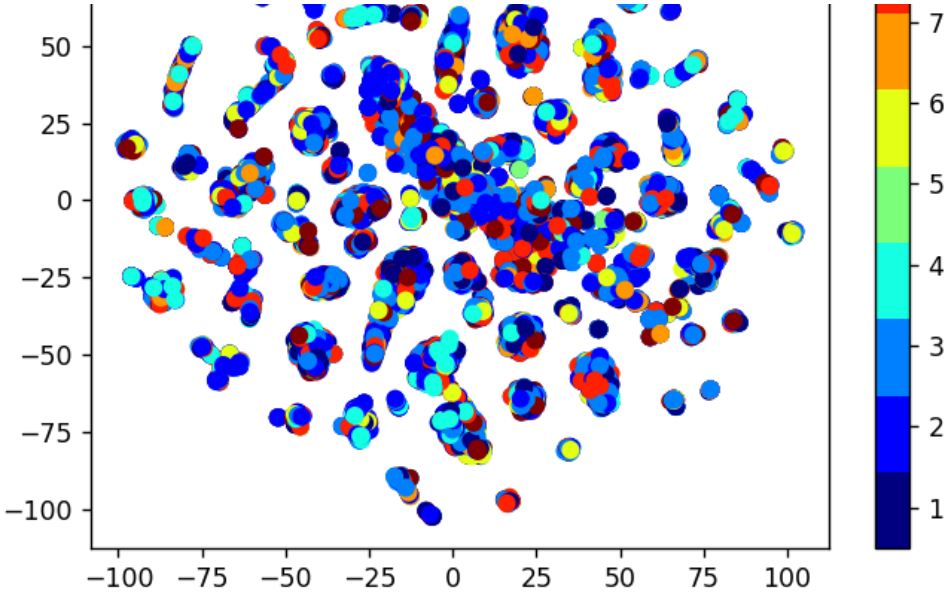
```
# check out the course content for more explantion on tsne
algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online
/lessons/t-distributed-stochastic-neighbourhood-embeddingt-sne-
part-1/

#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_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 [ ]:

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

xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(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()
```





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

In [ ]:
```
X_train_asm, X_test_asm, y_train_asm, y_test_asm =
train_test_split(asm_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm =
train_test_split(X_train_asm,
y_train_asm,stratify=y_train_asm,test_size=0.20)
```

In [ ]:
```
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

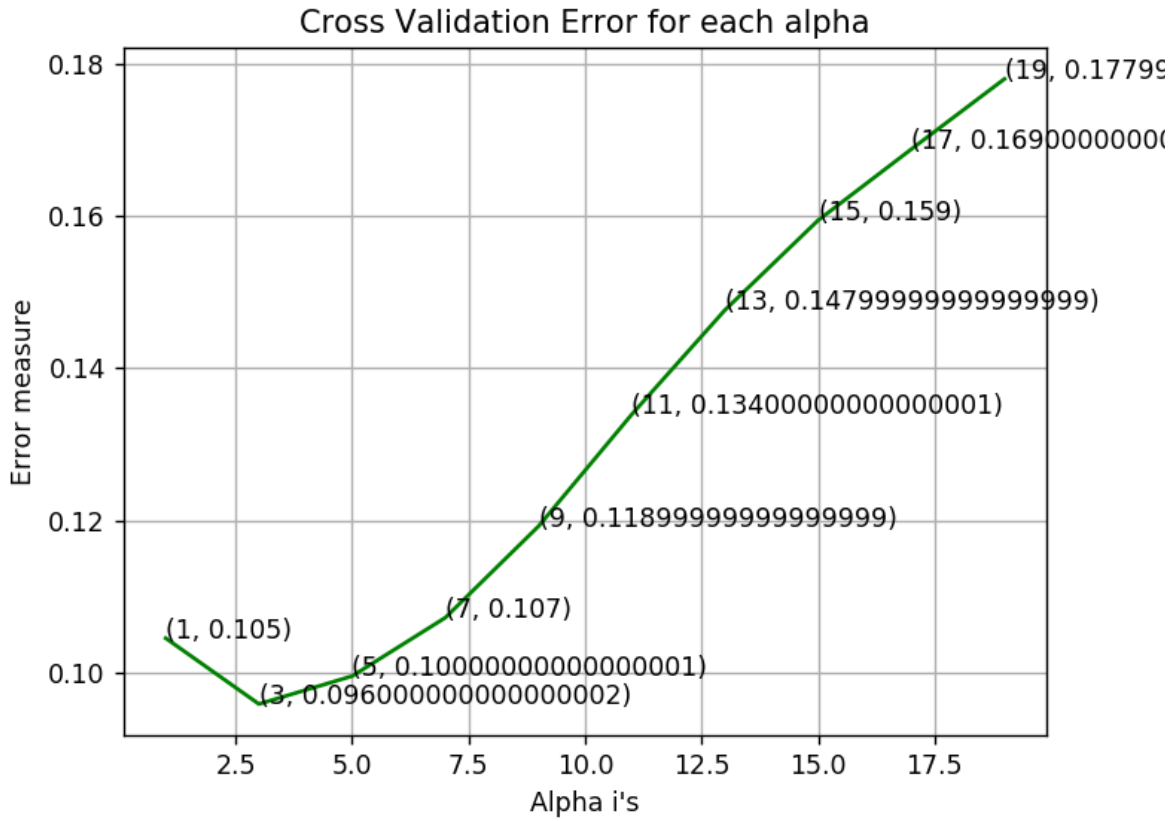
4.4.1 K-Nearest Neigbors

In [ ]:

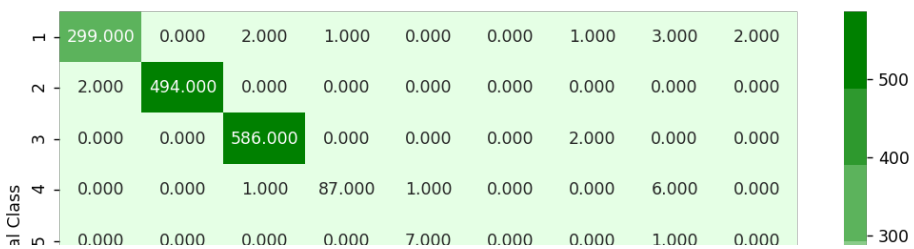
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated
/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform',
algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

# methods of
# fit(X, y) : Fit the model using X as training data and y as
target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data
X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-
course-online/lessons/k-nearest-neighbors-geometric-intuition-with-
a-toy-example-1/
#--------------------------------------


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

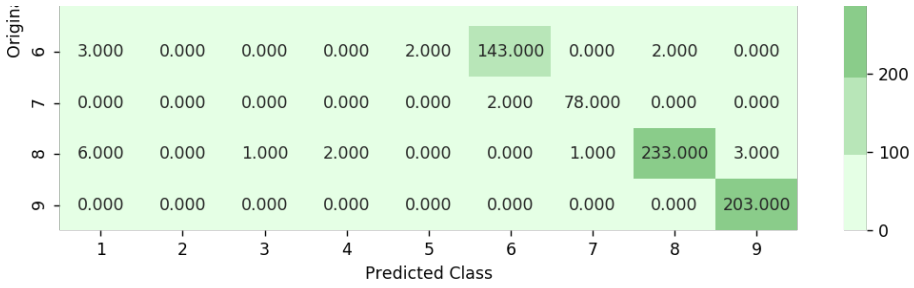
```
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y,
labels=k_cfl.classes_, eps=1e-15))

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))
```

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

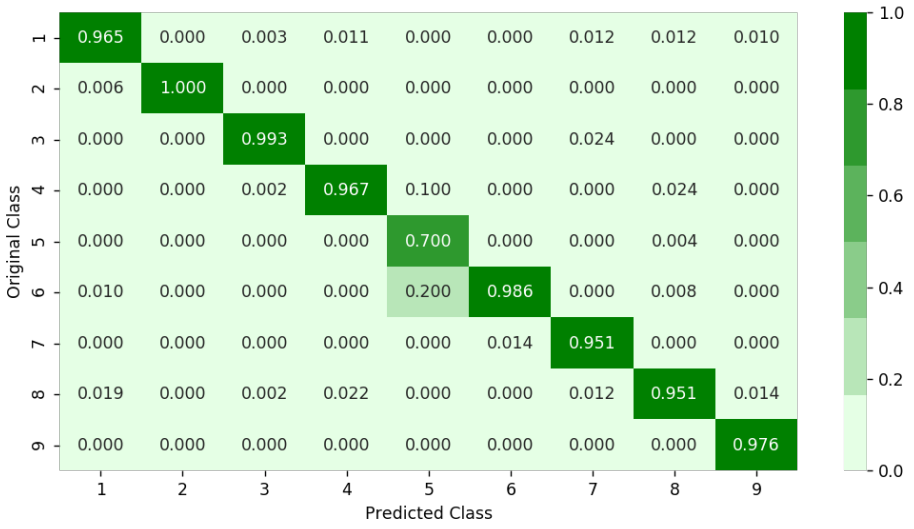



```
log loss for train data 0.0476773462198
log loss for cv data 0.0958800580948
log loss for test data 0.0894810720832
Number of misclassified points  2.02391904324
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```

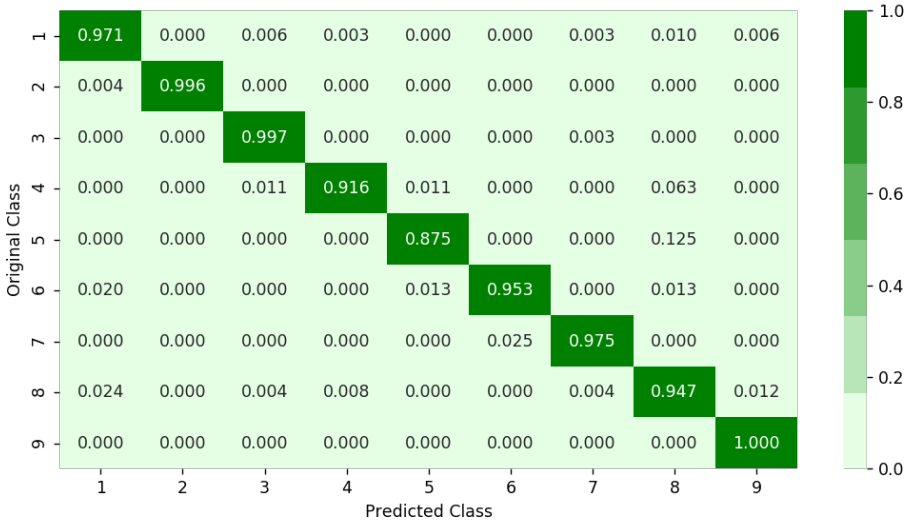





```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```




```
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.]
```

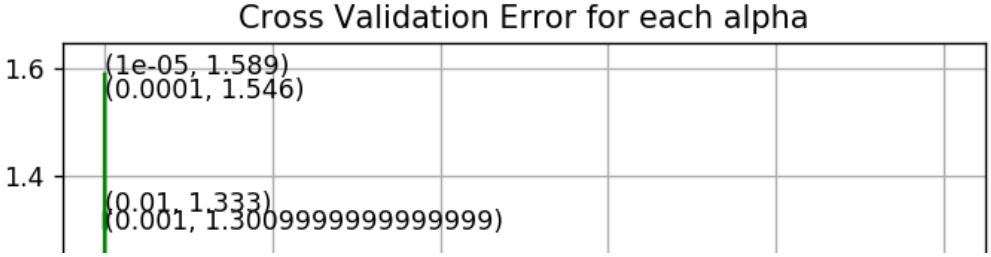
4.4.2 Logistic Regression

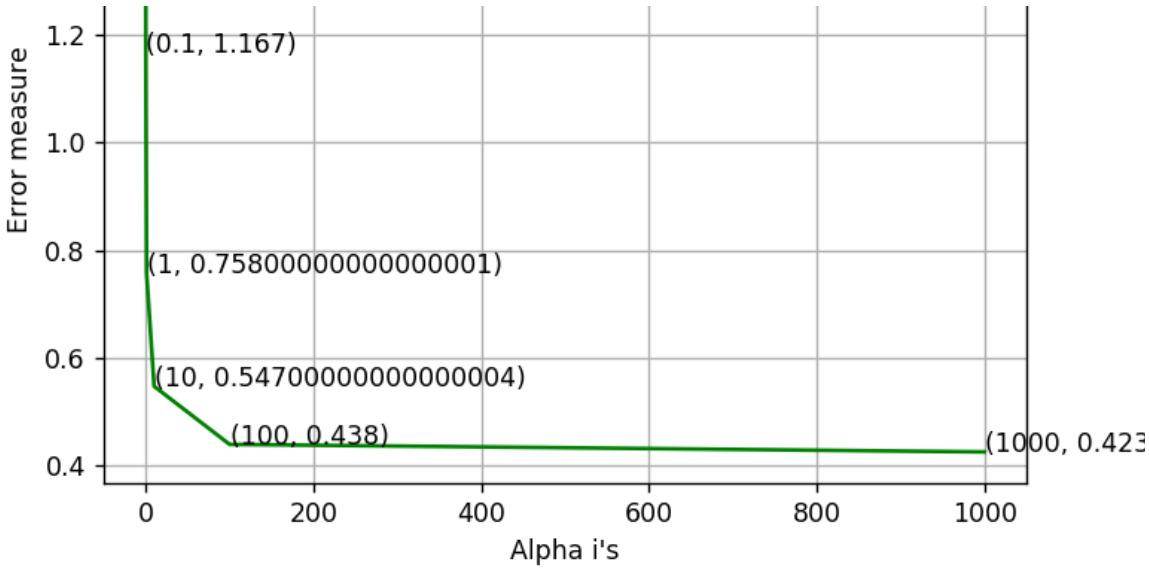
In [ ]:

```
# 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)
```

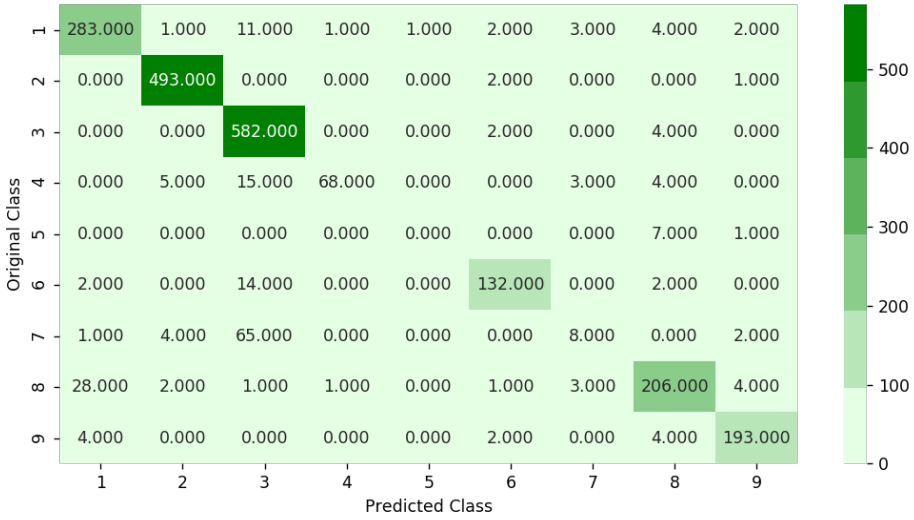
```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

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

```
log_loss for c =  1e-05 is 1.58867274165
log_loss for c =  0.0001 is 1.54560797884
log_loss for c =  0.001 is 1.30137786807
log_loss for c =  0.01 is 1.33317456931
log_loss for c =  0.1 is 1.16705751378
log_loss for c =  1 is 0.757667807779
log_loss for c =  10 is 0.546533939819
log_loss for c =  100 is 0.438414998062
log_loss for c =  1000 is 0.424423536526
```

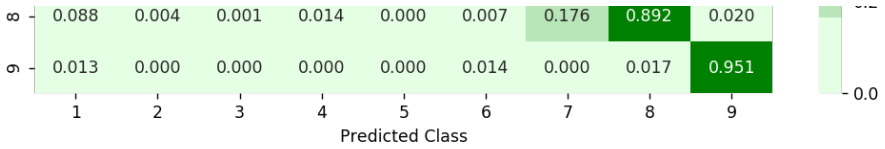




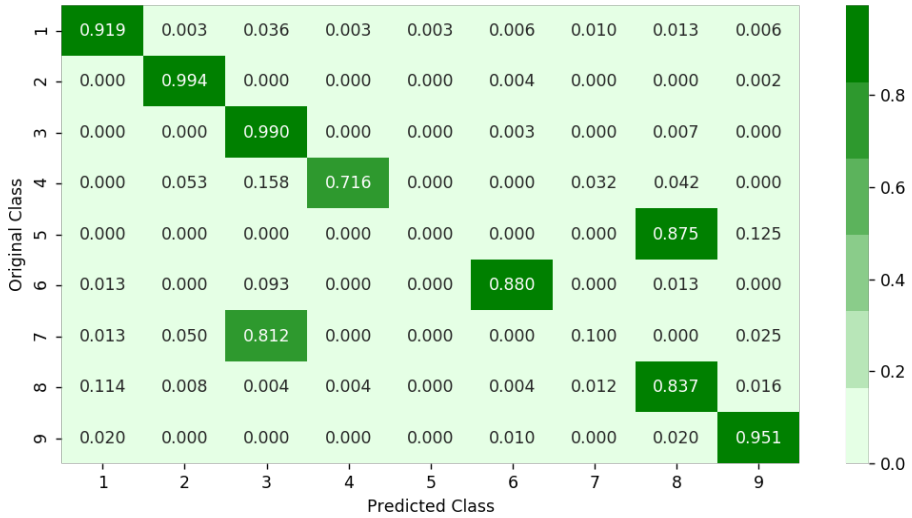

```
log loss for train data 0.396219394701
log loss for cv data 0.424423536526
log loss for test data 0.415685592517
Number of misclassified points  9.61361545538
---------------------------------------------------- Confusion matrix
----------------------------------------------------
```




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




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




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

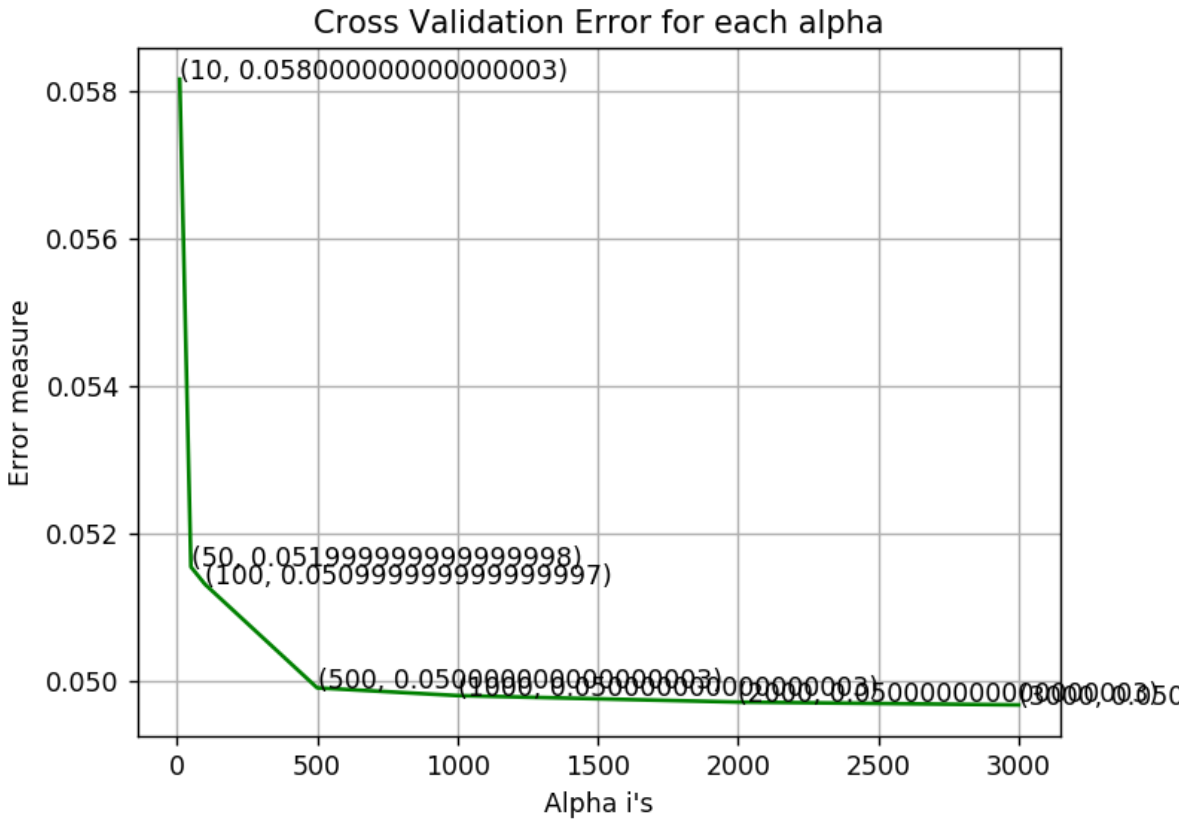
4.4.3 Random Forest Classifier

In [ ]:

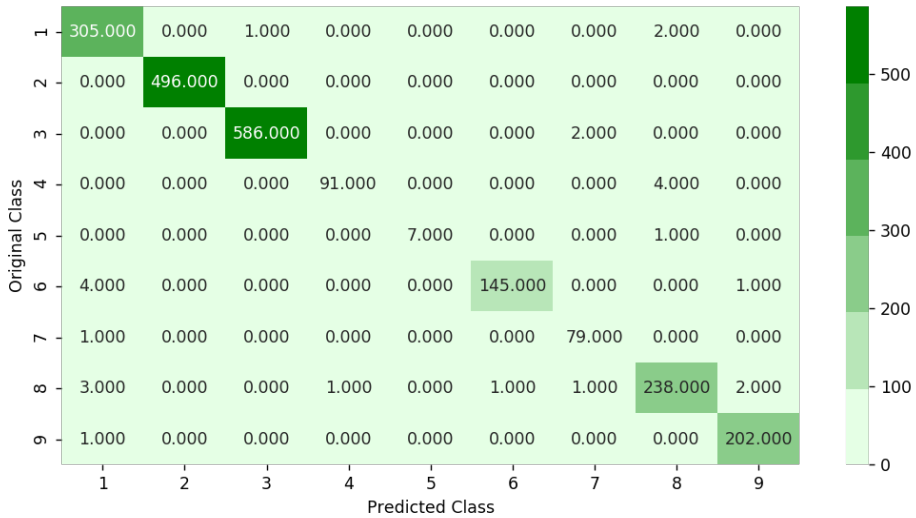
```
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10,
criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y,
labels=r_cfl.classes_, eps=1e-15))
```

```
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_st
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))
```

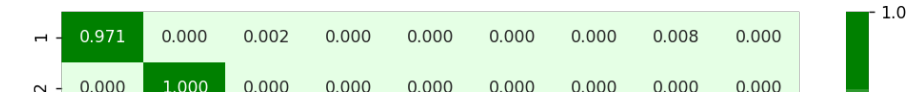
```
log_loss for c =  10 is 0.0581657906023
log_loss for c =  50 is 0.0515443148419
log_loss for c =  100 is 0.0513084973231
log_loss for c =  500 is 0.0499021761479
log_loss for c =  1000 is 0.0497972474298
log_loss for c =  2000 is 0.0497091690815
log_loss for c =  3000 is 0.0496706817633
```

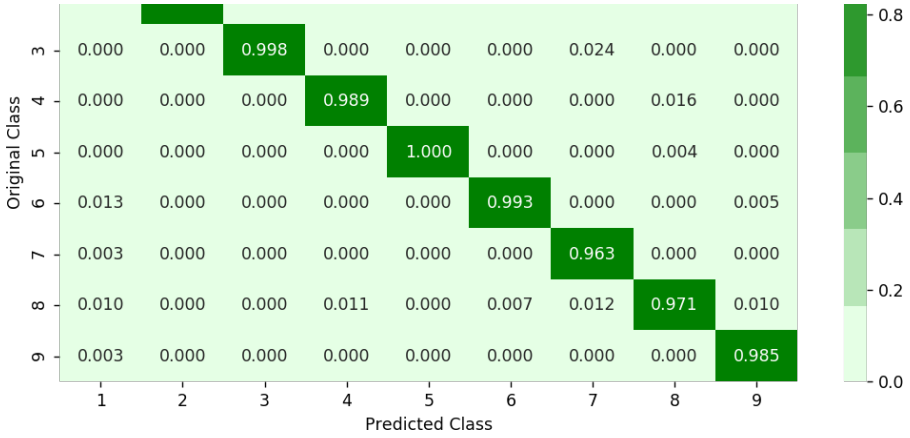



```
log loss for train data 0.0116517052676
log loss for cv data 0.0496706817633
log loss for test data 0.0571239496453
Number of misclassified points  1.14995400184
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```

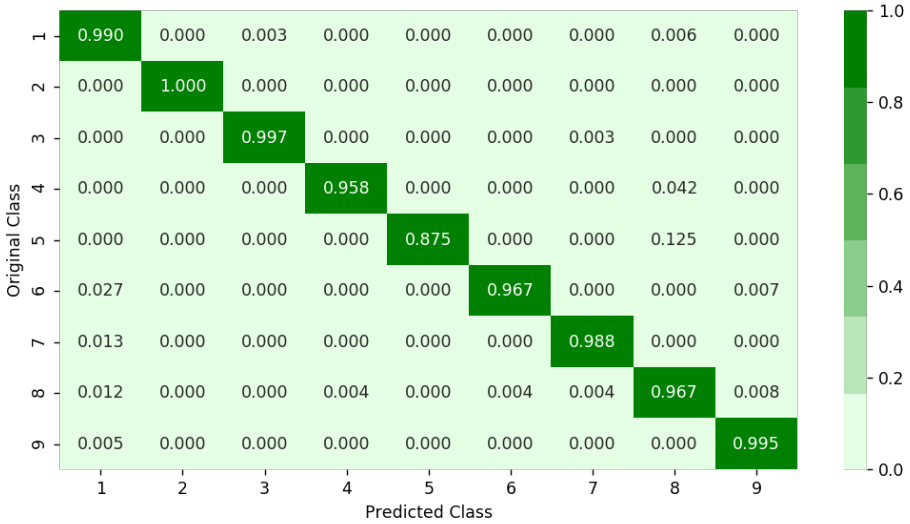



```
---------------------------------------------------- Precision matrix
-----------------------------------------------------
```






```
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

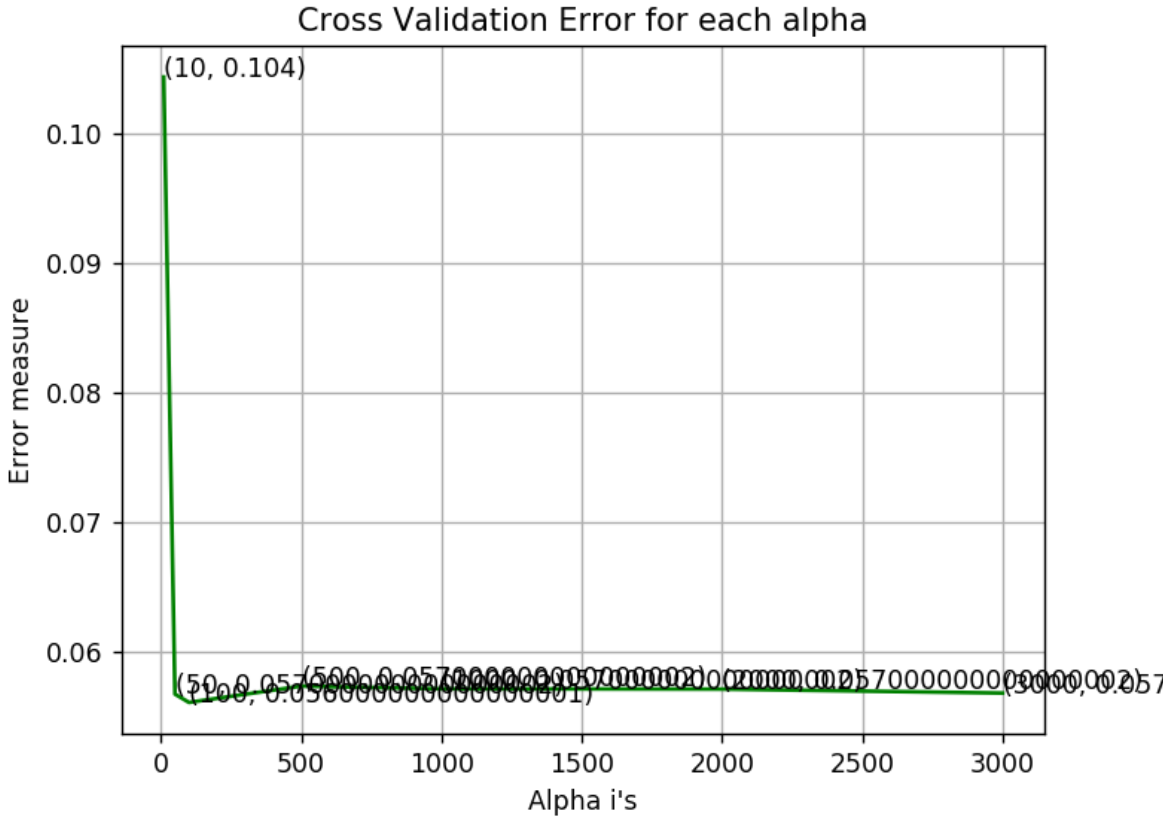
In [ ]:

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

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

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,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,
```

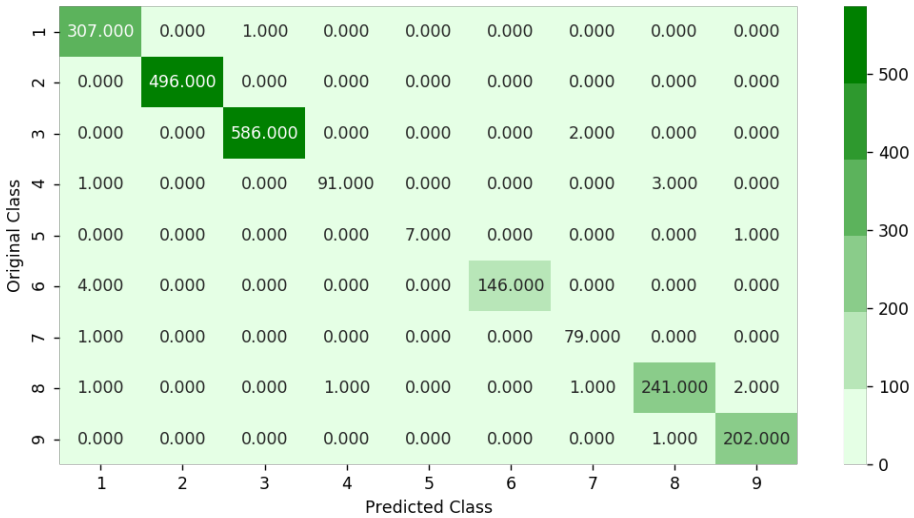
```
best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

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

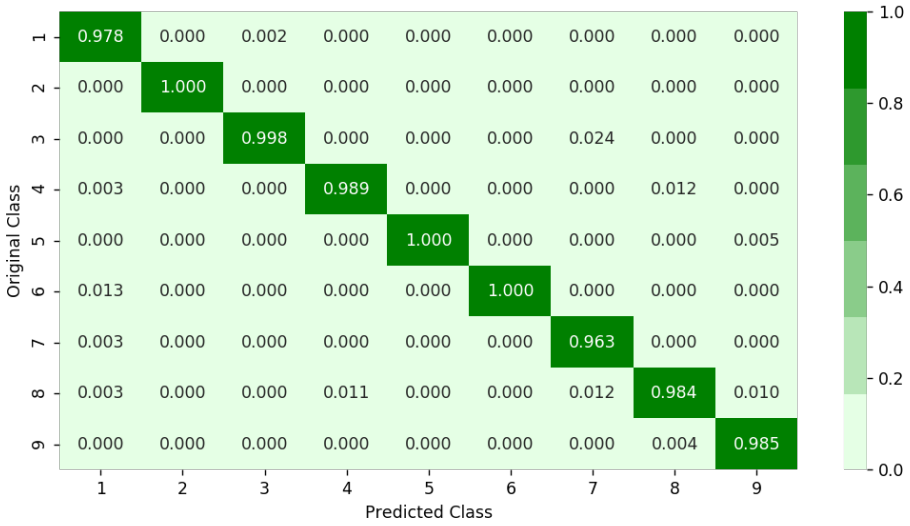
```
log_loss for c =  2000 is 0.057103406781
log_loss for c =  3000 is 0.0567993215778
```



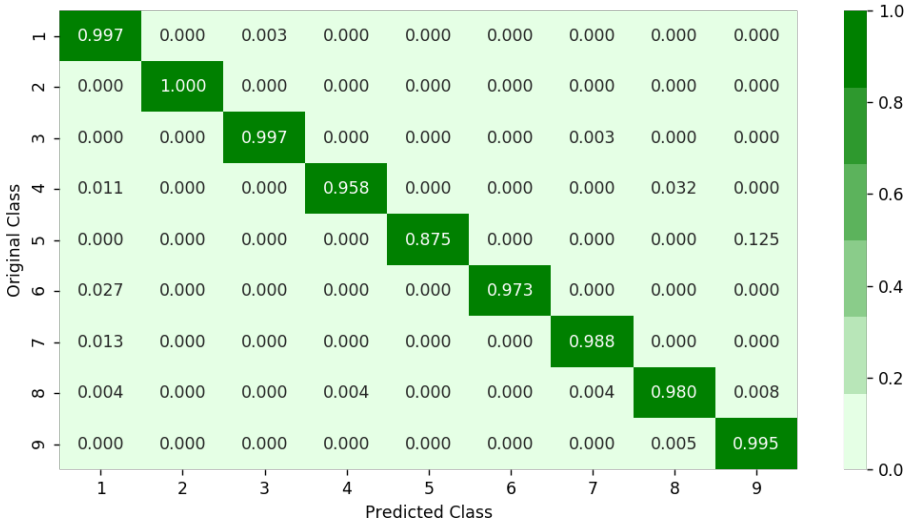

```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.0560750386
46
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```




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




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




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

4.4.5 Xgboost Classifier with best hyperparameters

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


prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[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
1,)
random_cfl.fit(X_train_asm,y_train_asm)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  1.1min remaining:   39.
3s
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  1.3min remaining:   23.
0s
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  1.4min remaining:    9.
2s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.3min finished
```

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

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

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

In [ ]:

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

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

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

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.1
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))
```

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

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

4.5.1. Merging both asm and byte file features

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

Out[ ]:

	ID	0	1	2	3	4	5	6	
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.00
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.00
2	01jsnpXSAlgw6aPeDxrU	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.00
3	01kcPWA9K2BOxQeS5Rju	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.00
4	01SuzwMJEIXsK7A8dQbl	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.00

5 rows × 260 columns

In [ ]:
```
result_asm.head()
```

Out[ ]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.0
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.0
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.0
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.0
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.0

5 rows × 54 columns

In [ ]:
```
print(result.shape)
print(result_asm.shape)
```

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

In [ ]:
```
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()
```

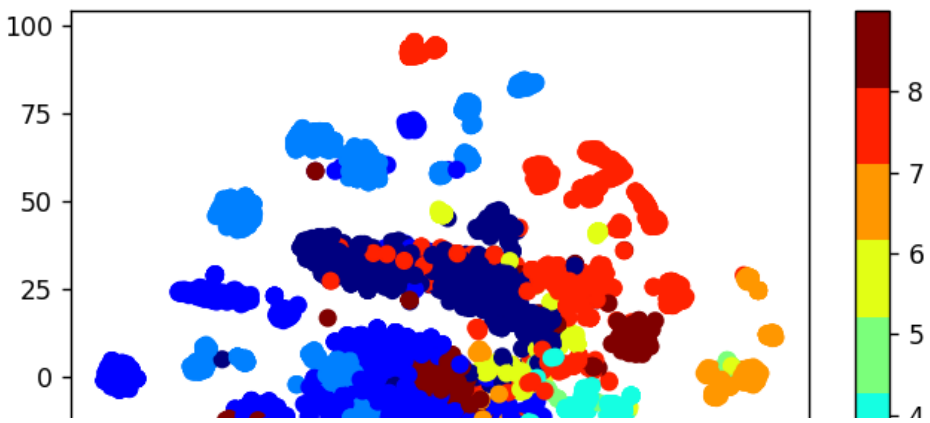
Out[ ]:

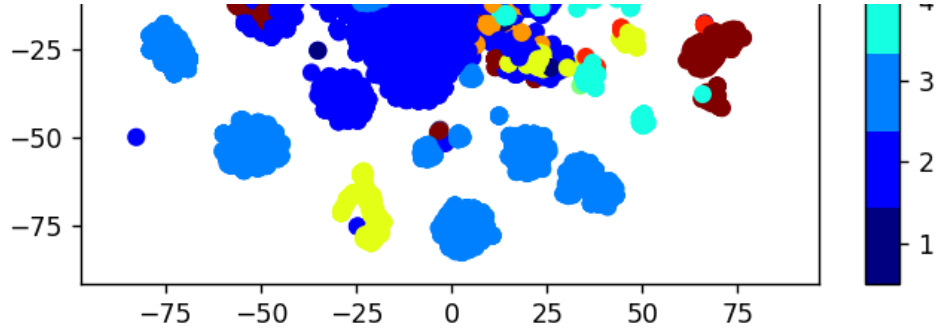
	0	1	2	3	4	5	6	7	8	9
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638	0.003531
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267	0.000394
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104	0.002707
3	0.009209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

5 rows × 307 columns

4.5.2. Multivariate Analysis on final fearures

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





4.5.3. Train and Test split

In [ ]:

```
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

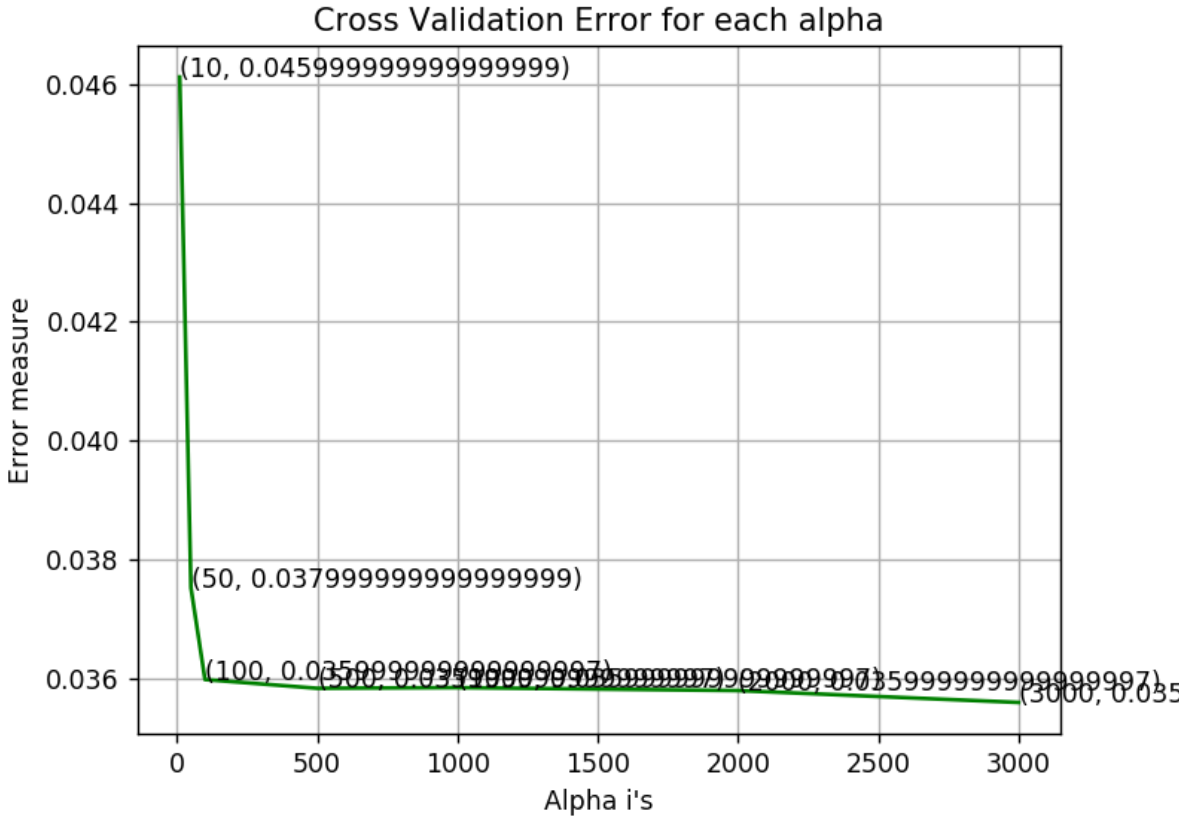
In [ ]:

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

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

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

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

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

r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1
    r_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y,
```

```
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_st
1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

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

```
log_loss for c =  2000 is 0.0357908022178
log_loss for c =  3000 is 0.0355909487962
```




```
For values of best alpha =  3000 The train log loss is: 0.0166267614753
For values of best alpha =  3000 The cross validation log loss is: 0.035590948
7962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

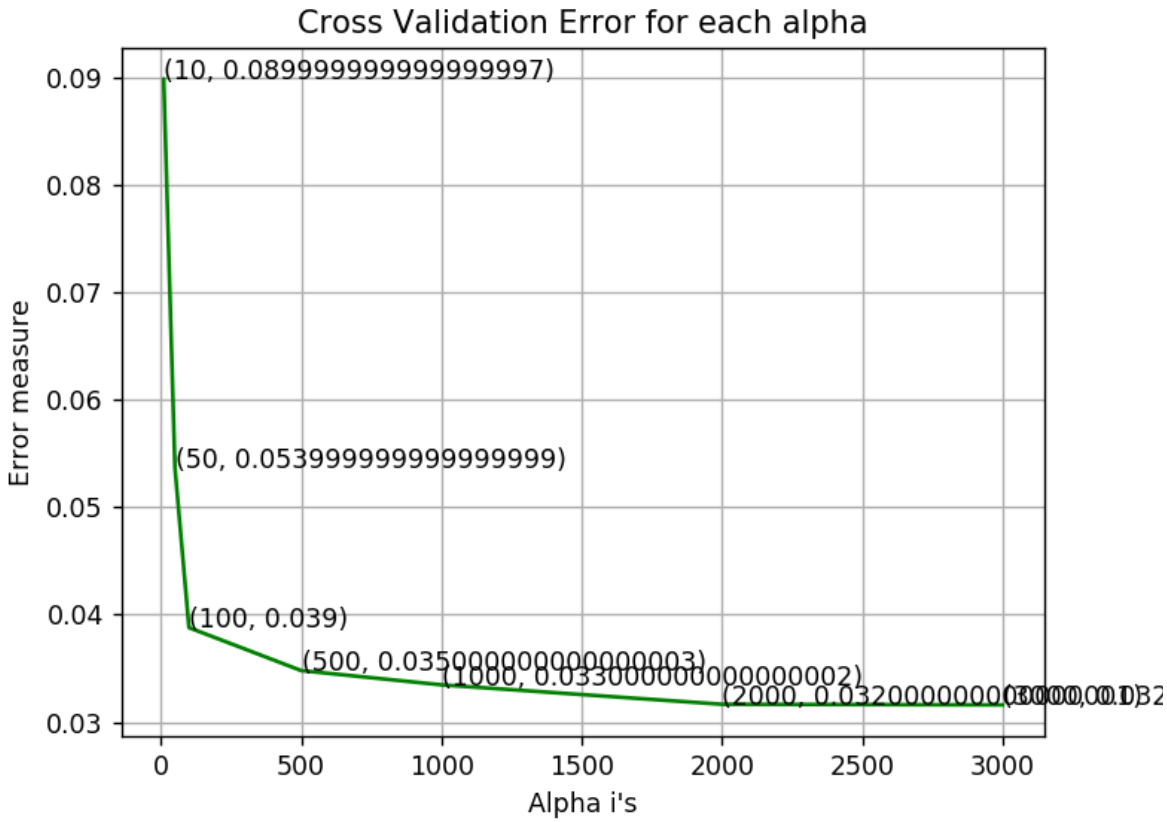
4.5.5. XgBoost Classifier on final features

In [ ]:

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

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

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

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,
```

```
best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  10 is 0.0898979446265
log_loss for c =  50 is 0.0536946658041
log_loss for c =  100 is 0.0387968186177
log_loss for c =  500 is 0.0347960327293
log_loss for c =  1000 is 0.0334668083237
log_loss for c =  2000 is 0.0316569078846
log_loss for c =  3000 is 0.0315972694477
```




```
For values of best alpha =  3000 The train log loss is: 0.0111918809342
For values of best alpha =  3000 The cross validation log loss is: 0.031597269
4477
For values of best alpha =  3000 The test log loss is: 0.0323978515915
```

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

In [ ]:

```
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
1,)
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  4.5min remaining:  2.6m
```

```
in
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remaining:  1.8m
in
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remaining:   44.
5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

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

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

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

In [ ]:

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

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

x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15
1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

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

```
For values of best alpha =  3000 The train log loss is: 0.0121922832297
For values of best alpha =  3000 The cross validation log loss is: 0.034495548
7471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

5. Assignments

1. Add bi-grams on byte files and improve the log-loss
2. Watch the video (video) and include pixel intensity features to improve the logloss

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

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

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

https://askubuntu.com/a/341637
```

In [5]:

```
# importing libraries
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
import scipy
from scipy.sparse import csr_matrix


#https://joblib.readthedocs.io/en/latest/
#https://joblib.readthedocs.io/en/latest/generated
/joblib.Parallel.html
from joblib import Parallel, delayed
import joblib
from datetime import datetime
```

In [6]:
```
#https://github.com/saicharanarishanapally/microsoft-malware-
detection/blob/master/MicrosoftMalwareDetection.ipynb
byte_index =
"ID,00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,1
byte_vocab =
"00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,1
```

In [7]:
```
# creating bigrams index. and vocab.
def byte_bigram(byte):
    byte_bigram_vocab = []
    if(byte is byte_index):
        for i, v in enumerate(byte.split(',')):
            if(v != 'ID'):
                for j in range(1, len(byte.split(','))):
                    byte_bigram_vocab.append(v + ' '
+byte.split(',')[j])
            else:
                byte_bigram_vocab.append(v)
        print("Length of index.: ",len(byte_bigram_vocab))
        return byte_bigram_vocab

    elif(byte is byte_vocab):
        for i, v in enumerate(byte.split(',')):
            for j in range(0, len(byte.split(','))):
                byte_bigram_vocab.append(v + ' '
+byte.split(',')[j])
        print("Length of vocab.: ",len(byte_bigram_vocab))
        return byte_bigram_vocab



byte_bigram_index = []
byte_bigram_vocab = []

byte_bigram_index = byte_bigram(byte_index)
byte_bigram_vocab = byte_bigram(byte_vocab)
```

```
Length of index.:  66050
Length of vocab.:  66049
```

In [5]:
```
# first five vocab.
byte_bigram_vocab[:5]
```

Out[5]:
```
['00 00', '00 01', '00 02', '00 03', '00 04']
```

In [21]:
```
# created a method, for bigrams vectorization
vector= CountVectorizer(lowercase=False,ngram_range=(2,2),
                        vocabulary=byte_bigram_vocab)


def bgrams(file, dir_name):
    """
    a method to create bigrams for the file and write it into a csv
file
    """
    global vector
    with open(dir_name + '/'+file,"r") as byte_flie:
        vec =
(vector.fit_transform([byte_flie.read().replace('\n','
').lower()]))
        df = pd.DataFrame(vec.todense())
        df['ID'] = file
        df.set_index('ID', inplace = True)
        df.to_csv('result_bgrams.csv', mode='a', header=False)
    byte_flie.close()
```

In [ ]:
```
byte_bigram_inx = ','.join([str(elem) for elem in
byte_bigram_index]) # converting list to string
# creating a csv file, putting the column names
byte_feature_file=open('result_bgrams.csv','w+')
byte_feature_file.write(byte_bigram_inx)
byte_feature_file.write("\n")
byte_feature_file.close()

dir_name = 'byteFiles'
files = os.listdir(dir_name)

# applying the joblib.Parallel, delayed to operate it in multi-
processing env.
Parallel(backend= 'loky', n_jobs=-1)(
    delayed(bgrams)(file, dir_name) for file in files)
byte_feature_file.close()
```

In [ ]:
```
import pandas as pd

df= pd.read_csv("result_bgrams.csv", iterator= True, chunksize=
5000,
                                low_memory= False)

byte_features = pd.concat(df, ignore_index=True)
```

In [10]:
```
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)
```

Out[10]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	...	?? f7	?? f8	?? f9	?? fa	?? fb	?? fc
0	01SuzwMJEIXsK7A8dQbl	16277	62	22	126	9	11	3	5	11	...	0.0	0.0	0.0	0.0	0.0	0.0
1	01kcPWA9K2BOxQeS5Rju	10423	225	61	69	114	40	25	22	63	...	0.0	0.0	0.0	0.0	0.0	0.0

2 rows × 66050 columns

In [14]:
```
byte_features_with_size = byte_features.merge(data_size_byte,
on='ID')
byte_features_with_size.head(2)
```

Out[14]:

	ID	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	...	?? f9	?? fa	?? fb	?? fc	?? fd	?? fe
0	01SuzwMJEIXsK7A8dQbl	16277	62	22	126	9	11	3	5	11	...	0.0	0.0	0.0	0.0	0.0	0.0
1	01kcPWA9K2BOxQeS5Rju	10423	225	61	69	114	40	25	22	63	...	0.0	0.0	0.0	0.0	0.0	0.0

2 rows × 66052 columns

In [16]:

```
data_y = byte_features_with_size['Class']
data_X = byte_features_with_size.drop(['ID','Class'], axis=1)


data_X.head(2)
```

Out[16]:

	00 00	00 01	00 02	00 03	00 04	00 05	00 06	00 07	00 08	00 09	...	?? f8	?? f9	?? fa	?? fb	?? fc	?? fd	?? fe	?? ff	?? ??	
0	16277	62	22	126	9	11	3	5	11	3	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.370
1	10423	225	61	69	114	40	25	22	63	15	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.574

2 rows × 66050 columns

In [3]:

```
from sklearn.preprocessing import MinMaxScaler


normalizer  = MinMaxScaler()


data_X = normalizer.fit_transform(data_X)
```

In [25]:
```
# feature selection
# https://github.com/saicharanarishanapally/microsoft-malware-detection/blob/master/MicrosoftMalwareDetection.ipynb

def imp_features(data, features, keep):
    rf = RandomForestClassifier(n_jobs = -1)
    rf.fit(data, data_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 [26]:
```
# getting column number of top 500 features
byte_index = imp_features(data_X, byte_bigram_vocab, 500)
```

In [ ]:
```
#https://stackoverflow.com/questions/45846189/how-to-delete-all-columns-in-dataframe-except-certain-ones/53214704
data_X = data_X.loc[:, data_X.columns.intersection(byte_index)]
```

In [34]:
```
# 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(data_X,
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)
```

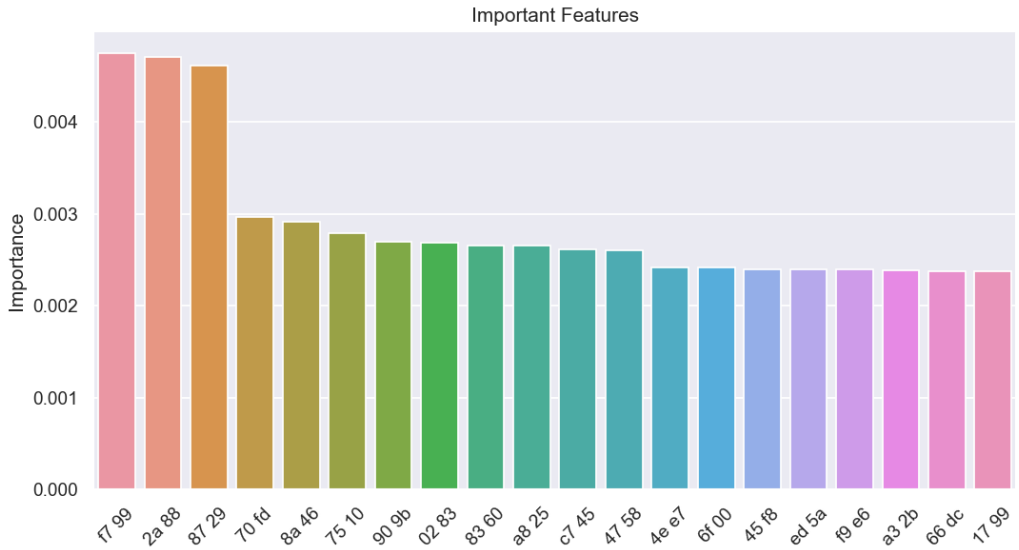
Applying K-NN Classifier

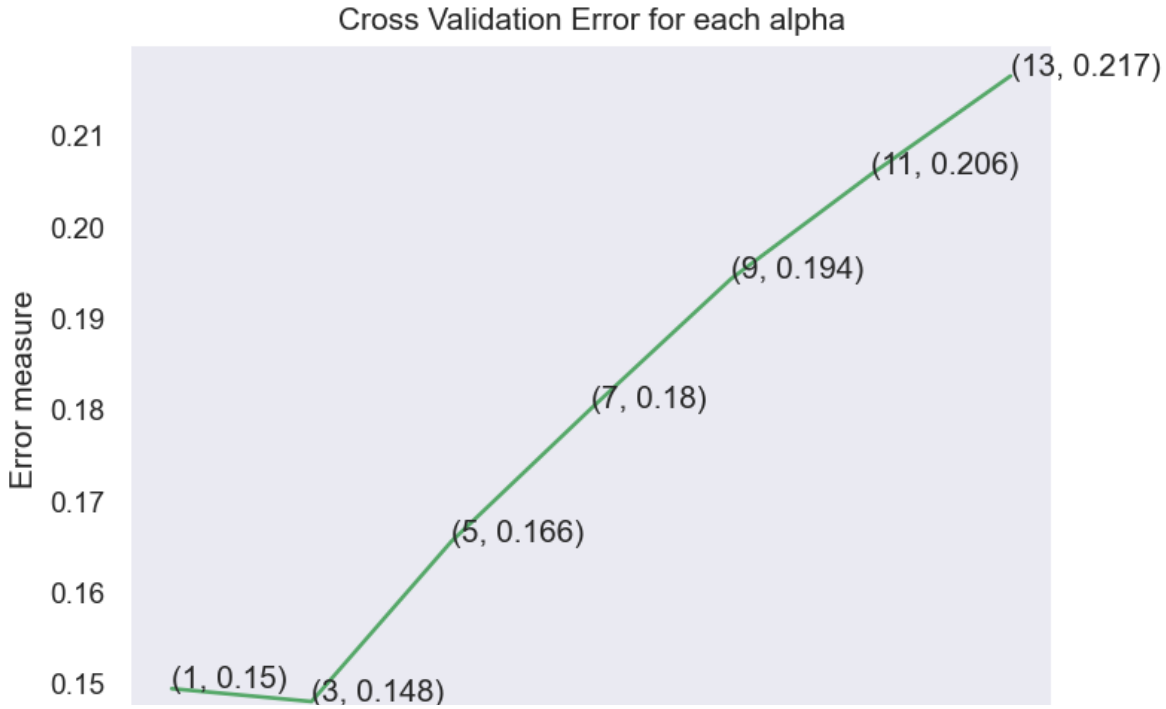
In [39]:

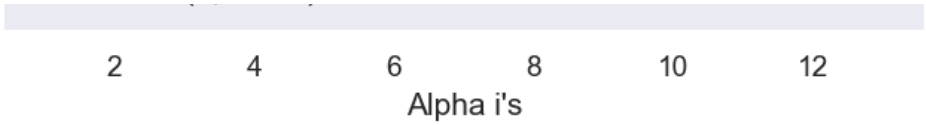
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated
/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform',
algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

# methods of
# fit(X, y) : Fit the model using X as training data and y as
target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data
X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-
course-online/lessons/k-nearest-neighbors-geometric-intuition-with-
a-toy-example-1/
#--------------------------------------


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

```
    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], n_jobs=-
1)
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
```

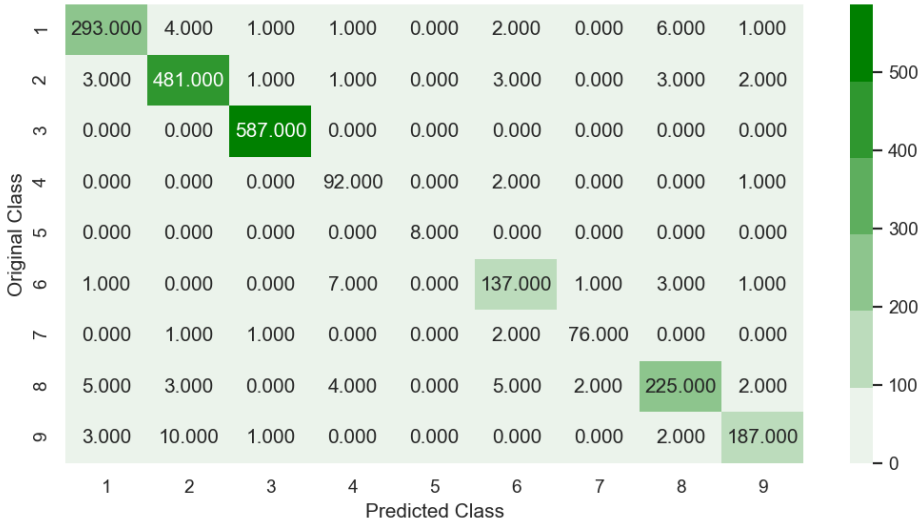
```
log_loss for k =  1 is 0.14953128040077876
log_loss for k =  3 is 0.14812005722826033
log_loss for k =  5 is 0.16562228511459154
log_loss for k =  7 is 0.18016959686346076
log_loss for k =  9 is 0.19433635843280128
log_loss for k =  11 is 0.20574753509871116
log_loss for k =  13 is 0.2165345962293062
```



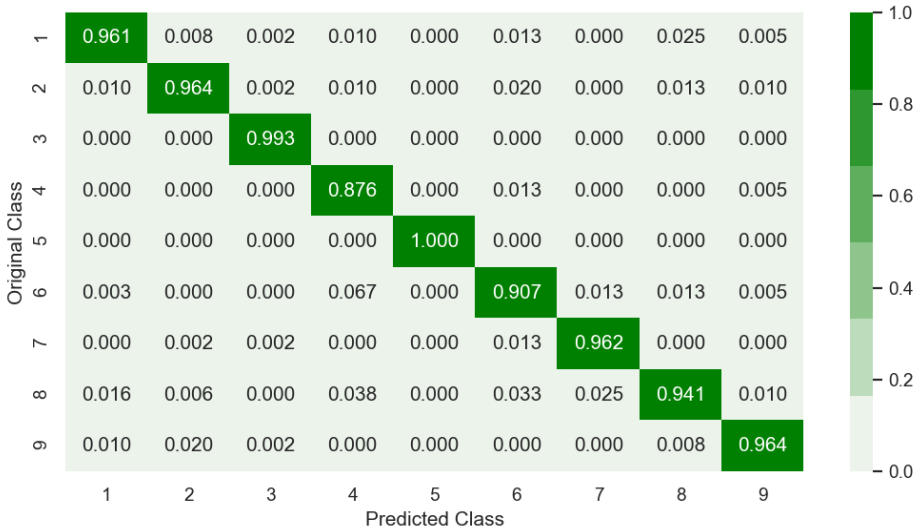




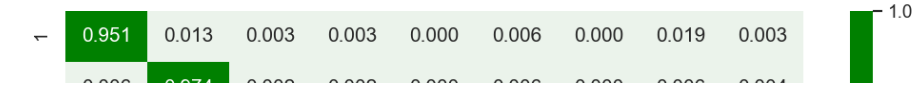

```
For values of best alpha =  3 The train log loss is: 0.09544810285645419
For values of best alpha =  3 The cross validation log loss is: 0.148120057228
26033
For values of best alpha =  3 The test log loss is: 0.16672633780355153
Number of misclassified points  3.91524643021649
---------------------------------------------------- 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.]
```

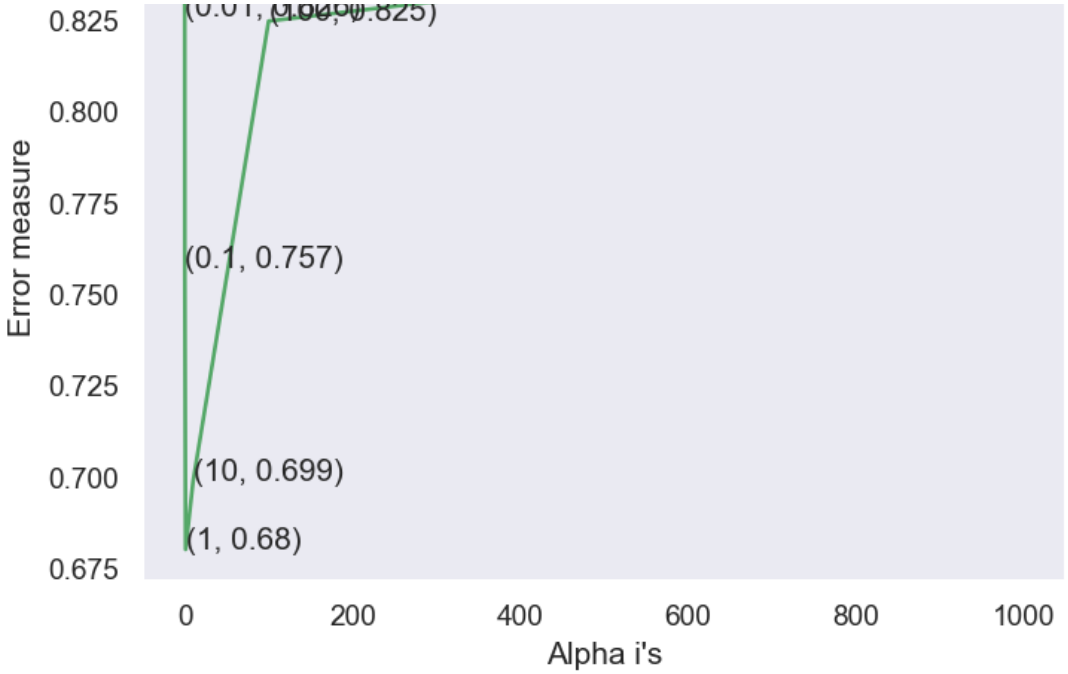
Logistic Regression

In [40]:

```
# 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'
n_jobs=-1)
    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)
```

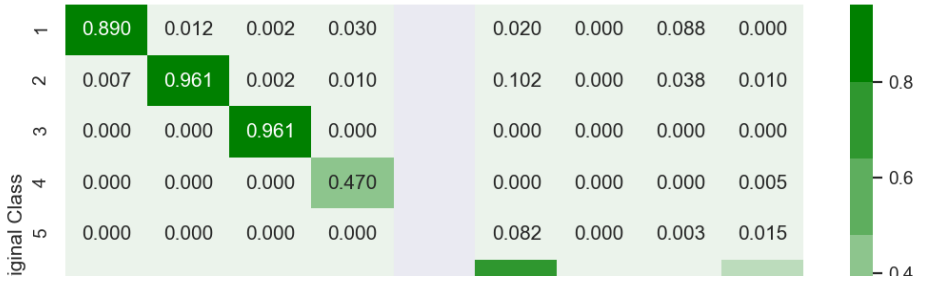
```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_w
n_jobs=-1)
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

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

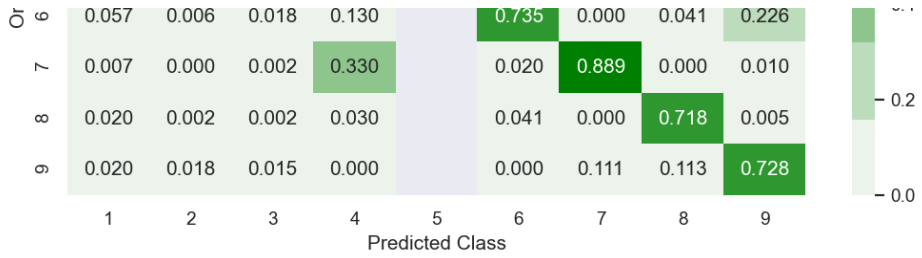
```
log_loss for c =  1e-05 is 0.8514329633619913
log_loss for c =  0.0001 is 0.8513804038169095
log_loss for c =  0.001 is 0.8481395298431571
log_loss for c =  0.01 is 0.8262185452443498
log_loss for c =  0.1 is 0.757151988411306
log_loss for c =  1 is 0.680313823162703
log_loss for c =  10 is 0.6990047491422992
log_loss for c =  100 is 0.8248871931950107
log_loss for c =  1000 is 0.8492543705090668
```



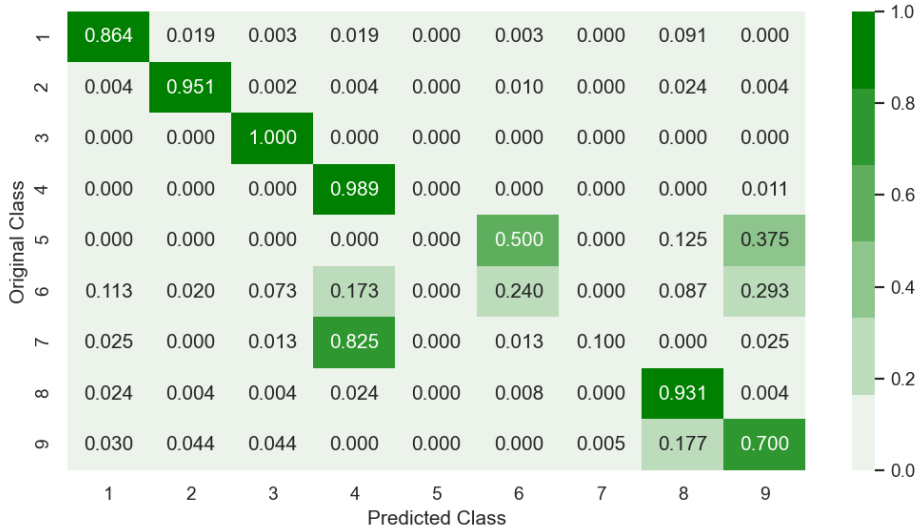

```
log loss for train data 0.6738217138085941
log loss for cv data 0.680313823162703
log loss for test data 0.6944311831932798
Number of misclassified points  15.614923998157531
---------------------------------------------------- Confusion matrix
----------------------------------------------------
```


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

Random Forest Classifier

In [41]:

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

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

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

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

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

r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
```

```
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_st
1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
log_loss for c =  10 is 0.05079594604435841
```

```
log_loss for c =  50 is 0.04549052963904461
log_loss for c =  100 is 0.0439454763215921
log_loss for c =  500 is 0.04183499589683186
log_loss for c =  1000 is 0.04163320606428248
log_loss for c =  2000 is 0.041627122229237576


For values of best alpha =  3000 The train log loss is: 0.017278256862096848
For values of best alpha =  3000 The cross validation log loss is: 0.041493815
352179206
For values of best alpha =  3000 The test log loss is: 0.04986752753503522
Number of misclassified points  1.1515430677107323
---------------------------------------------------- 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.]
```

Xg-Boost

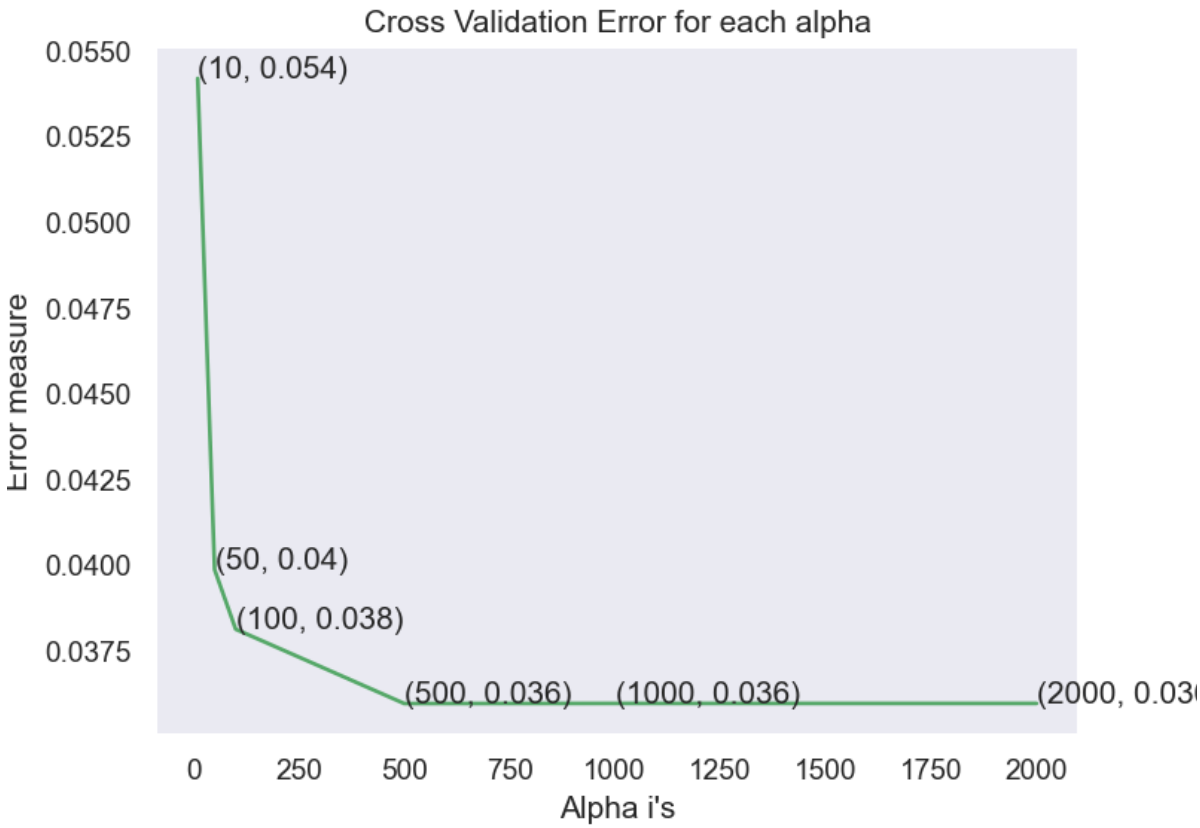
In [50]:

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

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

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None,
early_stopping_rounds=None, verbose=True, xgb_model=None)
# get_params([deep])    Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with
data. NOTE: This function is not thread safe.
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-
course-online/lessons/regression-using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-
course-online/lessons/what-are-ensembles/
# ------------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1,
eval_metric='mlogloss')
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
```

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for c =  10 is 0.05421503002142859
```

```
log_loss for c =  50 is 0.03987843437162498
log_loss for c =  100 is 0.0381389705758782
log_loss for c =  500 is 0.03596881533921364
log_loss for c =  1000 is 0.035972220411218135
```

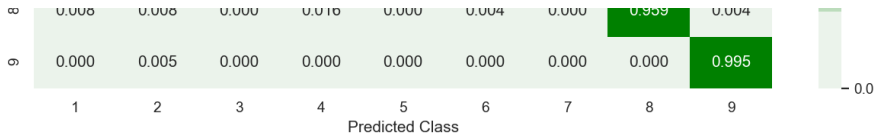



```
[12:29:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'multi:softprob' was changed from 'merror' to 'mlo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[12:31:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'multi:softprob' was changed from 'merror' to 'mlo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[12:32:00] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'multi:softprob' was changed from 'merror' to 'mlo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[12:32:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'multi:softprob' was changed from 'merror' to 'mlo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[12:33:39] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'multi:softprob' was changed from 'merror' to 'mlo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
[12:34:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation met
ric used with the objective 'multi:softprob' was changed from 'merror' to 'mlo
gloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
For values of best alpha =  500 The train log loss is: 0.016703034558395 69
For values of best alpha =  500 The cross validation log loss is: 0.0359688153
3921364
For values of best alpha =  500 The test log loss is: 0.04822400623462876
Number of misclassified points  0.8751727314601566
```

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

XgBoost Classification with best hyper parameters using RandomSearch

In [51]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-
parameter-tuning-xgboost-with-codes-python/
x_cfl=XGBClassifier(nthread=-1, eval_metric='mlogloss')

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,verbos
1,)
random_cfl1.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  9.4min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 24.3min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 40.9min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 49.8min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 53.6min remaining: 11.8m
in
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 59.4min remaining:  3.8m
in
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 61.0min finished
```

Out[51]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None,
                                           eval_metric='mlogloss', gamma=None,
                                           gpu_id=None, importance_type='gain
',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=Non
e,
                                           min_child_weight=None, missing=nan,
                                           monotone_constraints=None...
                                           random_state=None, reg_alpha=None,
                                           reg_lambda=None,
                                           scale_pos_weight=None,
```

```
                                              subsample=None, tree_method=None,
                                              validate_parameters=None,
                                              verbosity=None),
                   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]},
                   verbose=10)
```

In [52]:
```
random_cfl1.best_estimator_
```

Out[52]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='mlogloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=3, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=1000, n_jobs=8,
              nthread=-1, num_parallel_tree=1, objective='multi:softprob',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=Non
e,
              subsample=0.5, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [53]:

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

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

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

x_cfl=XGBClassifier(n_estimators=1000, learning_rate=0.01,
colsample_bytree=1, max_depth=3, eval_metric='mlogloss')
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
```

```
train loss 0.01693398420867603
cv loss 0.042667181817350665
test loss 0.050232145254239594
```

Task 2: implementing pixel intensive features

In [10]:

```
import array
import codecs
import imageio
import cv2
```

In [63]:

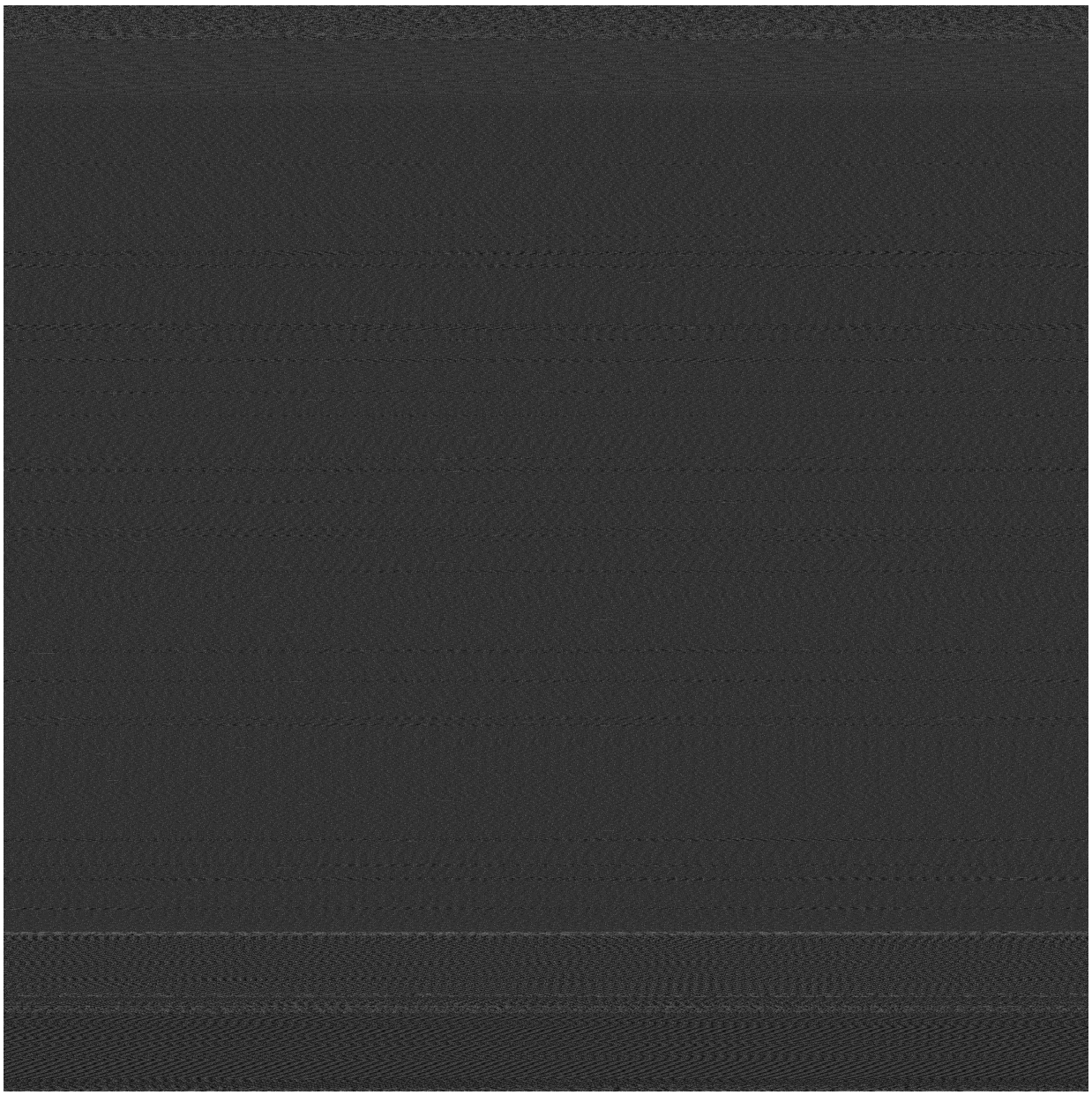
```
# https://raw.githubusercontent.com/saicharanarishanapally
/microsoft-malware-detection/master/MicrosoftMalwareDetection.pdf
def collect_img_asm(asmfile):
    filename = asmfile.split('.')[0]
    file = codecs.open('asmFiles/'+ asmfile, 'rb') # to open asm/
byte files
    file_len = os.path.getsize('asmFiles/'+ asmfile)
    width = int(file_len ** 0.5)
    rem = int(file_len / width)
    arr = array.array('B')
    arr.frombytes(file.read())
    file.close()
    reshaped = np.reshape(arr[:width * width], (width, width))
    reshaped = np.uint8(reshaped)
    imageio.imwrite('asm_img/'+ filename + '.png', reshaped)
```

In [ ]:

```
dir_name = 'asmFiles'
files = os.listdir(dir_name)

# applying the joblib.Parallel, delayed to operate it in multi-
processing env.
Parallel(backend= 'loky', n_jobs=-1, verbose=1)(
    delayed(collect_img_asm)(asmfile) for asmfile in files)
```

In [75]:
```
from IPython.display import Image
Image(filename= 'asm_img/01IsoiSMh5gxyDYTl4CB.png')
```

Out[75]:



In [86]:
```
img = cv2.imread('asm_img/01azqd4InC7m9JpocGv5.png')
img_arr = img.flatten()
img_arr.shape
```

Out[86]: (176855052,)

In [11]:
```
# First 500 pixels of images
# https://raw.githubusercontent.com/saicharanarishanapally
/microsoft-malware-detection/master/MicrosoftMalwareDetection.pdf
dir_name = 'asm_img'
files = os.listdir(dir_name)
n_features = 800

imagefeatures = np.zeros((len(files), n_features))
for i, asmfile in enumerate(files):
    img = cv2.imread("asm_img/"+ asmfile)
    img_arr = img.flatten()[:n_features]
    imagefeatures[i, :] += img_arr
```

In [12]:
```
imagefeatures
```

Out[12]:
```
array([[ 72.,  72.,  72., ..., 103., 104., 104.],
       [ 46.,  46.,  46., ...,  32.,  32.,  32.],
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       ...,
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       [ 72.,  72.,  72., ..., 103., 104., 104.],
       [ 72.,  72.,  72., ..., 103., 104., 104.]])
```

In [13]:
```
file_ID_df = pd.DataFrame(files, columns = ['ID'])
```

In [113…
```
from sklearn.preprocessing import normalize
imagefeatures = normalizer.fit_transform(imagefeatures)
```

In [173…
```
asm_img_df = pd.DataFrame(imagefeatures)
asm_img_df['ID'] = file_ID_df.ID.str.split('.').str[0]

asm_img_df.set_index('ID', inplace = True)
asm_img_df.head(2)
```

Out[173…

ID	0	1	2	3	4	5	6	
01azqd4InC7m9JpocGv5	0.556409	0.556409	0.556409	0.558238	0.558238	0.558238	0.238293	0.23829
01IsoiSMh5gxyDYTl4CB	0.194511	0.194511	0.194511	0.927035	0.927035	0.927035	0.658327	0.65832

2 rows × 800 columns

In [174…
```
asm_img_df.to_csv('asm_img_800.csv')
```

Combining features

In [295…
```
# asm image feature
asm_img_df = pd.read_csv('asm_img_800.csv')

col_names = ['ID']
for i in asm_img_df.columns:
    if(i != 'ID'):
        col_names.append('asm_img'+str(i))
asm_img_df.columns = col_names

asm_img_df.head(2)
```

Out[295…

	ID	asm_img0	asm_img1	asm_img2	asm_img3	asm_img4	asm_img5	asm_in
0	01azqd4InC7m9JpocGv5	0.556409	0.556409	0.556409	0.558238	0.558238	0.558238	0.238
1	01IsoiSMh5gxyDYTl4CB	0.194511	0.194511	0.194511	0.927035	0.927035	0.927035	0.658

2 rows × 801 columns

In [296…
```
# asm feature
from sklearn.preprocessing import MinMaxScaler

normalizer = MinMaxScaler()

asm_df = pd.read_csv('asmoutputfile.csv')
col_name = asm_df.columns[1:]
ID = asm_df.ID

asm_df = pd.DataFrame(normalizer.fit_transform(asm_df.drop(columns
= ['ID'])))
asm_df.columns = col_name
asm_df['ID'] = ID
asm_df.set_index('ID', inplace = True)
asm_df.head()
```

Out[296…

ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.
01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.00
1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.00
3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.00
3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.00
46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.00

In [297...

```
asm_with_size = pd.read_csv('asm_with_size.csv')
asm_with_size = pd.merge(asm_df,asm_with_size.drop(columns =
['Unnamed: 0', 'Class']),on='ID', how= 'left')
asm_with_size.head()
```

Out[297...

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.0
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.0
2	3ekVow2ajZHbTnBcsDfX	0.096045	0.000627	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.0
3	3X2nY7iQaPBIWDrAZqJe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.0
4	46OZzdsSKDCFV8h7XWxf	0.096045	0.000590	0.0	0.000353	0.000068	0.0	0.000000	0.0	0.0

5 rows × 53 columns

In [298...

```
# merging all the asm dataframes
asm_final = pd.merge(asm_df, asm_img_df,on='ID', how = 'left')
asm_final.head(2)
```

Out[298...

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	
0	01kcPWA9K2BOxQeS5Rju	0.107345	0.001092	0.0	0.000761	0.000023	0.0	0.000084	0.0	0.0
1	1E93CpP60RHFNiT5Qfvn	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.0

2 rows × 852 columns

In [299…

```
# https://medium.com/nerd-for-tech/removing-constant-variables-feature-selection-463e2d6a30d9
from sklearn.feature_selection import VarianceThreshold

var_thr = VarianceThreshold(threshold = 0.001) #Removing both constant and quasi-constant
var_thr.fit(asm_final.drop('ID', axis = 1))

col = asm_final.columns[1:]
concol = [column for column in col if column not in col[var_thr.get_support()]]

print(len(concol))
```

```
39
```

In [300…

```
asm_final.drop(concol, axis = 1, inplace = True)
```

In [287…

```
# byte bigrams featured dataframe
byte_bigram_df = pd.read_csv('byte_bigrams_2000.csv')

col_names = ['ID']
for i in byte_bigram_df.columns:
    if(i != 'ID'):
        col_names.append('byte'+str(i))
byte_bigram_df.columns = col_names

byte_bigram_df.head(2)
```

Out[287…

	ID	byte0	byte1	byte2	byte3	byte4	byte5	byte6	b
0	01SuzwMJEIXsK7A8dQbl	0.007119	0.004335	0.001314	0.008352	0.000619	0.001098	0.000305	0.00
1	01kcPWA9K2BOxQeS5Rju	0.004559	0.015733	0.003643	0.004574	0.007844	0.003992	0.002538	0.00

2 rows × 2001 columns

In [288…

```
unigram_bytes = pd.read_csv('result_with_size.csv', index_col= 0)
unigram_bytes['ID'] = unigram_bytes.ID.str.split('.').str[0]
ID = unigram_bytes['ID']
col_names = list()
for i in unigram_bytes.columns[2:]:
      col_names.append('unigram'+str(i))

unigram_bytes =
pd.DataFrame(normalizer.fit_transform(unigram_bytes.drop(columns
=['ID', 'Class'])))
unigram_bytes.columns = col_names
unigram_bytes['ID'] = ID
unigram_bytes.set_index('ID', inplace =True)
unigram_bytes.head(2)
```

Out[288…

ID	unigram1	unigram2	unigram3	unigram4	unigram5	unigram6	unigram7
01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058
01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747

2 rows × 258 columns

In [289…

```
# merging all the bytes
bytes_df = pd.merge(unigram_bytes ,byte_bigram_df,on='ID')
bytes_df.head(2)
```

Out[289…

	ID	unigram1	unigram2	unigram3	unigram4	unigram5	unigram6	unigram
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.00205
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.00474

2 rows × 2259 columns

In [301…

```
# merging all the bytes and asm features
result_x = pd.merge(bytes_df, asm_final, on='ID', how = 'left')
result_x.head(2)
```

Out[301…

	ID	unigram1	unigram2	unigram3	unigram4	unigram5	unigram6	unigram
0	01azqd4InC7m9JpocGv5	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.00205

	ID	unigram1	unigram2	unigram3	unigram4	unigram5	unigram6	unigram
1	01IsoiSMh5gxyDYTl4CB	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.00474

In [302…

```
result_y = pd.merge(result_x,data_size_byte,on='ID')
result_y = result_y.Class
result_y.head(2)
```

Out[302…

```
0    9
1    2
Name: Class, dtype: int64
```

In [304…

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

Out[304…

	unigram1	unigram2	unigram3	unigram4	unigram5	unigram6	unigram7	unigram8	unigram9
0	0.262806	0.005498	0.001567	0.002067	0.002048	0.001835	0.002058	0.002946	0.002638
1	0.017358	0.011737	0.004033	0.003876	0.005303	0.003873	0.004747	0.006984	0.008267
2	0.040827	0.013434	0.001429	0.001315	0.005464	0.005280	0.005078	0.002155	0.008104
3	0.009209	0.001708	0.000404	0.000441	0.000770	0.000354	0.000310	0.000481	0.000959
4	0.008629	0.001000	0.000168	0.000234	0.000342	0.000232	0.000148	0.000229	0.000376

5 rows × 3070 columns

In [307…

```
# 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_x,
result_y,stratify=result_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)
```

Random Forest Classifier

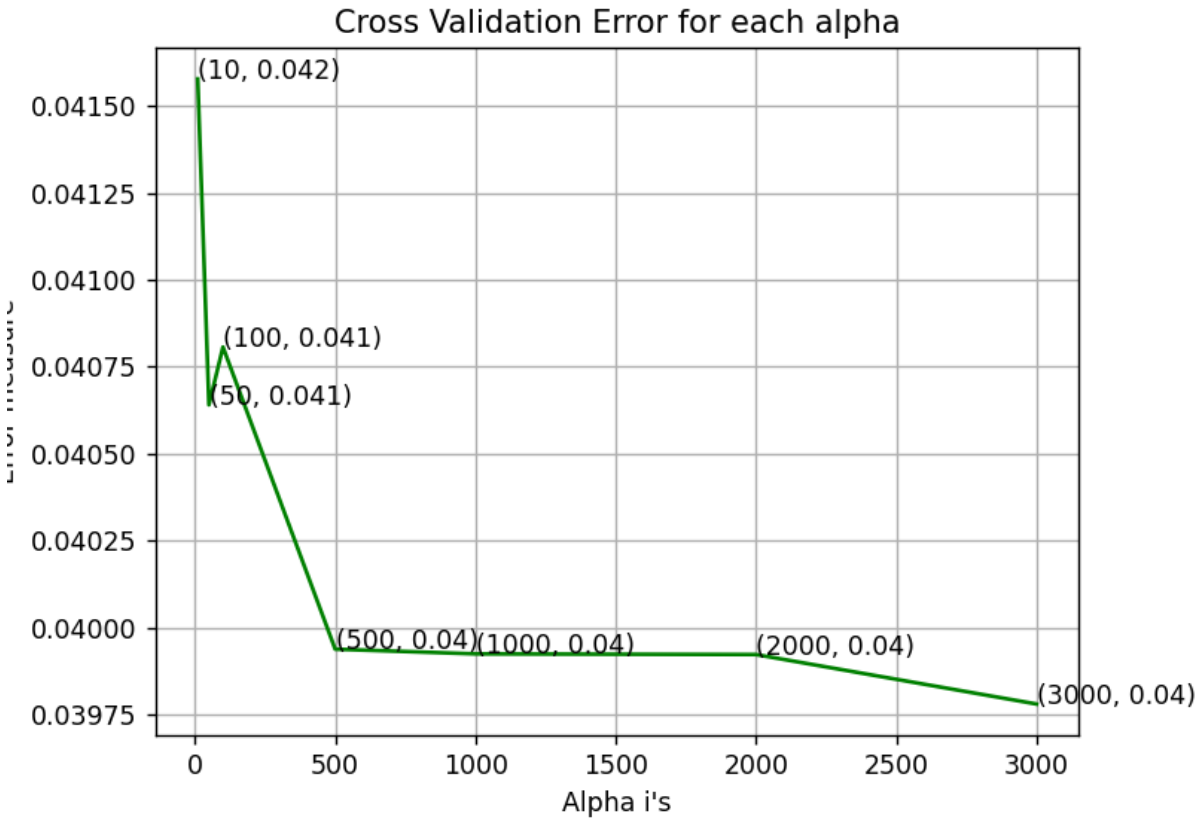
In [314…

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

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

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

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

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

r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
```

```
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_st
1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
log_loss for c =  10 is 0.041578055189776565
log_loss for c =  50 is 0.04064031441982335
log_loss for c =  100 is 0.04080662167927927
```

```
log_loss for c =  500 is 0.039937435966695155
log_loss for c =  1000 is 0.03992384858627934
log_loss for c =  2000 is 0.03992198165370125
log_loss for c =  3000 is 0.039779591440703005
```

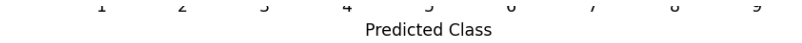



```
For values of best alpha =  3000 The train log loss is: 0.013780147189282489
For values of best alpha =  3000 The cross validation log loss is: 0.039779591
440703005
For values of best alpha =  3000 The test log loss is: 0.02629104306062404
```

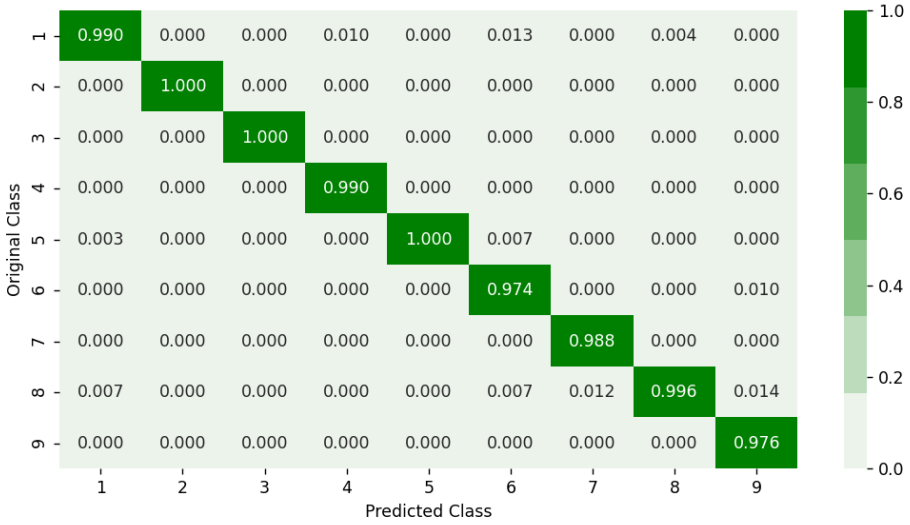
In [315…

```
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

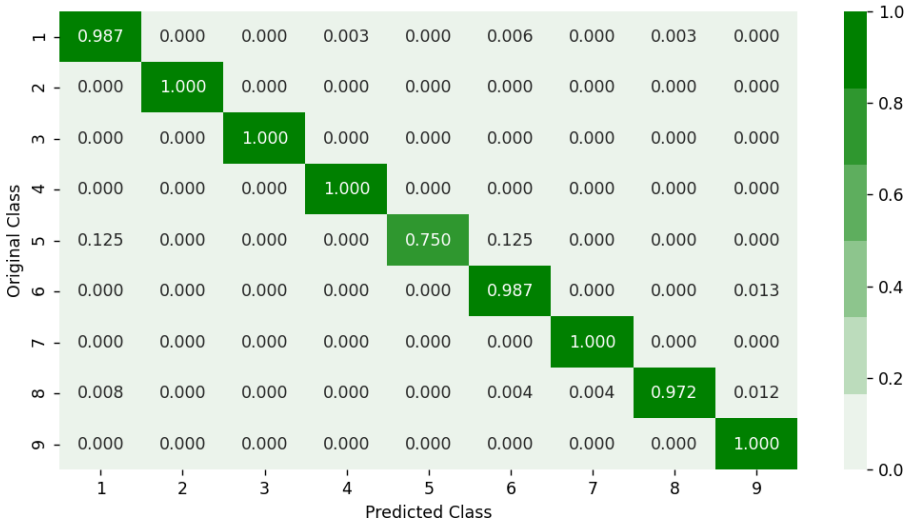
```
Number of misclassified points  0.6909258406264395
---------------------------------------------------- Confusion matrix
-----------------------------------------------------
```




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

Xg-Boost

In [135…

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-
parameter-tuning-xgboost-with-codes-python/
x_cfl=XGBClassifier(nthread=-1, eval_metric='mlogloss')

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,verbos
1)
random_cfl1.fit(X_train,y_train)

random_cfl1.best_estimator_
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  6.9min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 25.5min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 37.4min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 44.2min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 45.4min remaining: 10.0m
in
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 48.8min remaining:  3.1m
in
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 48.9min finished
```

Out[135…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='mlogloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=3, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=2000, n_jobs=8,
              nthread=-1, num_parallel_tree=1, objective='multi:softprob',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=Non
e,
              subsample=0.5, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

In [329…

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

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

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

x_cfl=XGBClassifier(n_jobs = -1, learning_rate=0.3, max_depth=3,
n_estimators=2000,eval_metric='mlogloss')
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))
```

```
train loss 0.008027763240738985
cv loss 0.021477194597758122
test loss 0.012303288509108485
```

Conclusion

In [330…

```
#https://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Feature", "Train-Loss", "Test-Loss"]

x.add_row(["knn", 'bytes(bigrams)', 0.095, 0.16])
x.add_row(["logistic regression", 'bytes(bigrams)', 0.673, 0.694])
x.add_row(["Random forest", 'bytes(bigrams)', 0.017, 0.049])
x.add_row(["Xg-B", 'bytes(bigrams)', 0.016, 0.048])
x.add_row(["Random forest", 'byte(unigram + bigrams) + asm + image-asm', 0.013, 0.026])
x.add_row(["Xg-B", 'byte(unigram + bigrams) + asm + image-asm', 0.008, 0.012])

print(x)
```

```
+---------------------+-------------------------------------------+-----------+-----------+
|        Model        |                  Feature                  | Train-Loss | Test-Loss |
+---------------------+-------------------------------------------+-----------+-----------+
|         knn         |              bytes(bigrams)               |   0.095   |    0.16   |
| logistic regression |              bytes(bigrams)               |   0.673   |   0.694   |
|    Random forest    |              bytes(bigrams)               |   0.017   |   0.049   |
|         Xg-B        |              bytes(bigrams)               |   0.016   |   0.048   |
|    Random forest    | byte(unigram + bigrams) + asm + image-asm |   0.013   |   0.026   |
|         Xg-B        | byte(unigram + bigrams) + asm + image-asm |   0.008   |   0.012   |
+---------------------+-------------------------------------------+-----------+-----------+
```

In [ ]: