MICROSOFT MALWARE DETECTION ASSIGNMENT'

IN THIS PROBLEM STATEMENT WE HAVE THE RAW DATA WITH 9 TYPES OF MALWARE WITH TWO TYPES OF FILES THE FILES ARE

- .ASM FILES
- .BYTE FILES

WE ARE JUST GIVEN WITH THESE DATA WE HAVE TO FIND THE MALWARE EACH PARTICULAR FILE IT BELONGS TO

MALWARES WE HAVE ARE

- RAMMIT
- LOLLIPOP
- KELIHOS
- VUNDO
- SIMDA
- TRACUR
- KELIHOS VERSION1
- OBFUSCVATOR
- GATAK

WE HAVE THE 10868 BYTE FILES AND .ASM FILES WE AHVE TO PERFORM THE EXPLORATORY DATA ANLYASIS,FEATURE EXTRACTION ,FEATURE ENGINEERING,MODEL SELECTION ,HYPERPARAMETER TUNING THE MODELS AND SHOULD TAKE THE STEPS TO

In [0]:
```
import pandas as pd
```

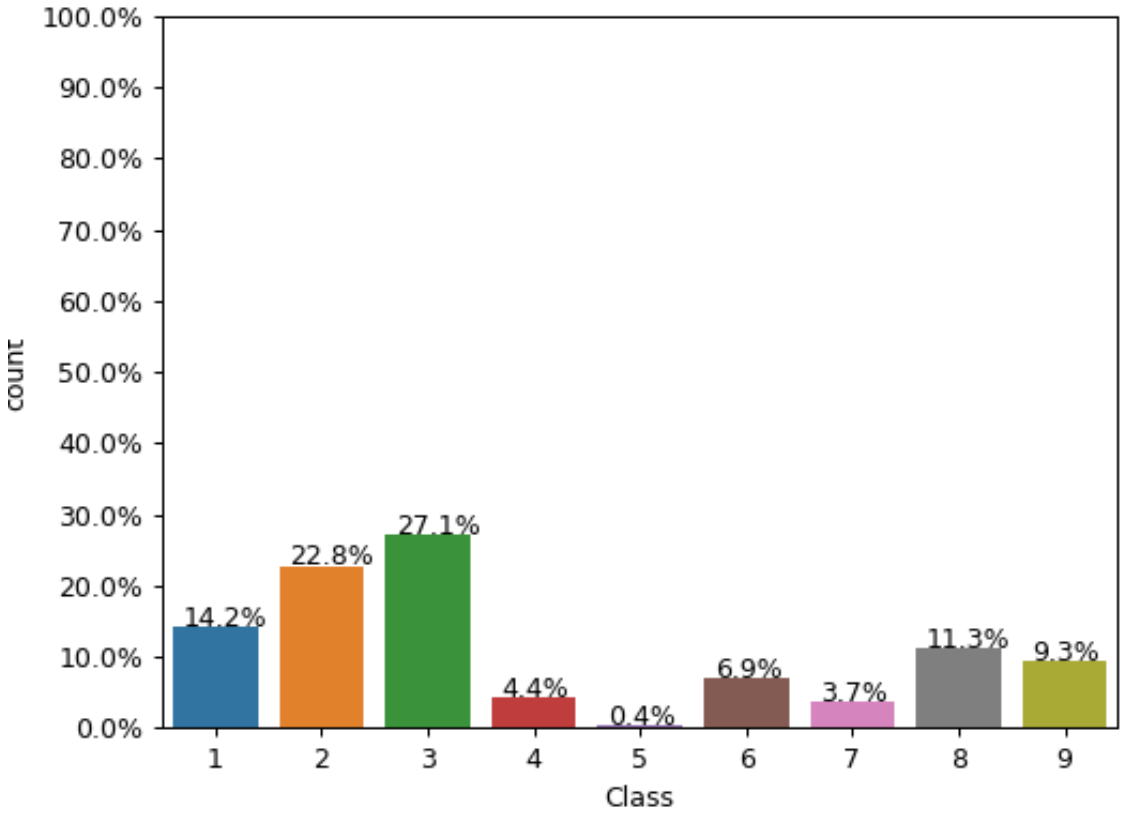
In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
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
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:
29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and
should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

THIS IS THE PROCESS OF EXPLORATORY DATA ANALYSIS CHECKING THE IMBALANCE IN THE DATA WE CAN APPLY STARATEGY TO HANDLE THE DATA IMBALANACE BY TAKING MEASURES OF UNDERSAMPLING AND OVER SAMPLING OR TAKING CARE WE DIVIDE THE DATA INTO TRAIN TEST SPLITTING WE CAN APPLLY STRATEGY TO SPLIT THE DATA BASED ON THE Y LABEL

In [0]:
```
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

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

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

```
<IPython.core.display.Javascript object>
```



THIS IS THE IMPORTANT STEP OF FEATURE EXTRACTION WHERE WE SEE TH E SIZE OF THE FILE WHICH IS THE IMPORTANT THING IN THE PREDICTION OF THE MALWARE BEACUSE THE MALWARE CAN BE BELONG TO MANY SIZES LATER WE CAN APPENF THIS AS THE FEATURE.

In [0]:
```
#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, s
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat
    statinfo=os.stat('byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

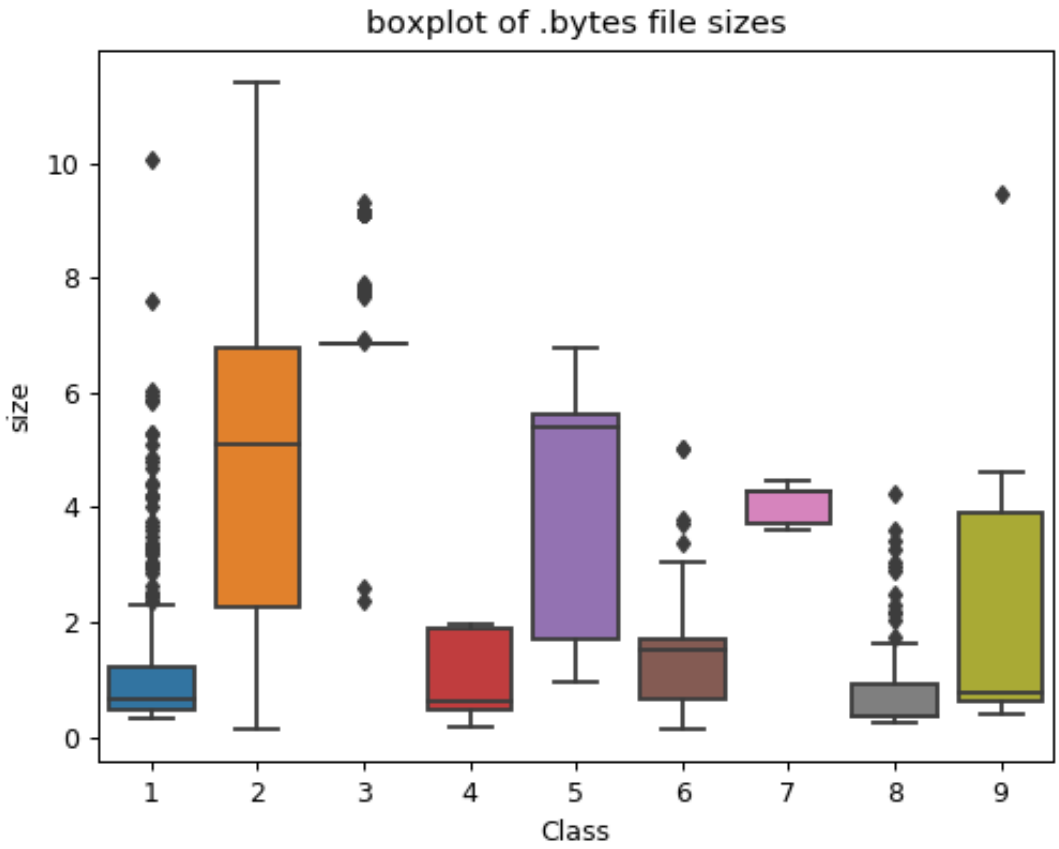
```
                     ID      size  Class
0  05aiMRw13bYWqZ8OHvjl  5.945557      2
1  05EeG39MTRrI6VY21DPd  0.430664      1
2  05IXcWGxvnkto4sq17zZ  1.303955      2
3  05Kps4iFw8mOLJZQrb1H  6.220703      2
4  05LHG8fR3iPn6agIo9z7  2.380615      6
```

In [0]:
```
data_size_byte.to_csv('classsize.csv')
```

In [0]:
```
data_size_byte=pd.read_csv('classsize.csv')
```

In [0]:
```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```

```
<IPython.core.display.Javascript object>
```



ASSIGNMENT 1 STARTS FROM HERE WE WILL TAKE THE BIGRAMS OF THE DATA IN FILES AND VISUALISE THE PERFORMANCE USING THE MODELS.

THE CELL BELOW SHOWS THE GENERATION OF UNIGRAMS FORM THE BYTE FILES. AS THIS IS ALREADY DONE AS A PART OF ASSIGNMENT WE GONNA CONSIDER THE BIGRAMS GENERATED FROM THE FILES NAD OBTAIN THE PERFORMANCE.

In [0]:
```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#------------------
#we remove the starting address 00401000

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(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()
```

In [0]:
```
byte_features1=byte_features.iloc[:,:258]
byte_features1=byte_features1['ID'].map(lambda x: x.replace('.txt',''))
```

In [0]:
```
result = pd.merge(byte_features, data_size_byte,on='ID', how='left')
print(result.head())
print(result.shape)
```

```
                     ID      0     1     2       3     4     5     6     7
\
0  05EeG39MTRrI6VY21DPd   19419   955   693     656   832   463   370   510
1  05IXcWGxvnkto4sq17zZ   15166  1049   566     642   779   377   478   304
2  05Kps4iFw8mOLJZQrb1H   77863 50918 48451  334222  8696  7113  6276  6626
3  05LHG8fR3iPn6agIo9z7  172538  3489  2653    3551  2569  2667  2390  2277
4  05rJTUWYAKNegBk2wE8X   79568  6445  3143    2517  4982  1824  2125  2964

      8  ...      f9     fa    fb     fc     fd    fe     ff  \
0   836  ...     354    472   393    384    303   390   2104
1  1166  ...    2526   1004  3225  23638  38048  3477   6900
2  6893  ...     188  94953   196   7456    176  7964  42052
3  2058  ...    2119   1960  2050   1943   2138  1969  64831
4  8301  ...    1511   1604  2407   5196   4614  4448  47484

   ??05aiMRw13bYWqZ8OHvjl.txt      size  Class
0                        8512  0.509766      1
1                       16368  1.543457      2
2                       11076  7.363281      2
3                       21260  2.817871      6
4                       44280  2.789551      1

[5 rows x 260 columns]
(3017, 260)
```

In [0]:
```
s="0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1
l=[]
l.append(s.split(','))
a=np.array(l)
a1=a.flatten()
l=list(a1)
print(len(l))
count=1
arr=np.zeros((256,256),dtype=int)
for i in l:
    for j in l:
        arr[int(i,16)][int(j,16)]=count
        count+=1

print(arr)
```

```
256
[[    1     2     3 ...   254   255   256]
 [  257   258   259 ...   510   511   512]
 [  513   514   515 ...   766   767   768]
 ...
 [64769 64770 64771 ... 65022 65023 65024]
 [65025 65026 65027 ... 65278 65279 65280]
 [65281 65282 65283 ... 65534 65535 65536]]
```

GENERATION OF BIGRAMS AND FEATURISE THEM WITH THE BASED ON THE HEXADECIMALDATA PRESENT INSIDE THE EACH FILE AND APPENDING THEM WITH THE SIZE AND CLASS LABEL BEFORE APPLYING THE MODELS.

In [0]:
```
print(256*256)
```

```
65536
```

In [0]:
```
files = os.listdir('byteFiles')
filenames2=[]
feature_matrix = np.zeros((len(files),65537),dtype=int)
```

In [0]:
```
k=0
byte_feature_file=open('resultfin.csv','w+')
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for i in range(len(line)):
                    if i<(len(line)-1):
                        a=line[i]
                        b=line[i+1]
                        if (a!='??') and (b!='??'):
                            score=arr[int(a,16)][int(b,16)]
                            feature_matrix[k][score]+=1


        byte_flie.close()
    for i in feature_matrix[k]:

        byte_feature_file.write(str(i)+",")
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [0]:
```
import pandas as pd
das=pd.read_csv('resultfin.csv',nrows=10)
print(das)
print(das.shape)
```

```
   05aiMRw13bYWqZ8OHvjl.txt  0  25141   782  140    77  211   21   25   27  \
0  05EeG39MTRrI6VY21DPd.txt  0  10352   178   74   122   77   41   13   30
1  05IXcWGxvnkto4sq17zZ.txt  0   8195   144   63    32   34    7    8    8
2  05Kps4iFw8mOLJZQrb1H.txt  0  16493  1185  560  2149  439  285  278  282
3  05LHG8fR3iPn6agIo9z7.txt  0  93134   325  256   403  251  237  273  311
4  05rJTUWYAKNegBk2wE8X.txt  0  37601   533  317   138  233   55   59  693
5  065EZhxgbLRSHsB87uIF.txt  0  20862   711   63    38  171   21   21   14
6  06aLOj8EUXMByS423sum.txt  0  26357  1014  457   491  518  371  317  193
7  06arUi9q3wHS2C8RZxeB.txt  0   5729    44   28    83   13   47   81   31
8  06KfrF7ltESna2ZHPVp5.txt  0   6122    33   23    70   13   56   16   38
9  06osXqPUVM1HbvBGNncT.txt  0   4051    21   16    44   14   29   10   26

        ...        220.1  133.2  8.1020  18.91  20.79  5.998  24.46  22.74  \
0       ...            2      9       1      3      2      8      3      5
1       ...            7      2       1      0      0      1      2      6
2       ...           22     12       0      8     18      7     14     13
3       ...          118    131     109    113    103    112    118    122
4       ...           39     24       3      5      2     14     13     17
5       ...           37     62      49     87    227     40     38    124
6       ...           43     29      33     53     86     73    108    111
7       ...            9     11      12     12     10      8     13     10
8       ...           15      7       8     13     14      9     20     16
9       ...           12     14      11     16      9     12     15     10

    6836  Unnamed: 65538
0    512             NaN
1   2405             NaN
2   3445             NaN
3  29213             NaN
4  14747             NaN
5   8864             NaN
6   7107             NaN
7     74             NaN
8    177             NaN
9     86             NaN

[10 rows x 65539 columns]
(10, 65539)
```

THE FILES ARE UPLOADED INTO THE DRIVE AFTER GENERATING THE IMAGE FEATURES AND MODELS ARE APPLIED ON THEM

we have obtained the bigrams of the byte files using the hexadecimal data .

we got the features and named the features we got finally the 65538 features

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% |████████████████████████████████| 993kB 20.2MB/s ta 0:00:01
  Building wheel for PyDrive (setup.py) ... done
```

In [0]:
```
link = 'https://drive.google.com/open?id=1cphD9gkKoD8Ojw8No_K9Wv2nyp1kNUSs' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1cphD9gkKoD8Ojw8No_K9Wv2nyp1kNUSs
```

generating the column names for byte files rangiing to 65539

In [0]:
```
l=[i for i in range(65539)]
l[0]='ID'
print(l)
```

```
['ID', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 4
0, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 5
9, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 7
8, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 9
7, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 1
13, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 12
8, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 15
9, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 19
0, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205,
206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 22
1, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236,
237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 25
2, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267,
268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 28
3, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 31
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('resultfin.csv')
df3 = pd.read_csv('resultfin.csv',names=l)
```

In [0]:
```
print(df3.shape)
```

In [0]:
```
df3.head(5)
```

In [0]:
```
df4=df3['ID'].map(lambda x: x.replace('.txt',''))
df4.head(5)
df3['ID']=df4
df3.head(5)
df3=df3.drop(columns=[65538])
```

In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [0]:
```
print(df3.columns)
```

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
link = 'https://drive.google.com/open?id=1EeuLtqkPWV875y0YUwTN-jeOzWCrS8Ac' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1EeuLtqkPWV875y0YUwTN-jeOzWCrS8Ac
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('classsize.csv')
dataframe = pd.read_csv('classsize.csv')
```

In [0]:
```
dataframe.head(5)
print(dataframe.columns)
dataframe=dataframe.drop(columns=['Unnamed: 0'])
dataframe.head(5)
```

```
Index(['Unnamed: 0', 'ID', 'size', 'Class'], dtype='object')
```

Out[6]:

	ID	size	Class
0	05aiMRw13bYWqZ8OHvjI	5.945557	2
1	05EeG39MTRrI6VY21DPd	0.430664	1
2	05IXcWGxvnkto4sq17zZ	1.303955	2
3	05Kps4iFw8mOLJZQrb1H	6.220703	2
4	05LHG8fR3iPn6agIo9z7	2.380615	6

In [0]:
```
df3.head(5)
```

Out[16]:

	ID	1	2	3	4	5	6	7	8	9	...	65528	65529	655
0	05aiMRw13bYWqZ8OHvjI	0	25141	782	140	77	211	21	25	27	...	49	220	1
1	05EeG39MTRrI6VY21DPd	0	10352	178	74	122	77	41	13	30	...	1	2	
2	05IXcWGxvnkto4sq17zZ	0	8195	144	63	32	34	7	8	8	...	10	7	
3	05Kps4iFw8mOLJZQrb1H	0	16493	1185	560	2149	439	285	278	282	...	135	22	
4	05LHG8fR3iPn6agIo9z7	0	93134	325	256	403	251	237	273	311	...	120	118	1

5 rows × 65538 columns

we have obtained the features and we will add one more feature which is thesize of the malware and we append the class label to the data. later we will use the models to train and test

In [0]:
```
result=pd.merge(df3,dataframe,on='ID',how='left')
result.head(5)
```

In [0]:
```
result.to_csv('resultingdataframe.csv')
```

In [0]:
```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once in a notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Create & upload a file.
uploaded = drive.CreateFile({'title': 'resultingdataframe.csv'})
uploaded.SetContentFile('resultingdataframe.csv')
uploaded.Upload()
print('Uploaded file with ID {}'.format(uploaded.get('id')))
```

we have uploaded th resultant file into google drive. start executing the file fro the below.otherwise it rewrites the existing file.

In [0]:
```
#resulting dataframe csv file link
https://drive.google.com/open?id=1buRmJX5bRd1y3S_t3RV1hkYNvz7tOx96
```

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% |████████████████████████████████| 993kB 23.1MB/s ta 0:00:01
  Building wheel for PyDrive (setup.py) ... done
```

In [0]:
```
link = 'https://drive.google.com/open?id=1buRmJX5bRd1y3S_t3RV1hkYNvz7tOx96' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1buRmJX5bRd1y3S_t3RV1hkYNvz7tOx96
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('resultingdataframe.csv')
```

In [0]:
```
result = pd.read_csv('resultingdataframe.csv')
```

In [0]:
```
result=result.set_index('Unnamed: 0')
```

In [0]:
```
print(result.head(5))
```

```
                              ID  1      2     3    4     5    6    7    8  \
Unnamed: 0
0           05aiMRw13bYWqZ8OHvjl  0  25141   782  140    77  211   21   25
1           05EeG39MTRrI6VY21DPd  0  10352   178   74   122   77   41   13
2           05IXcWGxvnkto4sq17zZ  0   8195   144   63    32   34    7    8
3           05Kps4iFw8mOLJZQrb1H  0  16493  1185  560  2149  439  285  278
4           05LHG8fR3iPn6agIo9z7  0  93134   325  256   403  251  237  273

                9  ...    65530  65531  65532  65533  65534  65535  65536  \
Unnamed: 0         ...
0              27  ...      133      8     18     20      5     24     22
1              30  ...        9      1      3      2      8      3      5
2               8  ...        2      1      0      0      1      2      6
3             282  ...       12      0      8     18      7     14     13
4             311  ...      131    109    113    103    112    118    122

              65537      size  Class
Unnamed: 0
0              6836  5.945557      2
1               512  0.430664      1
2              2405  1.303955      2
3              3445  6.220703      2
4             29213  2.380615      6

[5 rows x 65540 columns]
```

APPENDING THE SIZE AND CLASS FOR THE DATA SPLITTING THE DATA BASED ON THE Y LABEL INTO TRAIN CROSS VALIDATAION AND TEST DATA .

In [0]:
```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output v

X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], a
# split the train data into train and cross validation by maintaining same distri
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train
```

In [0]:
```
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])
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
print(X_cv.shape)
print(y_cv.shape)
```

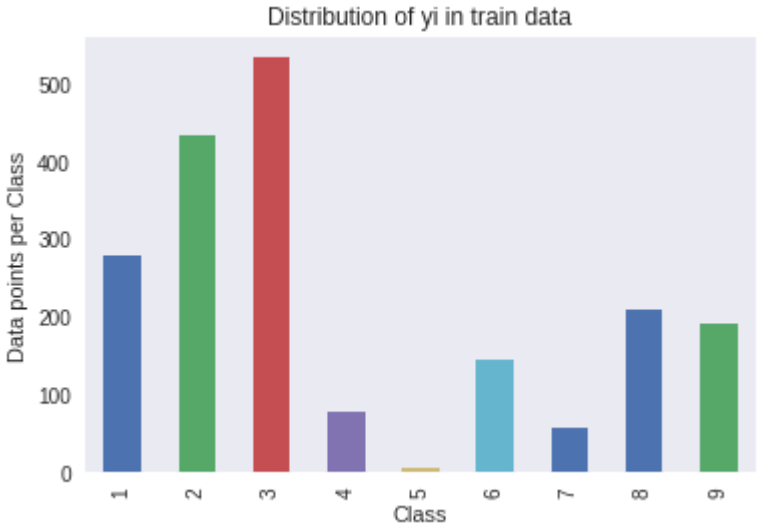
```
Number of data points in train data: 1931
Number of data points in test data: 604
Number of data points in cross validation data: 483
(1931, 65538)
(1931,)
(604, 65538)
(604,)
(483, 65538)
(483,)
```

now we have the data of bigrams we trian the models based on the data and prediuct the y labels

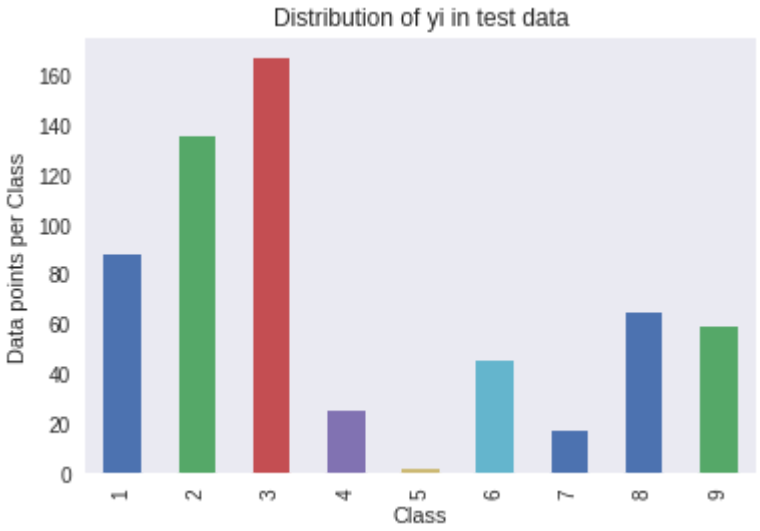
In [0]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
%matplotlib inline
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)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are p

    A =(((C.T)/(C.sum(axis=1))).T)


    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(15,8))
    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True,  fmt=".3f", xticklabels=labels, yticklabels=labels
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*50, "Precision matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(B, annot=True, 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,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))
```

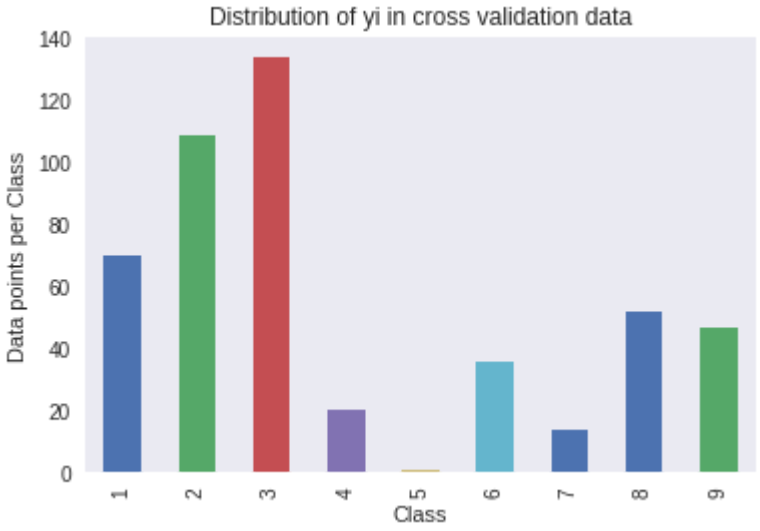
In [0]:
```
# it returns a dict, keys as class labels and values as the number of data points
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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.val


print('-'*80)

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

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

print('-'*80)

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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
```




```
Number of data points in class 3 : 534 ( 27.654 %)
Number of data points in class 2 : 433 ( 22.424 %)
Number of data points in class 1 : 280 ( 14.5 %)
Number of data points in class 8 : 210 ( 10.875 %)
Number of data points in class 9 : 190 ( 9.839 %)
Number of data points in class 6 : 144 ( 7.457 %)
Number of data points in class 4 : 78 ( 4.039 %)
Number of data points in class 7 : 56 ( 2.9 %)
Number of data points in class 5 : 6 ( 0.311 %)
------------------------------------------------------------------------------
-
```




```
Number of data points in class 3 : 167 ( 27.649 %)
Number of data points in class 2 : 136 ( 22.517 %)
Number of data points in class 1 : 88 ( 14.57 %)
Number of data points in class 8 : 65 ( 10.762 %)
Number of data points in class 9 : 59 ( 9.768 %)
Number of data points in class 6 : 45 ( 7.45 %)
Number of data points in class 4 : 25 ( 4.139 %)
Number of data points in class 7 : 17 ( 2.815 %)
Number of data points in class 5 : 2 ( 0.331 %)
------------------------------------------------------------------------------
-
```




```
Number of data points in class 3 : 134 ( 27.743 %)
Number of data points in class 2 : 109 ( 22.567 %)
Number of data points in class 1 : 70 ( 14.493 %)
Number of data points in class 8 : 52 ( 10.766 %)
Number of data points in class 9 : 47 ( 9.731 %)
Number of data points in class 6 : 36 ( 7.453 %)
Number of data points in class 4 : 20 ( 4.141 %)
Number of data points in class 7 : 14 ( 2.899 %)
Number of data points in class 5 : 1 ( 0.207 %)
```

#APPLYING THE MACHINE LEARNING MODELS ON TOP OF DATA GENERAITED AFTER BIGRAMS

4. Machine Learning Models

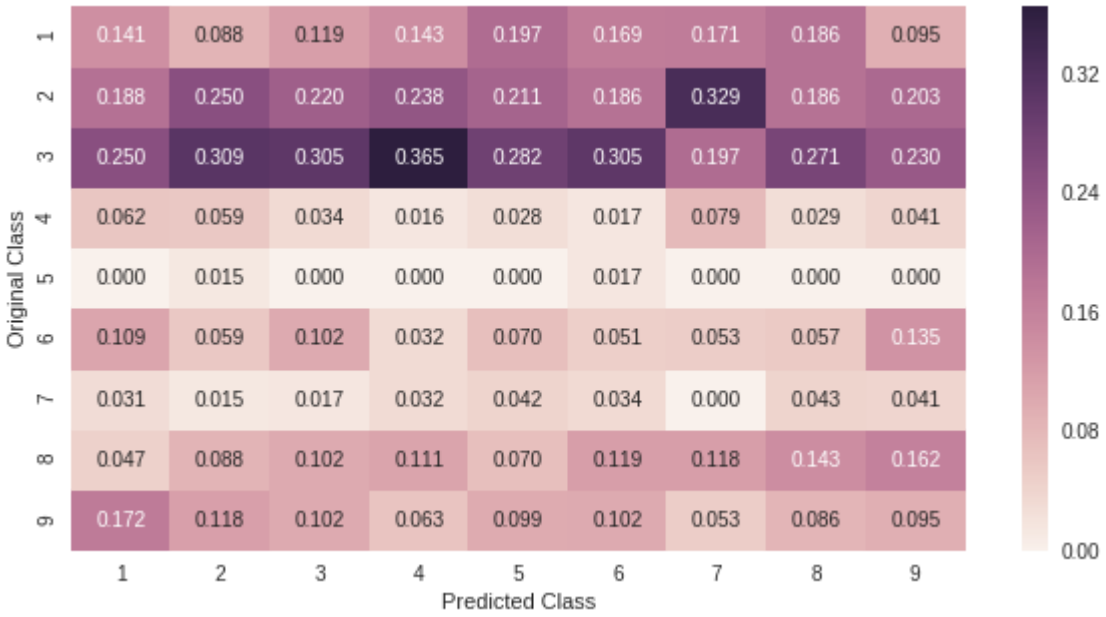
4.1. Machine Leaning Models on bytes files

4.1.1. Random Model

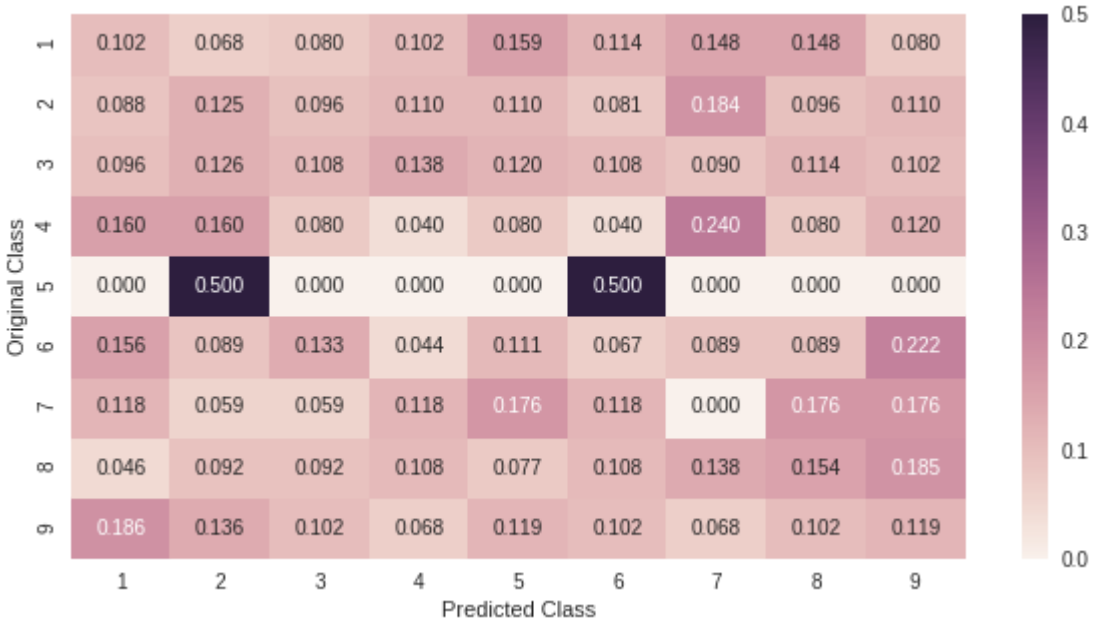
In [0]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their s
# ref: https://stackoverflow.com/a/18662466/4084039

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_pre


# 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

predicted_y =np.argmax(test_predicted_y, axis=1)
y_test=np.array(y_test)

plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.4623613440107714
Log loss on Test Data using Random Model 2.5097805028149645
Number of misclassified points  89.23841059602648
---------------------------------------------------- Confusion matrix -----------
---------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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




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



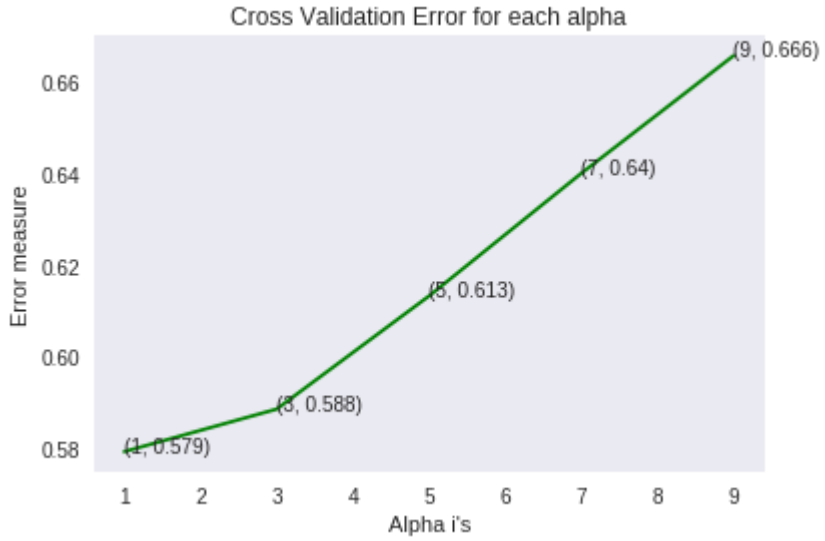

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.2. K Nearest Neighbour Classification

In [0]:

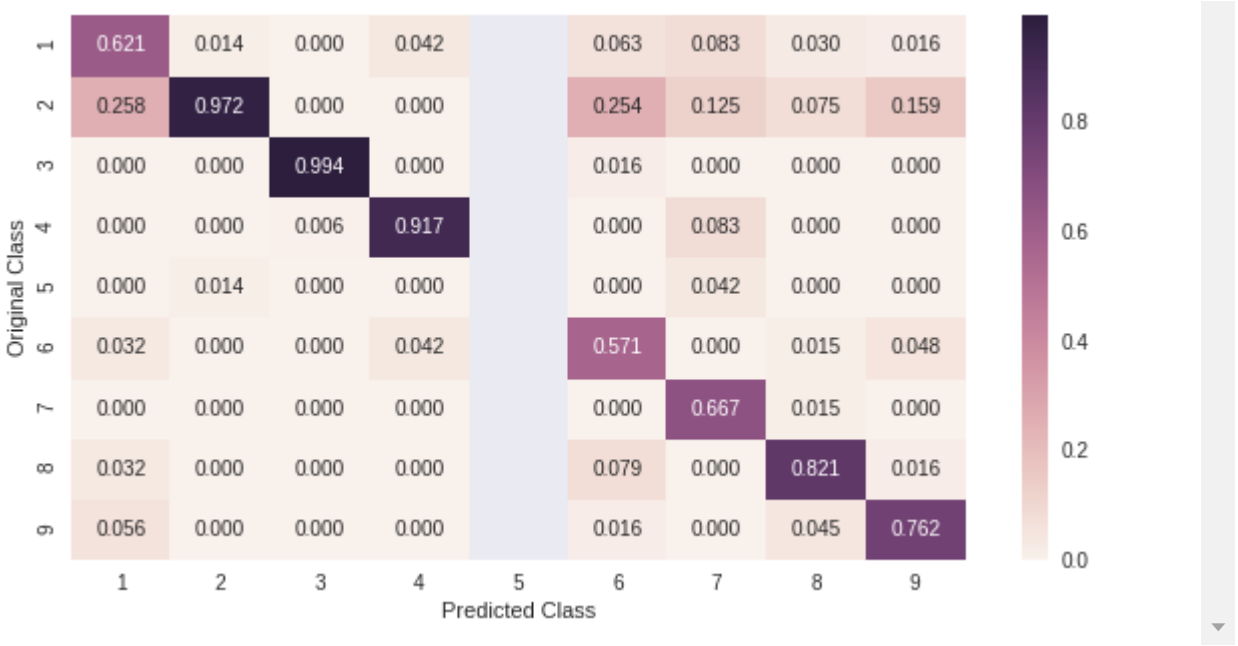
```
alpha = [x for x in range(1, 10, 2)]
cv_log_error_array=[]
for i in alpha:
    print(i)
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, ep

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

```
1
3
5
7
9
log_loss for k =  1 is 0.5791606031704255
log_loss for k =  3 is 0.5884541997326155
log_loss for k =  5 is 0.6131761414736544
log_loss for k =  7 is 0.6400133868683122
log_loss for k =  9 is 0.6657502168011453
```

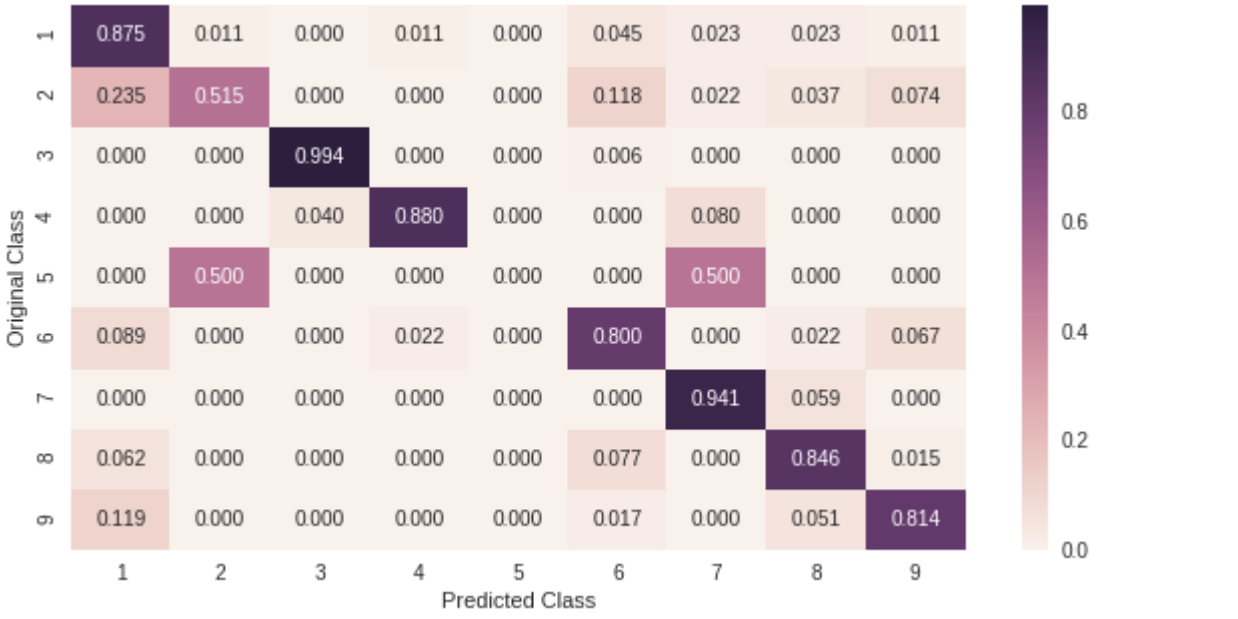



```
For values of best alpha =  1 The train log loss is: 0.2901329071244141
For values of best alpha =  1 The cross validation log loss is: 0.5791606031704
255
For values of best alpha =  1 The test log loss is: 0.7038217108964496
Number of misclassified points  18.874172185430464
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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




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



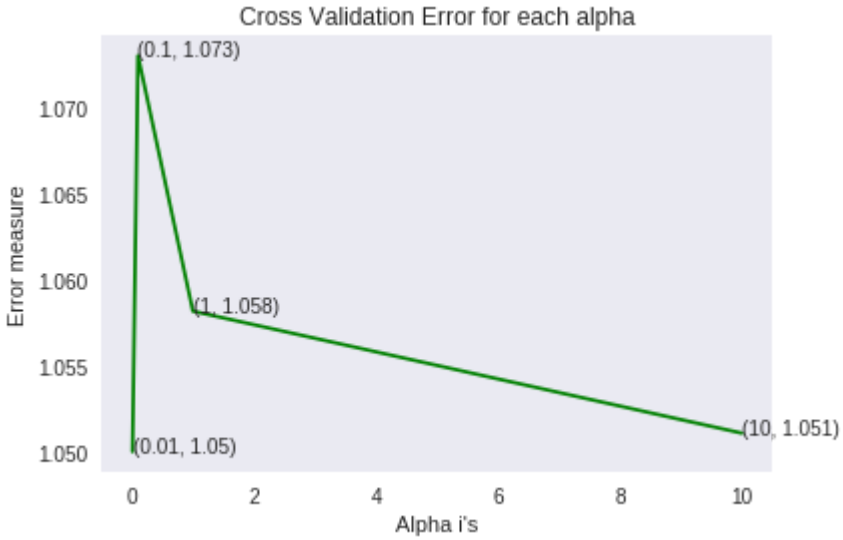

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.3. Logistic Regression

In [0]:

```
alpha = [10 ** x for x in range(-2, 2)]
cv_log_error_array=[]
for i in alpha:
    print('current alpha value is',i)
    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_

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

best_alpha = np.argmin(cv_log_error_array)

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

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balan
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.cl
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.clas
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

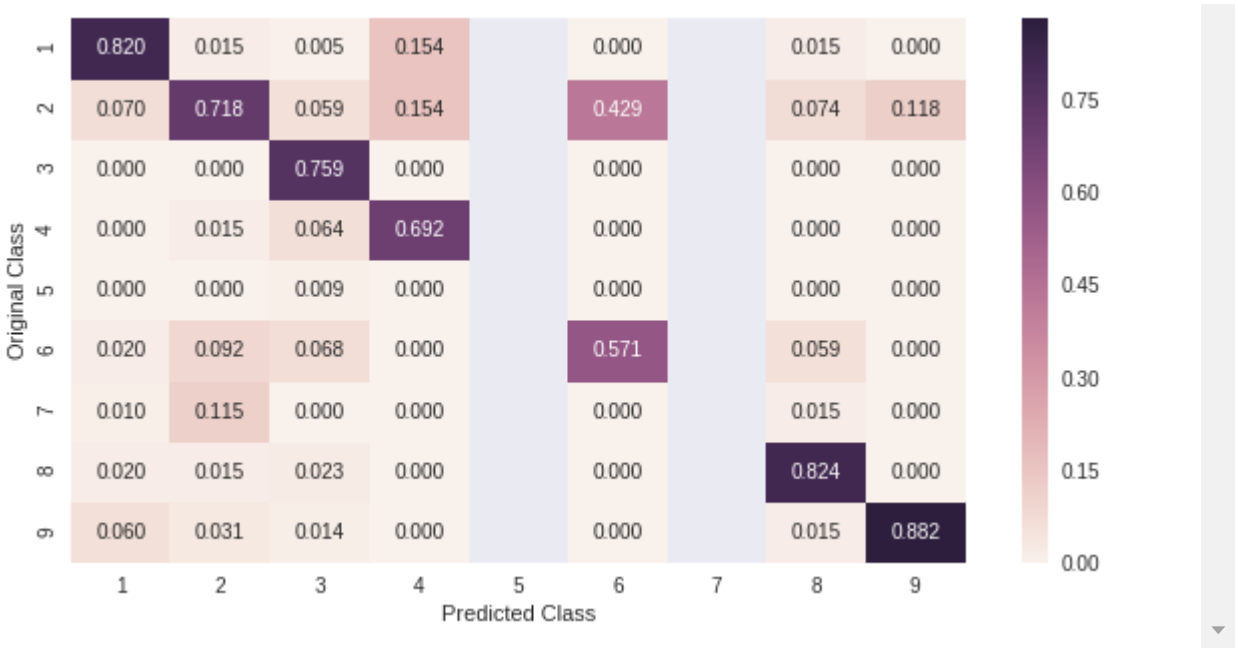
```
current alpha value is 0.01
current alpha value is 0.1
current alpha value is 1
current alpha value is 10
log_loss for c =  0.01 is 1.0499902467589572
log_loss for c =  0.1 is 1.0729569508613177
log_loss for c =  1 is 1.0581383830487323
log_loss for c =  10 is 1.051061617048442
```



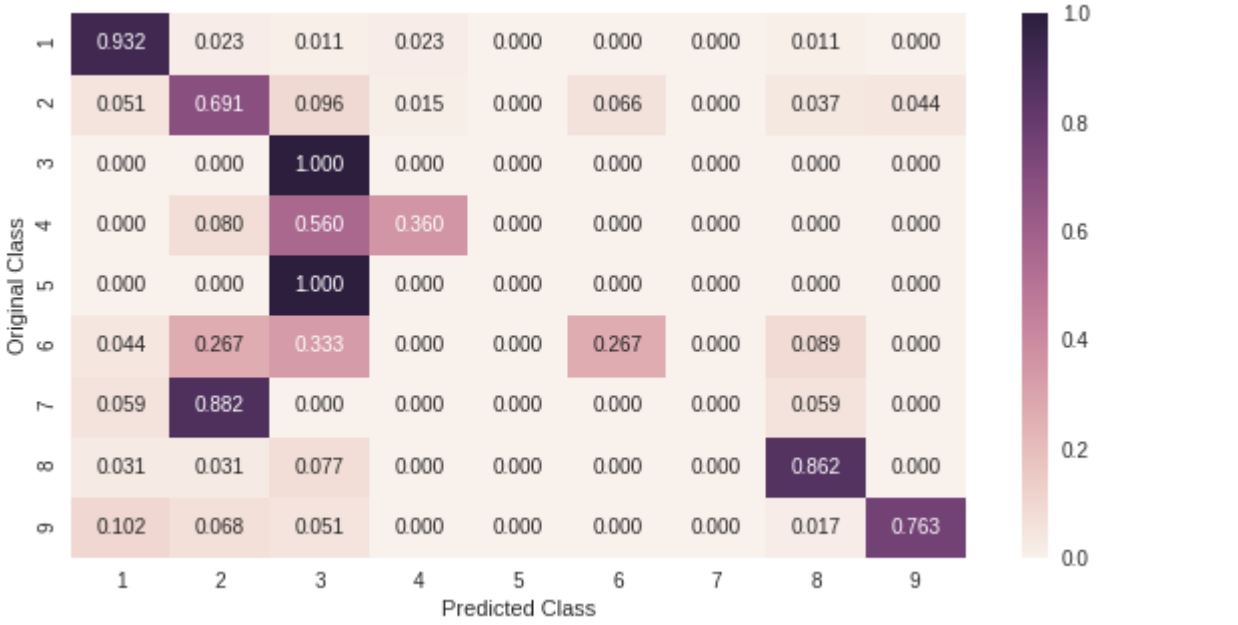

```
log loss for train data 0.9075279622711779
log loss for cv data 1.0499902467589572
log loss for test data 1.0407109799229066
Number of misclassified points  23.013245033112582
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```

```
<Figure size 1080x576 with 0 Axes>
```


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




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



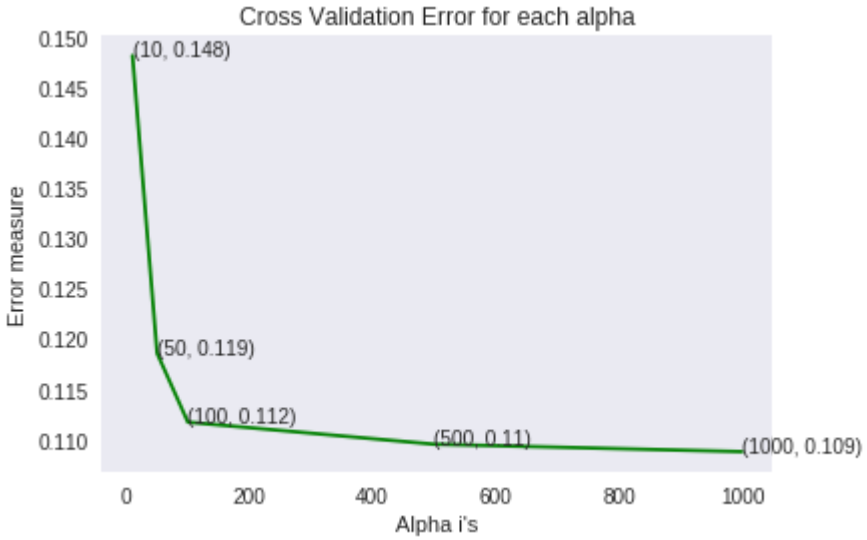

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.4. Random Forest Classifier

In [0]:

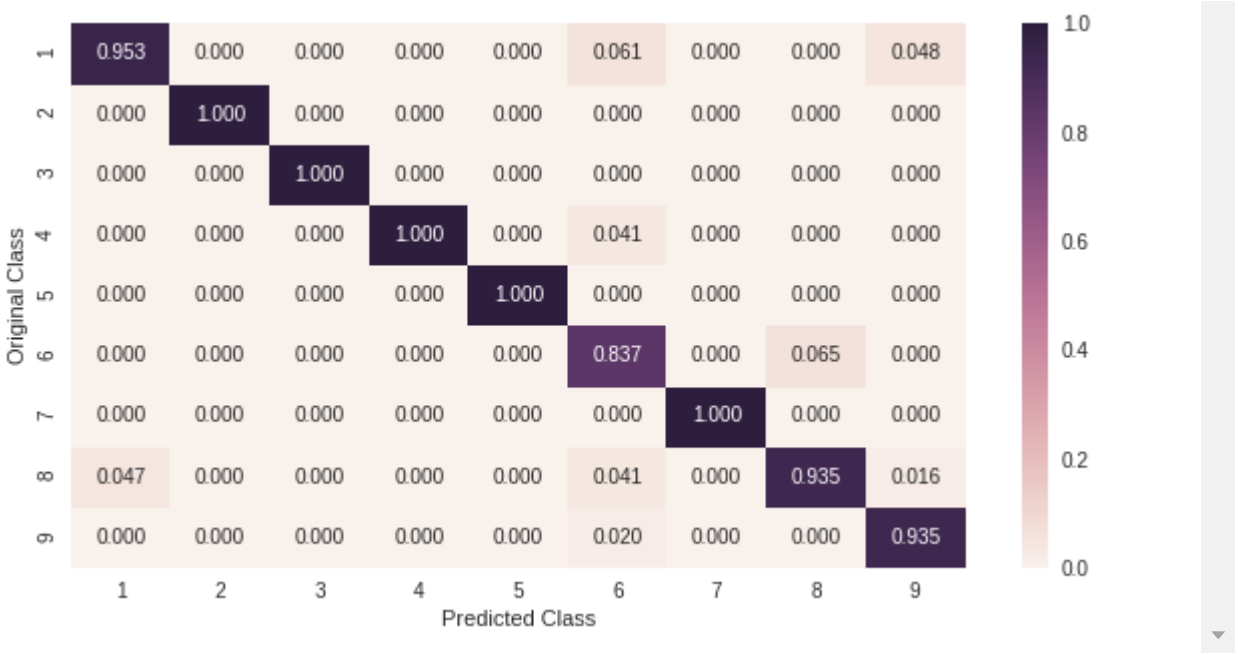
```
alpha=[10,50,100,500,1000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    print('current alpha values is',i)
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, ep

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_job
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:",
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
current alpha values is 10
current alpha values is 50
current alpha values is 100
current alpha values is 500
current alpha values is 1000
log_loss for c =  10 is 0.1481877887567859
log_loss for c =  50 is 0.11856598065687102
log_loss for c =  100 is 0.11170462724068957
log_loss for c =  500 is 0.1095091190751437
log_loss for c =  1000 is 0.1087707479310827
```

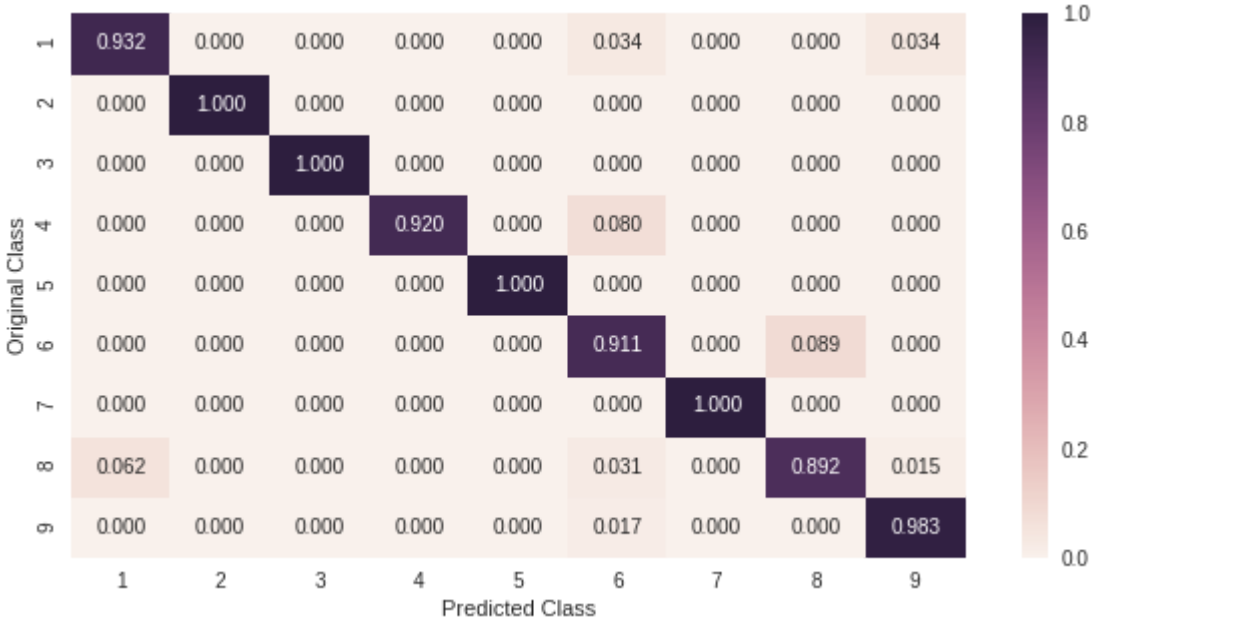



```
For values of best alpha =  1000 The train log loss is: 0.045470831665197164
For values of best alpha =  1000 The cross validation log loss is: 0.1087707479
310827
For values of best alpha =  1000 The test log loss is: 0.15214755949951503
Number of misclassified points  3.3112582781456954
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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

SEESION IS GETTING CRASHED DUE TO THE LARGE DATA REDUCING THEDATA TO 20 PERCENT

In [0]:
```
data_y = result['Class']
zz=result.drop(['ID','Class'], axis=1)
zz=zz.sample(frac=0.2, random_state=99,axis=1)
```

In [0]:
```
# split the data into test and train by maintaining same distribution of output va
X_train, X_test, y_train, y_test = train_test_split(zz, data_y,stratify=data_y,te
# split the train data into train and cross validation by maintaining same distri
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train
```

In [0]:
```
print(X_train.shape)
print(X_test.shape)
print(X_cv.shape)
print(y_train.shape)
print(y_test.shape)
print(y_cv.shape)
```

```
(1931, 13108)
(604, 13108)
(483, 13108)
(1931,)
(604,)
(483,)
```

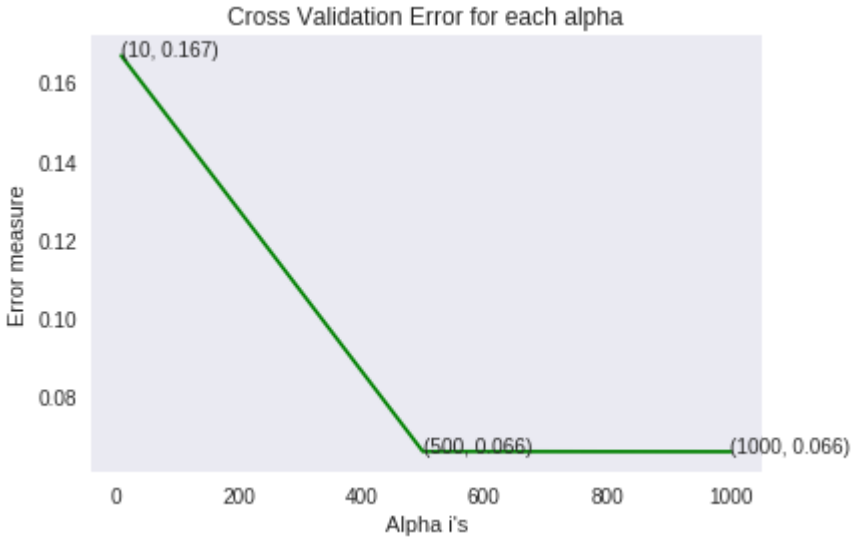
+

4.1.5. XgBoost Classification

In [0]:

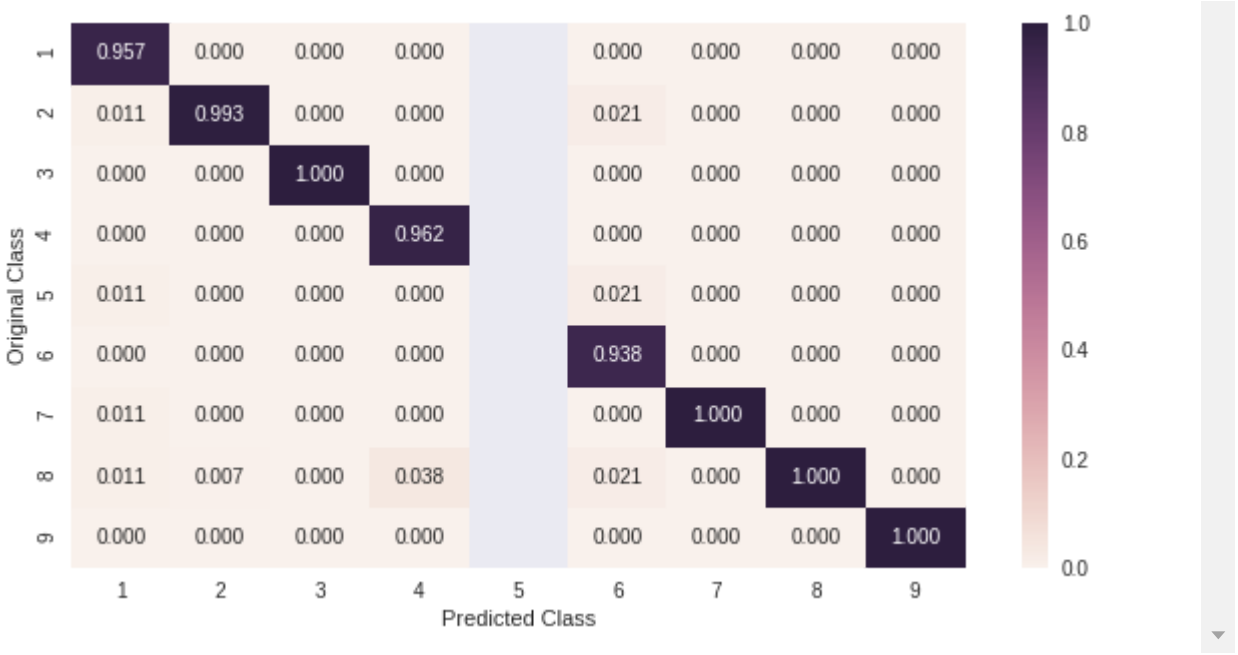
```
alpha=[10,500,1000]
cv_log_error_array=[]
for i in alpha:
    print('currenet alpha value is',i)
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, ep

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

```
currenet alpha value is 10
currenet alpha value is 500
currenet alpha value is 1000
log_loss for c =  10 is 0.1668037527416192
log_loss for c =  500 is 0.06614198935543888
log_loss for c =  1000 is 0.06609387321850578
```

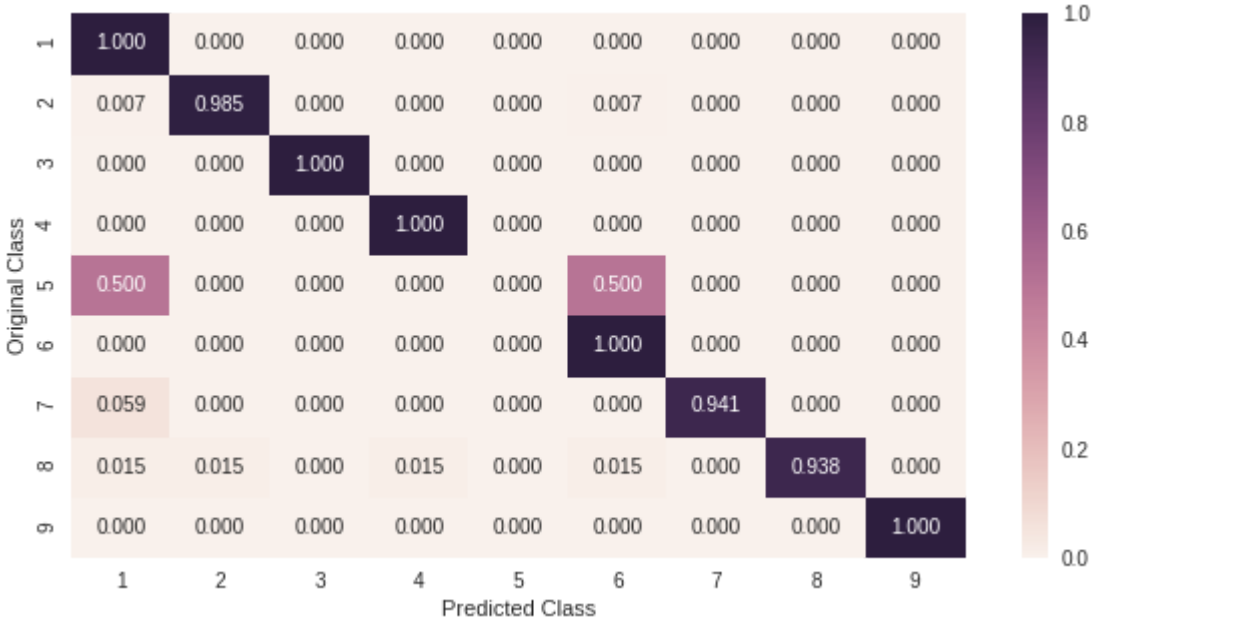



```
For values of best alpha =  1000 The train log loss is: 0.040810080588316 73
For values of best alpha =  1000 The cross validation log loss is: 0.0660938732
1850578
For values of best alpha =  1000 The test log loss is: 0.08880656976892347
Number of misclassified points  1.490066225165563
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [0]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgb
x_cfl=XGBClassifier()

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

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 39.0min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 93.5min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 147.8min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 208.6min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 426.0min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 511.2min finished
```

Out[17]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', 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], 'co
lsample_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='warn', scoring=None, verbose=10)
```

In [0]:
```
print (random_cfl1.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 5, 'learning_rate': 0.2,
'colsample_bytree': 0.5}
```

In [0]:
```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.2, colsample_bytree=0.5, m
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

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

```
train loss 0.03551925222127126
cv loss 0.09008565321978028
test loss 0.09088136809655797
```

DOCUMENTATION, CONCLUSIONS AND KEYTAKEAWAYS FROM THE TASK1

In [0]:
```
import pandas as pd
dta = [['k-nearest neighbors',1,0.29,0.57,0.7],['Logistic regression',0.01,0.9,1.
aa=pd.DataFrame(dta, columns=['model','BEST HYPER PARAMETER','TRAIN LOG LOSS','CV
aa
```

Out[4]:

	model	BEST HYPER PARAMETER	TRAIN LOG LOSS	CV LOG LOSS	TEST LOGLOSS
0	k-nearest neighbors	1	0.290	0.57	0.70
1	Logistic regression	0.01	0.900	1.04	1.04
2	RANDOM FOREST classifier	1000	0.045	0.10	0.15
3	XG boost with calibrated classifier cv	1000(number of estimators)	0.040	0.06	0.08
4	xgbOOST classifier WITH RANDOM SEARCH CV	2000(number of estimators),maximum depth=5,LEA...	0.030	0.09	0.09

the key take aways are due to 65536 columns instead of 256 columns the model is affected by the curse of dimensionality where we were able to achieve the 0.07 log loss in case of 256 columns we are able to achieve 0.08 log loss . the major cahnge occured in logistic regression model where due to more number of dimensions log loss is nearly 1.0 which is 0.7 in the previous case

##DOCUMENTATION

- we have the data of .byte files we have parse throught the every file and otained the bigram features of dimensions 65536.
- we have the features and aplitted the data into train,crossvalidataion and test
- we have applied models like

- knn
- logistic regression
- random forest
- xg boost with calibrated classifier
- ag boost with random search
- log loss comapred to the 256 features it got degraded but still uisng the xgboost with calibrated classifier cv tunung the number of estimators we are able to achieve the log oloss of 0.8 which is 0.7 in the previous case

#FROM HERE TASK 2 STARTS

Task 2 using the dchads github account analysis done by him and selecting the best features and models

key take aways from the dchads analysis is

* select the best features and apply the extra trees classifier on top of it.

* we do this analysis selecting the 10percent best features of total features

* 20 percent of best features

* 30 percent of best features so on..

* we select the features based on the chi squared test performed on top of them.

* using the extra trees classifier we obtain the parameters of the model such as logloss.

* finally we select the percent of features which are best and apply models on them and predcit the y labels.

use the image features with the byte and .asm features and aplly models on them like

- xgboost
- extra trees classifier
- logistic regression
- naive bayes
- knn

BASED ON THE RESULTS OF LOGLOSS AND CONFUSION MATRIX OBTAIN THE BEST MODEL.

FINALLY WE WILL GET THE OPIMUM FEATURES WHAICH ARE TO BE CONSIDERD AND THE MODEL TO BE EMPLOYED ON THE DATA.

for merging the .bytes features sharable link

https://drive.google.com/open?id=1Gwb9wEDI4tKhSnOHVwTJyqVe1yTlDh3IEYUXBFdmB2c (https://drive.google.com/open?id=1Gwb9wEDI4tKhSnOHVwTJyqVe1yTlDh3IEYUXBFdmB2c)

#asm file sharable link https://drive.google.com/open?id=1XhfwsDj4G_R94oyW-QZJHAKfoilmKE1G (https://drive.google.com/open?id=1XhfwsDj4G_R94oyW-QZJHAKfoilmKE1G)

as documented by dchad the .byte image features are weak learners.we are extracting the image features of the .asm files anfd teake as the features . we consider the 1000 pixel of data and append them with the .asm features and .byte features we apply model on top of them and visualise the results.

function below will parse through all the .asm files and generate the image features and converted into csv file

we will upload this file into drive and append the features and apply the models.

this is the code used to obtain the image featrues and stores in the form of csv file . later this csv file uploaded to drive and used featurise and train the model using the iamge features. as a note we have considerd 0only 1000 pixel of data .

In [0]:
```
from multiprocessing import Pool
import os
from csv import writer
import numpy as np
import math
import scipy.misc
import array
import time as tm
```

In [0]:
```
def read_image(filename):
    #startTime = time.time()
    f=open(filename,'rb')
    ln=os.path.getsize(filename)
    width=int(ln**0.5)
    rem=ln%width
    a=array.array("B")
    a.fromfile(f,ln-rem)
    f.close()
    print(int(len(a)/width),width)
    print(type(a))
    g=np.reshape(a,(int(len(a)/width),width))
    print(g.shape)

    g=np.uint8(g)

    g=np.resize(g,(1000,))
    return list(g)
```

In [0]:
```
def extract_byte_image_features(tfiles):
    asm_files = [i for i in tfiles if '.bytes' in i]
    ftot = len(asm_files)

    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' + str(pid) + '-train-image-features-byte.csv'
    print('feature file:', feature_file)

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("BYTE_{:s}".format(str(x))) for x in rang
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image('bbytefiles/'+ fname)
            outrows.append([file_id] + image_data)
            print('yes')
            # Print progress
            if (idx+1) % 10 == 0:
                print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

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

In [0]:
```
def extract_asm_image_features(tfiles):
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)
    count=0
    pid = os.getpid()
    print('Process id:', pid)
    feature_file = 'data/' + str(pid) + '-test-image-featuresout-asm.csv'
    print('feature file:', feature_file)

    outrows = []
    with open(feature_file,'w') as f:
        fw = writer(f)
        column_names = ['filename'] + [("ASM_{:s}".format(str(x))) for x in range
        fw.writerow(column_names)
        for idx, fname in enumerate(asm_files):
            file_id = fname.split('.')[0]
            image_data = read_image('basmFiles/' + fname)
            outrows.append([file_id] + image_data)
            print(count)
            count+=1
            # Print progress
            if (idx+1) % 10 == 0:
                print(pid, idx + 1, 'of', ftot, 'files processed.')
                fw.writerows(outrows)
                outrows = []

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

In [0]:
```
# TRAIN FILES BYTE
# Now divide the train files into four groups for multiprocessing
start_time = tm.time()
ext_drive = 'basmFiles'
tfiles = os.listdir(ext_drive)
extract_asm_image_features(tfiles)
```

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
    100% |████████████████████████████████| 993kB 12.5MB/s ta 0:00:01
  Building wheel for PyDrive (setup.py) ... done
```

In [0]:
```
link = 'https://drive.google.com/open?id=10Nrz6CTKEG0oLmBsHy7sMHfL2v8zRjxb' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
10Nrz6CTKEG0oLmBsHy7sMHfL2v8zRjxb
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('result_with_size.csv')
dataframe = pd.read_csv('result_with_size.csv')
```

In [0]:
```
dataframe.head(5)
```

Out[6]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...	310
1	1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	...	43
2	2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	...	224
3	3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	...	48
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	...	35

5 rows × 261 columns

In [0]:
```
print(dataframe.shape)
```

```
(10868, 261)
```

we got the byte features file with the class laebl we also get the .asm feature file and apppen them so that we can start the process and apply chisqaure tests on the these features.

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
link = 'https://drive.google.com/open?id=1XhfwsDj4G_R94oyW-QZJHAKfoilmKE1G' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1XhfwsDj4G_R94oyW-QZJHAKfoilmKE1G
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('asmoutputfile.csv')
dat= pd.read_csv('asmoutputfile.csv')
```

In [0]:
```
dat.head(5)
```

Out[12]:

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

5 rows × 52 columns

In [0]:
```
print(dat.shape)
```

```
(10868, 52)
```

In [0]:
```
dataframe.head(5)
```

Out[14]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	1
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...	310
1	1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	...	43
2	2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	...	224
3	3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	...	48
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	...	35

5 rows × 261 columns

In [0]:
```
result=pd.merge(dataframe,dat,on='ID',how='left')
result.head(5)
print(result.shape)
```

```
(10868, 312)
```

In [0]:
```
result.head(5)
```

Out[16]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	:dw
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...	4
1	1	01IsoiSMh5gxyDYTI4CB	39755	8337	7249	7186	8663	6844	8420	7589	...	1
2	2	01jsnpXSAlgw6aPeDxrU	93506	9542	2568	2438	8925	9330	9007	2342	...	
3	3	01kcPWA9K2BOxQeS5Rju	21091	1213	726	817	1257	625	550	523	...	
4	4	01SuzwMJEIXsK7A8dQbl	19764	710	302	433	559	410	262	249	...	1

5 rows × 312 columns

In [0]:
```
asm_y = result['Class']
asm_x = result.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

we have taken merged the features of asm files and the byte feature files

In [0]:
```
print(asm_x.shape)
print(asm_y.shape)
```

```
(10868, 307)
(10868,)
```

In [0]:
```
from sklearn.feature_selection import chi2
x_,y_=chi2(asm_x,asm_y)
```

In [0]:
```
print(x_)
print(y_)
```

```
[1.14341795e+05 3.96296388e+08 8.29424914e+07 2.11866508e+08
 1.21518834e+08 1.56646103e+08 3.77045925e+07 1.50660934e+08
 3.67865391e+07 3.98035682e+07 1.85680256e+07 2.05170518e+07
 5.09745245e+07 3.74615486e+07 1.36008327e+07 1.64122105e+07
 4.58867963e+07 1.19408013e+08 9.10233722e+07 1.19800971e+08
 8.91709202e+07 3.85225315e+07 1.95004632e+07 2.61291230e+07
 2.49322500e+07 4.51176763e+07 3.14994291e+07 1.76055421e+07
 2.32777739e+07 2.61497954e+07 1.64436549e+07 2.40118909e+07
 2.72054860e+07 8.72142804e+07 6.28374499e+07 6.88625460e+07
 6.90361203e+07 2.12679337e+07 1.02287734e+07 9.25638142e+06
 1.17846363e+07 9.55342397e+06 8.09213410e+06 8.56765717e+06
 9.13113731e+06 1.30361363e+07 8.85251561e+06 8.74861499e+06
 1.01578612e+07 9.64213901e+07 6.39678473e+07 7.63267913e+07
 9.63187771e+07 6.82188948e+06 1.07745550e+07 9.76914967e+06
 1.29361868e+07 1.52510927e+07 2.65284624e+07 7.55811432e+06
 2.82021999e+07 1.25944700e+07 1.23870312e+07 8.60819574e+06
 7.80445372e+06 2.29188963e+07 4.80024614e+06 1.07449512e+07
 2.61557473e+07 1.94970061e+07 2.45290898e+07 2.80350716e+07
 8.35130304e+06 2.39797988e+07 8.26110962e+06 2.21932538e+07
 1.55096393e+07 6.82202812e+06 2.29193601e+07 1.17169544e+07
 8.24101285e+06 2.10933456e+07 9.56477686e+06 9.40836477e+06
 1.99271126e+07 8.01192604e+06 2.53034958e+07 2.67434246e+07
 2.33396797e+07 1.08146024e+07 2.89749941e+07 8.74032658e+06
 7.29187784e+06 8.13167262e+06 2.55645589e+07 2.81805872e+07
 1.32293593e+07 6.53701558e+06 5.66472254e+06 1.29099366e+07
 8.85836725e+06 9.58186054e+06 8.37632523e+06 6.64552302e+06
 8.69360608e+06 2.53642176e+07 1.51844548e+07 2.43651561e+07
 1.28224436e+07 1.04244086e+07 1.33841188e+07 7.74185089e+06
 1.58272859e+07 7.68515776e+06 1.28486641e+07 1.12068929e+07
 1.35986565e+07 1.91236330e+07 2.89523067e+07 1.09235663e+07
 8.81238786e+06 8.64597076e+06 1.43645832e+07 1.14435136e+07
 8.98319404e+06 7.15000999e+06 1.83905894e+07 8.05189187e+06
 8.18685505e+06 1.67937230e+07 7.06479915e+06 7.81887838e+06
 1.77267055e+07 1.68795689e+07 1.75058522e+07 9.72172825e+06
 9.33386677e+06 1.79811326e+07 1.73483254e+07 8.83821988e+06
 4.09829863e+07 7.96613660e+06 1.55432523e+07 8.70380552e+06
 8.81554472e+06 1.11903584e+07 9.42852734e+06 8.80229330e+06
 9.76317777e+06 8.01558554e+06 9.00134829e+06 9.11655667e+06
 1.07158572e+07 8.84707967e+06 8.51767052e+06 9.74309002e+06
 9.15427575e+06 9.15269961e+06 1.01901481e+07 8.91515035e+06
 1.04389954e+07 7.39670301e+06 7.28951423e+06 1.04178388e+07
 8.23601320e+06 8.54827896e+06 8.88133834e+06 9.93605278e+06
 8.98101185e+06 8.26562167e+06 9.08747509e+06 8.95291399e+06
 8.89318604e+06 8.92941778e+06 1.02959482e+07 9.34119070e+06
 1.01727726e+07 8.51030036e+06 1.00736733e+07 8.90845629e+06
 1.01217630e+07 8.47033925e+06 1.05480795e+07 1.82001170e+07
 8.73714480e+06 8.14548029e+06 8.73526260e+06 9.17380861e+06
 9.05486517e+06 8.80747396e+06 9.18908094e+06 1.02625853e+07
 9.24478293e+06 1.68038410e+07 1.98113330e+07 6.62864929e+06
 1.96578210e+07 1.75221891e+07 9.06427107e+06 1.74496523e+07
 1.88674859e+07 7.51646764e+06 7.76110137e+06 8.46934065e+06
 9.13774947e+06 2.74379005e+08 9.15097392e+06 8.43286212e+06
 8.97877077e+06 7.92591348e+06 8.49485473e+06 8.13098386e+06
 8.95734670e+06 8.06514164e+06 8.46204012e+06 1.30251137e+07
```

```
 8.91800616e+06 7.41177613e+06 8.31496439e+06 9.46034742e+06
 8.23052773e+06 8.19607156e+06 8.28192861e+06 8.91598422e+06
 9.24401153e+06 6.85211688e+06 8.10458604e+06 8.81503689e+06
 1.06646688e+07 8.09354730e+06 8.95325842e+06 8.75849067e+06
 9.43313846e+06 1.84040234e+07 1.83638787e+07 8.58995314e+06
 1.84319835e+07 1.60317246e+07 8.72642409e+06 9.40663601e+06
 8.49073010e+06 2.07374311e+07 1.40694150e+07 1.47349286e+07
 8.17868811e+06 7.85353179e+06 8.31645187e+06 7.13819766e+06
 2.17285299e+07 1.73320953e+07 7.78946447e+06 9.96734628e+06
 7.89312915e+06 1.97629727e+07 9.07567490e+06 1.65653404e+07
 1.22228707e+08 1.00883416e+10 1.87010599e+04 4.44864126e+04
 2.58496943e+08 8.45270985e+06 1.10646461e+06 7.45641166e+08
 6.53196559e+06 4.86753063e+09 2.22098246e+06 1.96132646e+05
 3.84777302e+06 5.14992811e+04 6.60191463e+06 5.15694500e+07
 1.06316664e+06 3.64531442e+07 9.66690057e+06 5.82340440e+06
 4.70927529e+06 1.62090225e+07 1.59152115e+06 1.80036001e+06
 8.81593170e+05 5.49508369e+06 1.76015517e+07 2.03245662e+06
 1.15647041e+06 7.31834488e+05 1.02416325e+07 1.91017212e+07
 6.20977437e+05 5.99534974e+04 5.77696292e+04 3.15095188e+05
 9.97422384e+06 1.09828104e+07 1.26608045e+06 1.71855124e+07
 2.93222329e+07 5.49633486e+07 1.83313230e+07 2.91203393e+07
 1.97666495e+07 3.30460734e+07 2.29168887e+07 1.03479693e+04
 3.70105698e+04 1.01724872e+06 5.53452523e+05]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

chi2square documentation and returning the proability values of y

chi2 : array, shape = (n_features,) chi2 statistics of each feature. this retrned the statistics of each feature and also the probabiltiy values of each feature

pval : array, shape = (n_features,) p-values of each feature.

considering the 10 percent features of the total features 10 percent of 312 we are considering the top 32 features

In [0]:
```
import numpy as np
print(type(y_))
a=np.argsort(y_)[::-1][:32]
print(a)
```

```
<class 'numpy.ndarray'>
[306 104  97  98  99 100 101 102 103 105  76 106 107 108 109 110 111 112
  96  95  94  93  78  79  80  81  82  83  84  85  86  87]
```

In [0]:
```
#as suggested by dchad we gonna consider the models performance on 10 percent
#20 oercent and 30 percent of the total features
#for the model performance we gonna consider logloss,accuracy,confusion matrix
#we gon na see the model performance using the extra trees classifier
```

In [0]:
```
traindatafor10percentfeatures=asm_x.iloc[:,a]
```

In [0]:
```
traindatafor10percentfeatures.head(5)
```

Out[42]:

	eip	67	60	61	62	63	64	65	66	68	...	4d	4e	4f	50	
0	456	4375	3580	3344	3481	2945	3300	3447	2851	3730	...	2774	2732	3161	3955	45
1	227	801	711	952	733	648	1088	1591	1109	11323	...	8913	465	65918	9127	5
2	117	2883	2601	2553	2843	3055	2755	2712	2403	2748	...	2443	2464	2584	16251	27
3	29	499	545	814	399	538	1092	1041	941	1129	...	602	423	420	1108	5
4	76	365	1512	2134	484	585	368	568	347	385	...	845	290	389	562	4

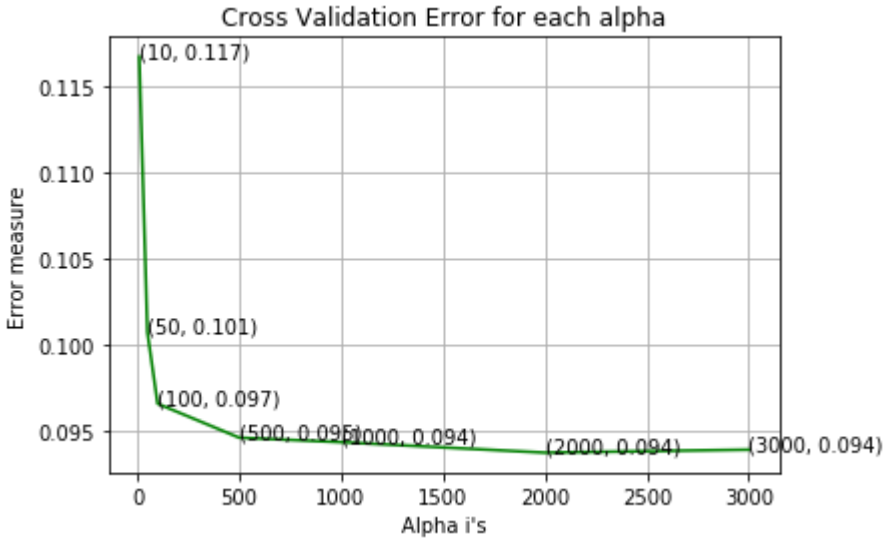
5 rows × 32 columns

In [0]:
```
from sklearn.model_selection import train_test_split
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(traindatafor1
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_tr
```

In [0]:
```
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import ExtraTreesClassifier
for i in alpha:
    print('current alpha value is ',i)
    r_cfl=ExtraTreesClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=ExtraTreesClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
current alpha value is  10
current alpha value is  50
current alpha value is  100
current alpha value is  500
current alpha value is  1000
current alpha value is  2000
current alpha value is  3000
log_loss for c =  10 is 0.11673606524838116
log_loss for c =  50 is 0.10075306929177441
log_loss for c =  100 is 0.09659916231276144
log_loss for c =  500 is 0.09462803478471248
```

```
log_loss for c =  1000 is 0.09436658540263229
log_loss for c =  2000 is 0.093757295762736
log_loss for c =  3000 is 0.09392943133367676
```




```
For values of best alpha =  2000 The train log loss is: 0.03354420236772946
For values of best alpha =  2000 The cross validation log loss is: 0.0937572957
62736
For values of best alpha =  2000 The test log loss is: 0.11686961272569582
```

considerinf the 20 percent features of total features

In [0]:
```
import numpy as np
print(type(y_))
a=np.argsort(y_)[::-1][:64]
print(a)
```

```
<class 'numpy.ndarray'>
[306 104  97  98  99 100 101 102 103 105  76 106 107 108 109 110 111 112
  96  95  94  93  78  79  80  81  82  83  84  85  86  87  88  89  90  91
  92 113 114 115 134 136 137 138 139 140 141 142 143 144 145 146 147 148
 149 150 135 133 116 132 117 118 119 120]
```

In [0]:
```
#as suggested by dchad we gonna consider the models performance on 10 percent
#20 oercent and 30 percent of the total features
#for the model performance we gonna consider logloss,accuracy,confusion matrix
#we gon na see the model performance using the extra trees classifier
```

In [0]:
```
traindatafor10percentfeatures=asm_x.iloc[:,a]
```

In [0]:
```
traindatafor10percentfeatures.head(5)
```

Out[30]:

	eip	67	60	61	62	63	64	65	66	68	...	94	95	86	84	7
0	456	4375	3580	3344	3481	2945	3300	3447	2851	3730	...	3920	2952	2796	3290	373
1	227	801	711	952	733	648	1088	1591	1109	11323	...	781	977	755	7153	6583
2	117	2883	2601	2553	2843	3055	2755	2712	2403	2748	...	2862	2510	2516	2425	263
3	29	499	545	814	399	538	1092	1041	941	1129	...	408	377	376	746	72
4	76	365	1512	2134	484	585	368	568	347	385	...	253	277	231	269	40

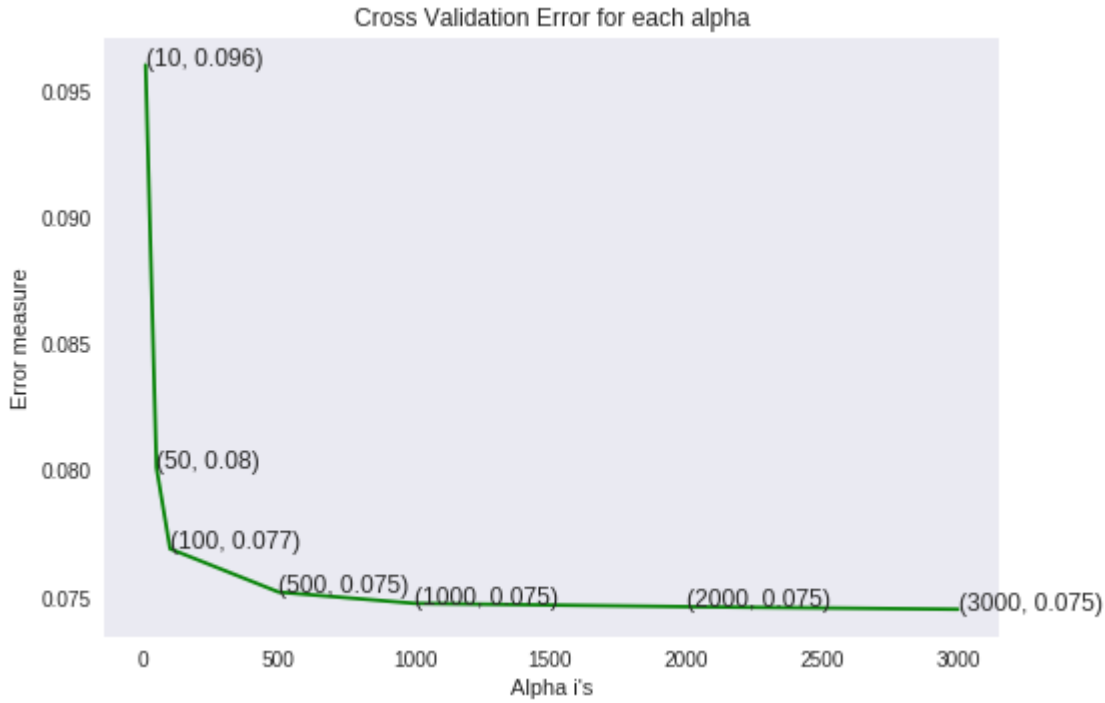
5 rows × 64 columns

In [0]:
```
from sklearn.model_selection import train_test_split
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(traindatafor1
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_tr
```

In [0]:
```
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import ExtraTreesClassifier
for i in alpha:
    print('current alpha value is ',i)
    r_cfl=ExtraTreesClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=ExtraTreesClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
current alpha value is  10
current alpha value is  50
current alpha value is  100
current alpha value is  500
current alpha value is  1000
current alpha value is  2000
current alpha value is  3000
log_loss for c =  10 is 0.09604268783702838
log_loss for c =  50 is 0.0801253872753246
log_loss for c =  100 is 0.07691579590699824
log_loss for c =  500 is 0.07520384984483364
```

```
log_loss for c =  1000 is 0.07475897100917324
log_loss for c =  2000 is 0.0746270519739272
log_loss for c =  3000 is 0.07452613334498641
```




```
For values of best alpha =  3000 The train log loss is: 0.02898564815591404
For values of best alpha =  3000 The cross validation log loss is: 0.0745261333
4498641
For values of best alpha =  3000 The test log loss is: 0.09296194748361782
```

In [0]:
```
import numpy as np
print(type(y_))
a=np.argsort(y_)[::-1][:100]
print(a)
```

```
<class 'numpy.ndarray'>
[306 104  97  98  99 100 101 102 103 105  76 106 107 108 109 110 111 112
  96  95  94  93  78  79  80  81  82  83  84  85  86  87  88  89  90  91
  92 113 114 115 134 136 137 138 139 140 141 142 143 144 145 146 147 148
 149 150 135 133 116 132 117 118 119 120 121 122 123 124 125 126 127 128
 129 130 131  77  75 152  27  20  21  22  23  24  25  26  28  74  29  30
  31  32  33  34  35  19  18  17  16   1]
```

In [0]:
```
#as suggested by dchad we gonna consider the models performance on 10 percent
#20 oercent and 30 percent of the total features
#for the model performance we gonna consider logloss,accuracy,confusion matrix
#we gon na see the model performance using the extra trees classifier
```

considering the 30 percent of the total features

In [0]:
```
traindatafor10percentfeatures=asm_x.iloc[:,a]
```

In [0]:
```
traindatafor10percentfeatures.head(5)
```

Out[47]:

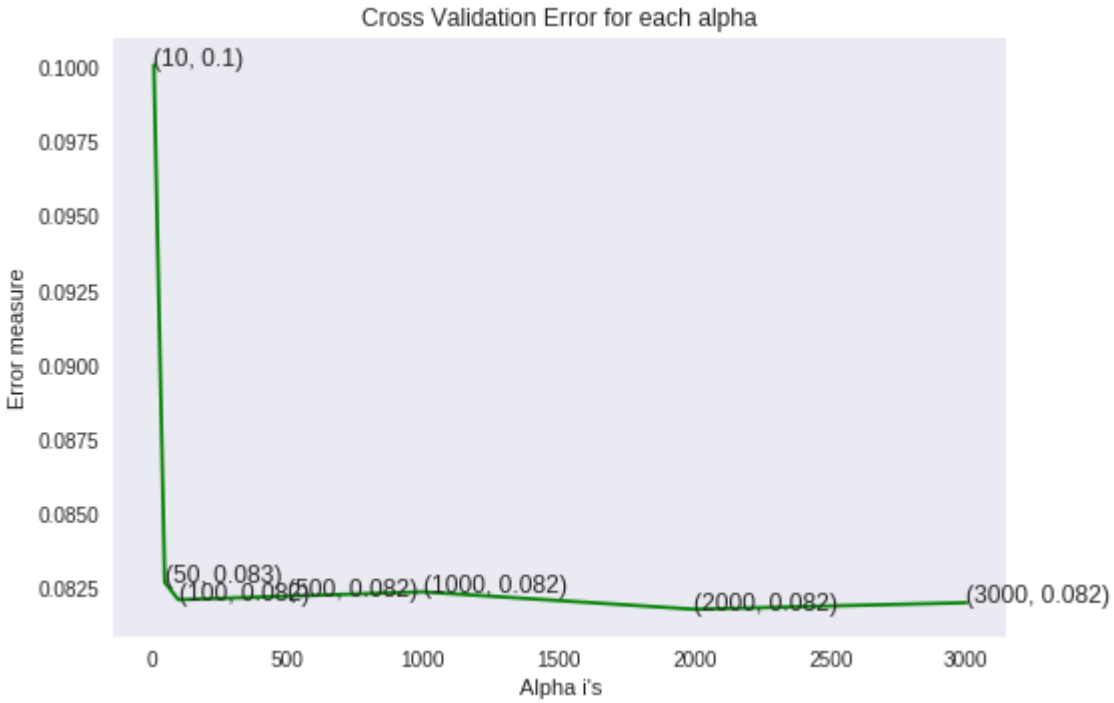
	Unnamed: 0	ebx	7	64
0	0	587	3201	3300
1	1	905	7589	1088
2	2	451	2342	2755
3	3	43	523	1092
4	4	1546	249	368

In [0]:
```
from sklearn.model_selection import train_test_split
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(traindatafor1(
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_tr
```

In [0]:
```
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import ExtraTreesClassifier
for i in alpha:
    print('current alpha value is ',i)
    r_cfl=ExtraTreesClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=ExtraTreesClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
current alpha value is  10
current alpha value is  50
current alpha value is  100
current alpha value is  500
current alpha value is  1000
current alpha value is  2000
current alpha value is  3000
log_loss for c =  10 is 0.10007166534501818
log_loss for c =  50 is 0.08272449199994654
log_loss for c =  100 is 0.08214194884276566
log_loss for c =  500 is 0.08226022793142136
```

```
log_loss for c =  1000 is 0.08240476320614683
log_loss for c =  2000 is 0.08182134329564608
log_loss for c =  3000 is 0.0820416198197262
```




```
For values of best alpha =  2000 The train log loss is: 0.02871133773648726
For values of best alpha =  2000 The cross validation log loss is: 0.0818213432
9564608
For values of best alpha =  2000 The test log loss is: 0.08349945842357324
```

considering the all the features of the model and checking the performance metrics

In [0]:
```
import numpy as np
print(type(y_))
a=np.argsort(y_)[::100]
print(a)
```

```
<class 'numpy.ndarray'>
[  0 301   8 101]
```

In [0]:
```
#as suggested by dchad we gonna consider the models performance on 10 percent
#20 oercent and 30 percent of the total features
#for the model performance we gonna consider logloss,accuracy,confusion matrix
#we gon na see the model performance using the extra trees classifier
```

In [0]:
```
traindatafor10percentfeatures=asm_x.iloc[:,a]
```

In [0]:
```
traindatafor10percentfeatures.head(5)
```

Out[40]:

	Unnamed: 0	cf	ce	cd	cc	cb	ca	c9	c8	c7	...	68	66	65	
0	0	3401	3347	3068	3105	2607	3103	3753	3222	4039	...	3730	2851	3447	33
1	1	613	607	480	1359	520	778	1127	662	7169	...	11323	1109	1591	10
2	2	2588	2796	2457	2641	174342	2720	2838	2685	2719	...	2748	2403	2712	27
3	3	344	363	345	433	359	370	566	609	531	...	1129	941	1041	10
4	4	188	245	237	285	228	230	212	238	375	...	385	347	568	3

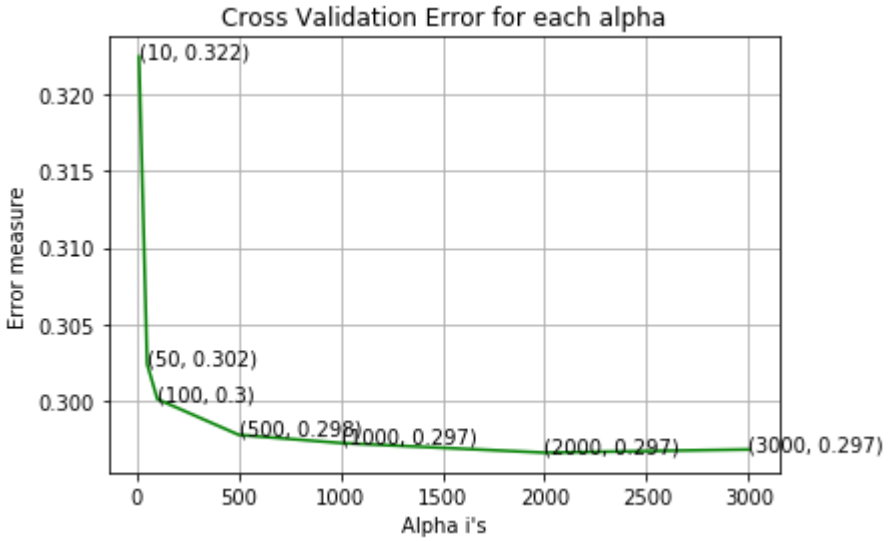
5 rows × 307 columns

In [0]:
```
from sklearn.model_selection import train_test_split
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(traindatafor1
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_tr
```

In [0]:
```
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import ExtraTreesClassifier
for i in alpha:
    print('current alpha value is ',i)
    r_cfl=ExtraTreesClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=ExtraTreesClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
current alpha value is  10
current alpha value is  50
current alpha value is  100
current alpha value is  500
current alpha value is  1000
current alpha value is  2000
current alpha value is  3000
log_loss for c =  10 is 0.322454305064343
log_loss for c =  50 is 0.3023925443469619
log_loss for c =  100 is 0.3001046106544286
log_loss for c =  500 is 0.2977680388248046
```

```
log_loss for c =  1000 is 0.2972394059439264
log_loss for c =  2000 is 0.2966102882870639
log_loss for c =  3000 is 0.29681675133522983
```




```
For values of best alpha =  2000 The train log loss is: 0.08567315215010517
For values of best alpha =  2000 The cross validation log loss is: 0.2966102882
870639
For values of best alpha =  2000 The test log loss is: 0.2873658900334469
```

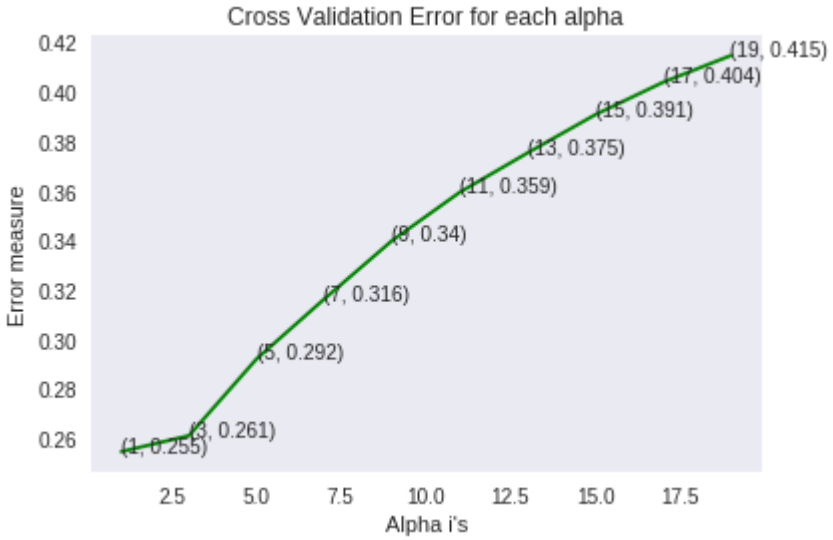
In [0]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
%matplotlib inline
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are p

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


    B =(C/C.sum(axis=0))

    plt.figure(figsize=(15,8))
    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True,  fmt=".3f", xticklabels=labels, yticklabels=labels
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*50, "Precision matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(B, annot=True, 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,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))
```

using knn

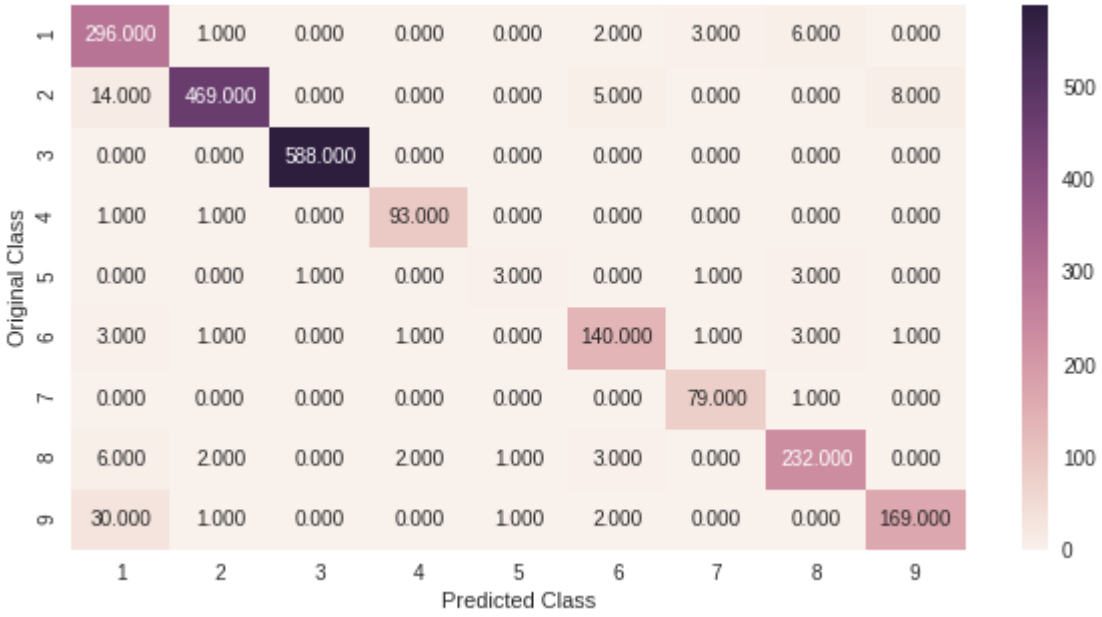
In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [0]:
```
alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

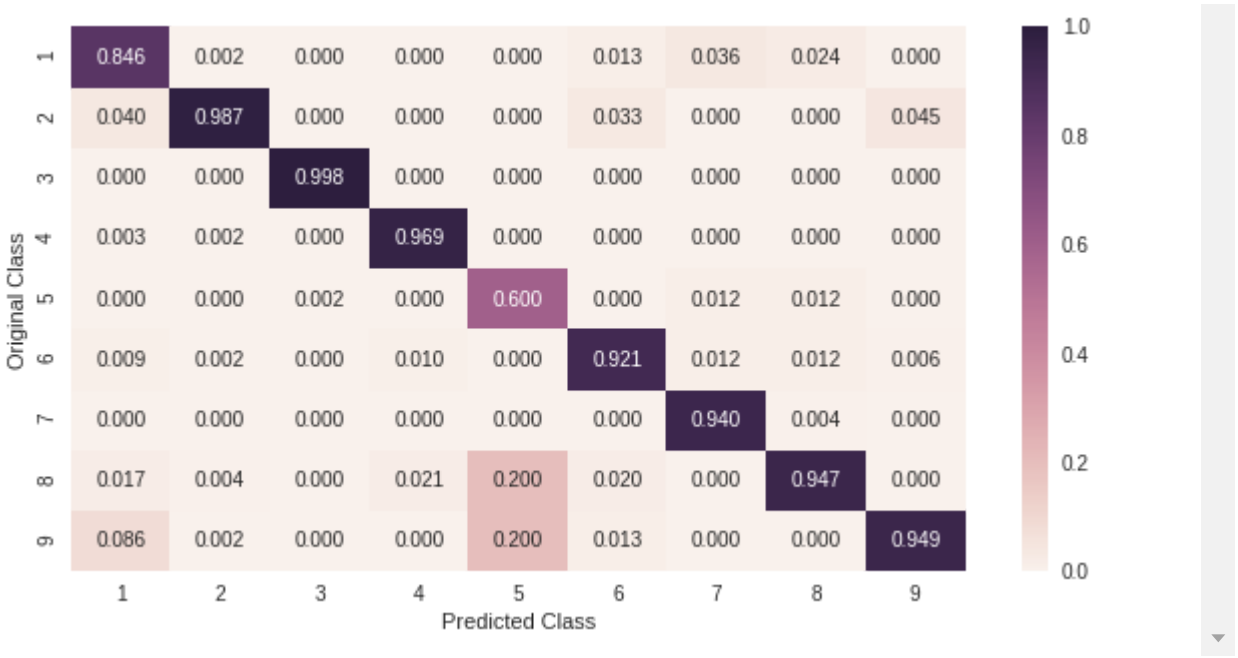
```
log_loss for k =  1 is 0.25464462500299606
log_loss for k =  3 is 0.2611059232576465
log_loss for k =  5 is 0.2921579026264672
log_loss for k =  7 is 0.31635093630130773
log_loss for k =  9 is 0.3399155429599249
log_loss for k =  11 is 0.3594522901540095
log_loss for k =  13 is 0.37533661571554444
log_loss for k =  15 is 0.39076336659271643
log_loss for k =  17 is 0.4038524146779985
log_loss for k =  19 is 0.414643193359781
```



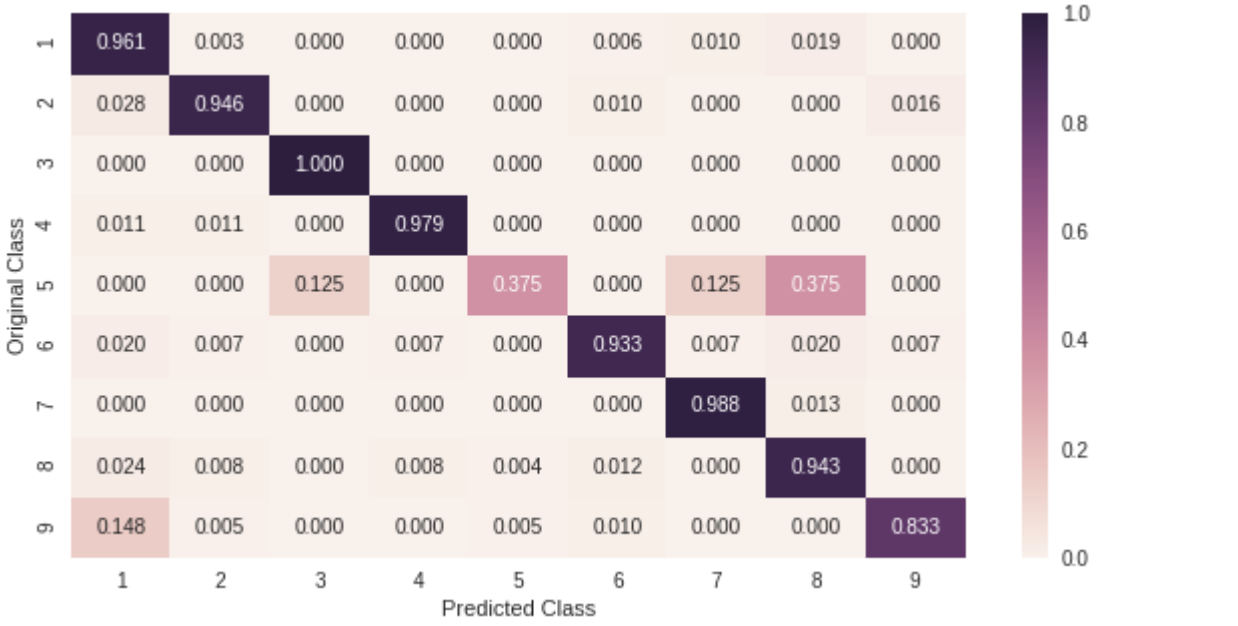

```
log loss for train data 0.08952853188535019
log loss for cv data 0.25464462500299606
log loss for test data 0.24004561397708524
Number of misclassified points  4.8298068077276906
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```




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




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

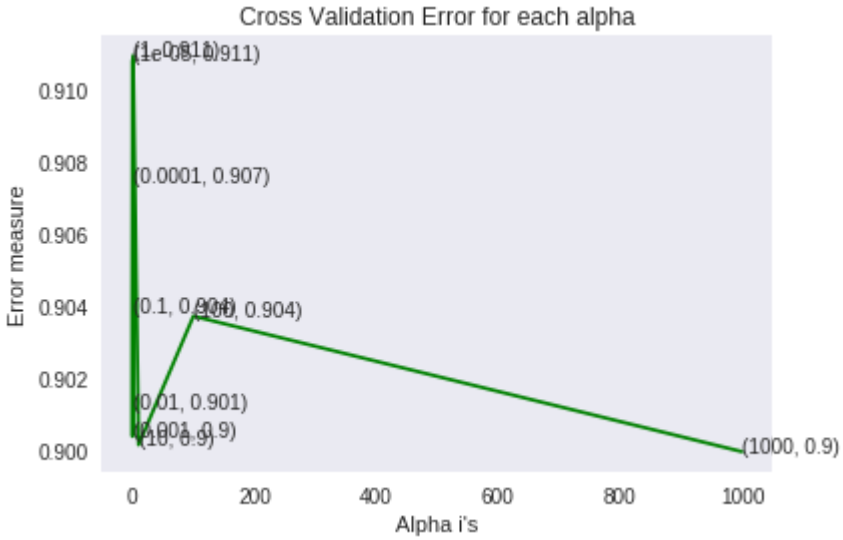



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

performing the logistic regression

In [0]:
```
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.clas

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

best_alpha = np.argmin(cv_log_error_array)

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

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balan
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=logisti
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.cla
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

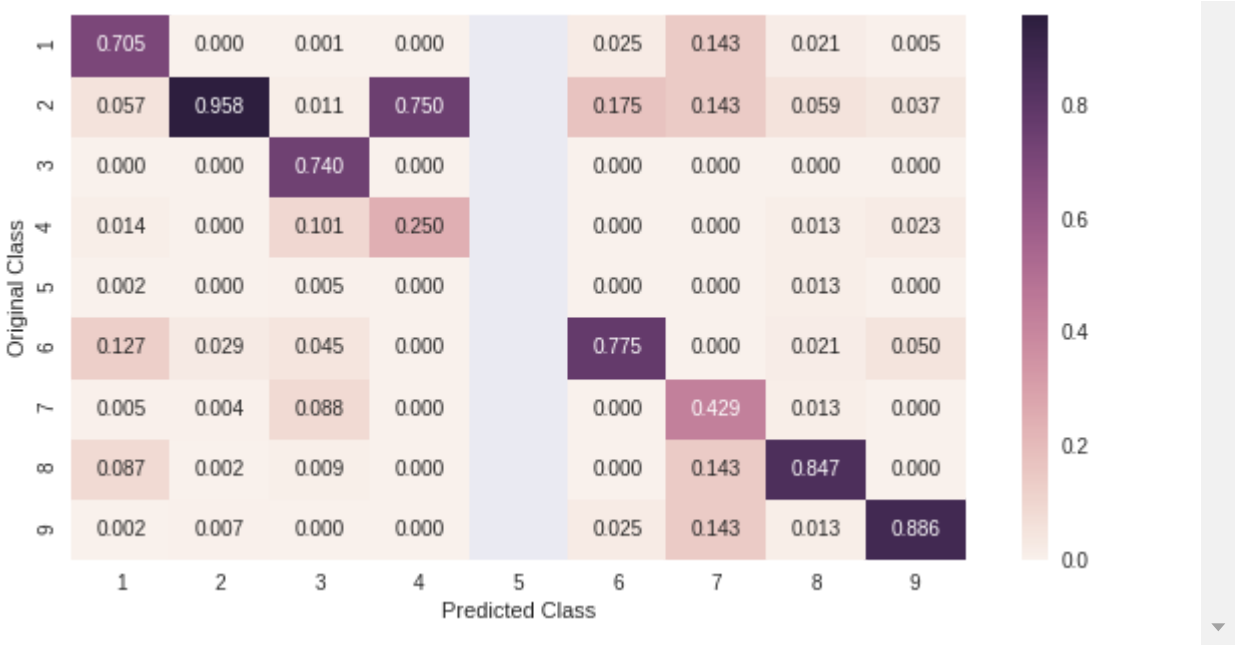
```
log_loss for c =  1e-05 is 0.9108268313408105
log_loss for c =  0.0001 is 0.9074469992749733
log_loss for c =  0.001 is 0.9003858534116725
log_loss for c =  0.01 is 0.901146026225207
log_loss for c =  0.1 is 0.9038174284100232
log_loss for c =  1 is 0.9109609189503458
log_loss for c =  10 is 0.9001506529005922
log_loss for c =  100 is 0.9037259486954758
log_loss for c =  1000 is 0.8999618815449928
```



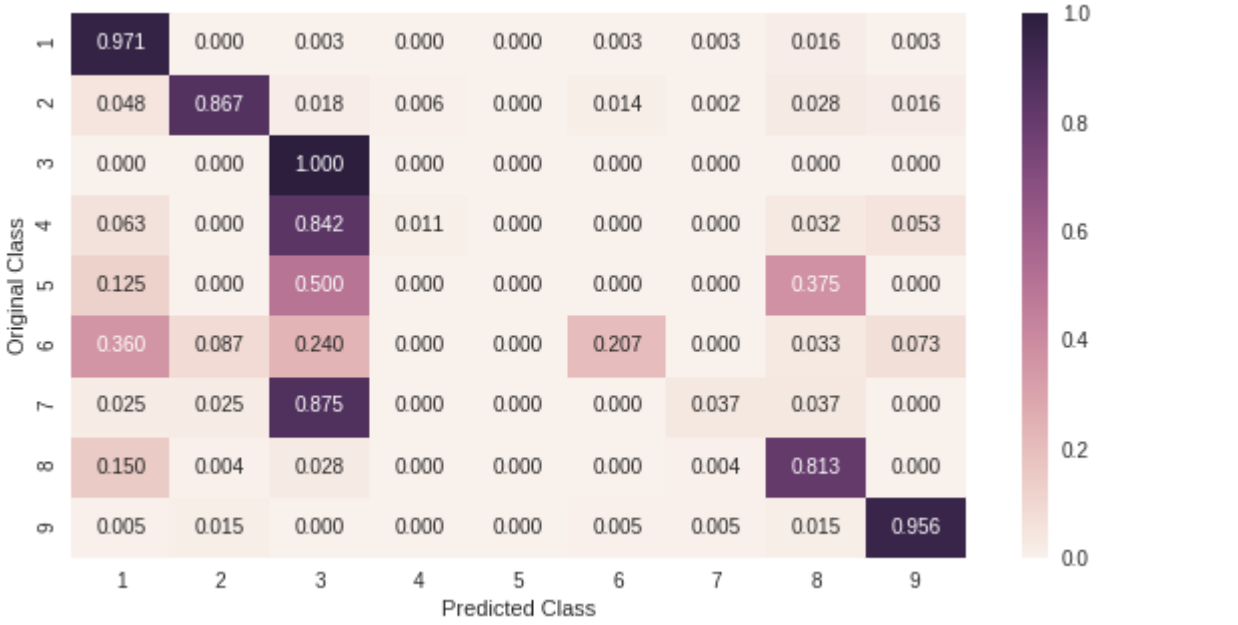

```
log loss for train data 0.8354959617323964
log loss for cv data 0.8999618815449928
log loss for test data 0.8868849826934874
Number of misclassified points  19.687212511499542
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```

```
<Figure size 1080x576 with 0 Axes>
```


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




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



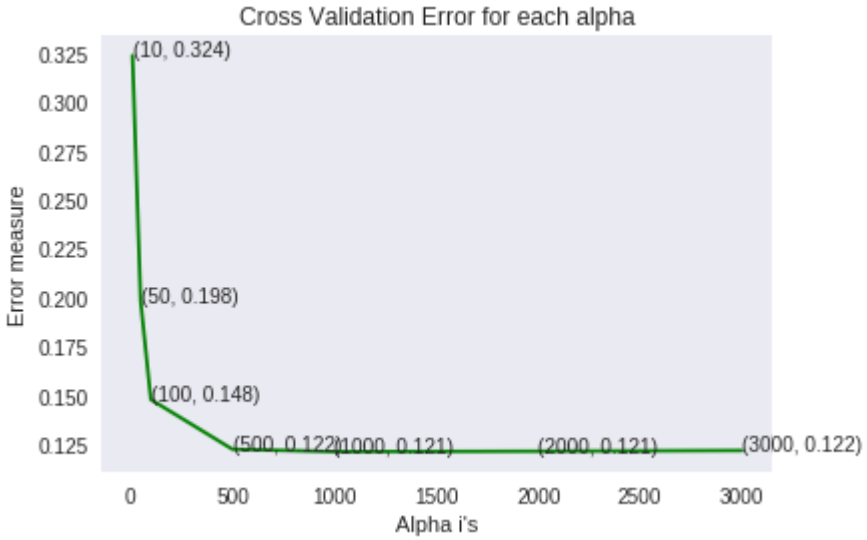

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [0]:
```
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
```

performing the xgboost classfier

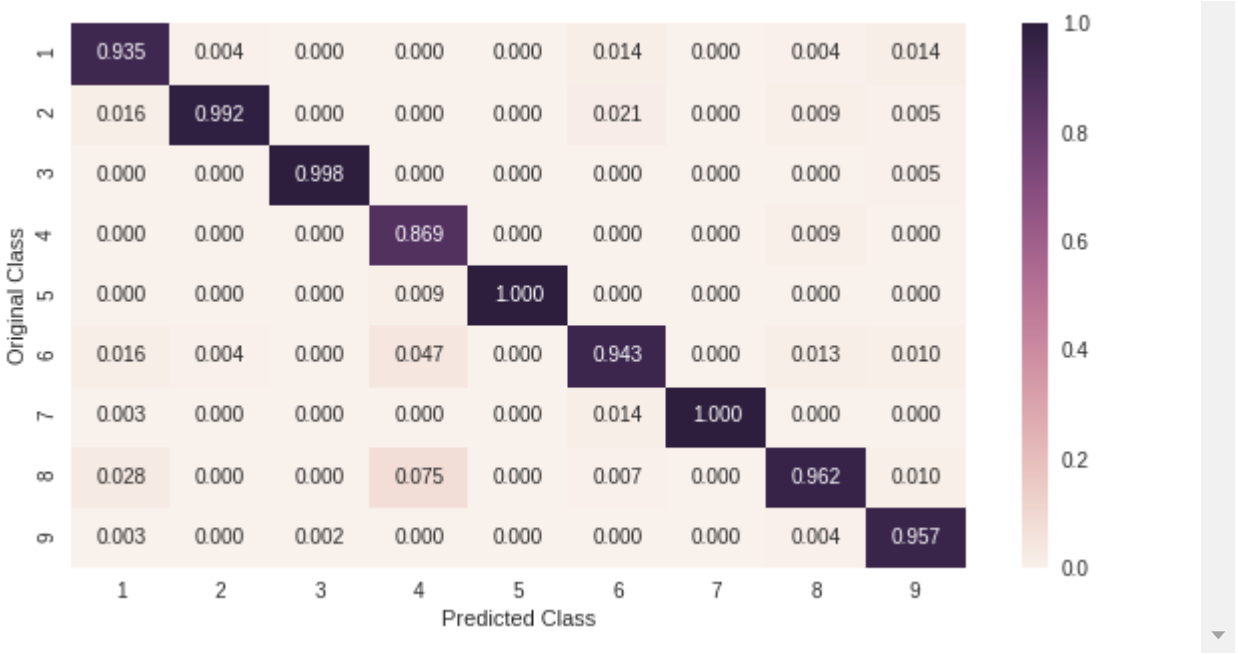
In [0]:
```
%matplotlib inline
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_cfl.classes_

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


best_alpha = np.argmin(cv_log_error_array)

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

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

predict_y = sig_clf.predict_proba(X_train_asm)

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

```
log_loss for c =  10 is 0.3235558845671812
log_loss for c =  50 is 0.19808666668071795
log_loss for c =  100 is 0.14781031593982852
log_loss for c =  500 is 0.12240207360914032
log_loss for c =  1000 is 0.12115657918298504
log_loss for c =  2000 is 0.12138005117726003
log_loss for c =  3000 is 0.12173681249500563
```

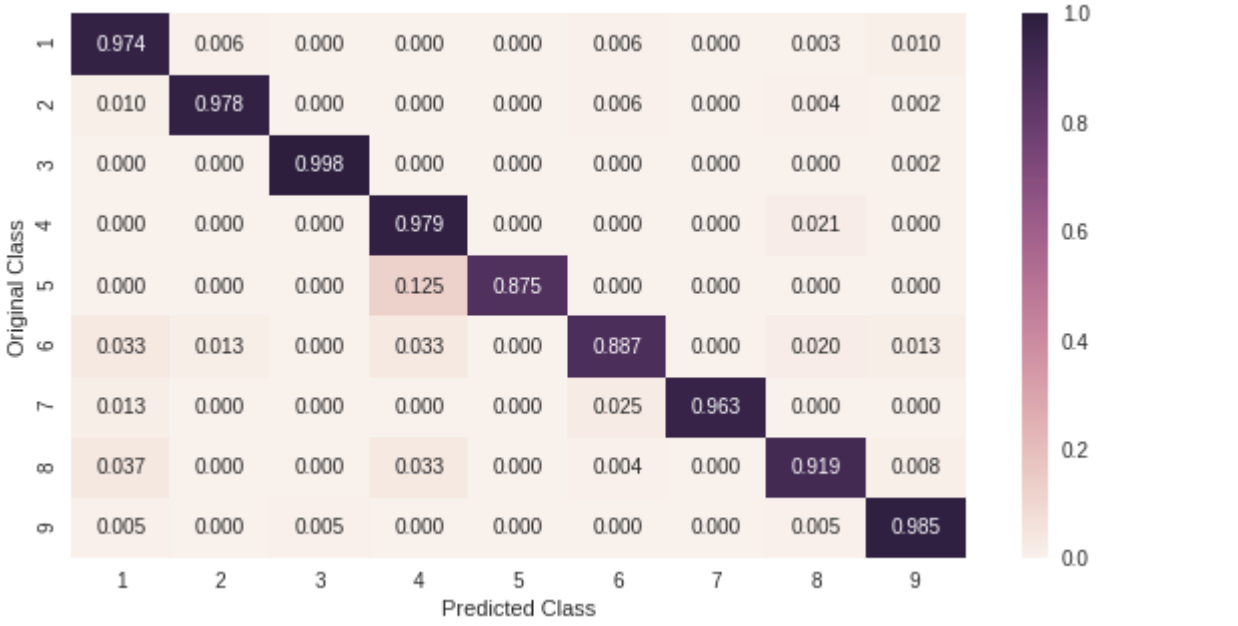



```
For values of best alpha =  1000 The train log loss is: 0.03824069498262199
For values of best alpha =  1000 The cross validation log loss is: 0.1211565791
8298504
For values of best alpha =  1000 The test log loss is: 0.13977494894836945
Number of misclassified points  3.035878564857406
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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

In [0]:
```
x_cfl=XGBClassifier()

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

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed: 11.2min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 17.2min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 23.9min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 46.4min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 64.8min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 75.3min finished
```

Out[46]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', 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], 'co
lsample_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='warn', scoring=None, verbose=10)
```

In [0]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.2, 'co
lsample_bytree': 0.3}
```

In [0]:
```
x_cfl=XGBClassifier(n_estimators=100,subsample=1,learning_rate=0.2,colsample_bytr
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

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

```
train loss 0.04210134627502086
cv loss 0.12931697122069016
test loss 0.13685257159475508
```

In [0]:
```
dta = [['knn',1,0.08,0.25,0.24],['logistic regression',1000,0.83,0.9,0.88],['xgb
aa=pd.DataFrame(dta, columns=['model',"best hyperparameter",'train logloss','cv l
aa
```

Out[75]:

	model	best hyperparameter	train logloss	cv logloss	test logloss
0	knn	1	0.08	0.25	0.24
1	logistic regression	1000	0.83	0.90	0.88
2	xgb classifier	1000(number of estimators)	0.03	0.12	0.13

USING THE IMAGE FEATURES WITH THE FEATURES OF .BYTE FILES AND .ASM FILES

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
link = 'https://drive.google.com/open?id=1gYNngh_VHdqW-VtQsPIYSv0_88k1b_JD' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
1gYNngh_VHdqW-VtQsPIYSv0_88k1b_JD
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('8324-test-image-featuresout-asm.csv')
dataframe1 = pd.read_csv('8324-test-image-featuresout-asm.csv')
```

In [0]:
```
dataframe1.head(2)
```

Out[27]:

	filename	ASM_0	ASM_1	ASM_2	ASM_3	ASM_4	ASM_5	ASM_6	ASM_7	ASM
0	01azqd4InC7m9JpocGv5	72	69	65	68	69	82	58	48	
1	01IsoiSMh5gxyDYTl4CB	46	116	101	120	116	58	48	48	

2 rows × 1001 columns

In [0]:
```
dataframe1['ID']=dataframe1['filename']
dataframe1=dataframe1.drop(columns=['filename'])
dataframe1.head(5)
```

Out[28]:

	ASM_0	ASM_1	ASM_2	ASM_3	ASM_4	ASM_5	ASM_6	ASM_7	ASM_8	ASM_9	...	ASM_9
0	72	69	65	68	69	82	58	48	48	52	...	1
1	46	116	101	120	116	58	48	48	52	48	...	
2	72	69	65	68	69	82	58	48	48	52	...	1
3	72	69	65	68	69	82	58	49	48	48	...	
4	72	69	65	68	69	82	58	48	48	52	...	1

5 rows × 1001 columns

In [0]:
```
result=pd.merge(dataframe,dataframe1,on='ID',how='left')
result.head(5)
print(result.shape)
```

```
(10868, 1261)
```

In [0]:
```
asm_y = result['Class']
asm_x = result.drop(['ID','Class'], axis=1)
```

#we have generated the data with the image features of .asm files asn we are using the byte features and .asm file features

we are doing the preprocessing step of data standardisation and apply the models.

we are applying the models

- knn
- naive bayes
- logistic regression
- random forest

- xgboost with random search cv

In [0]:
```
from sklearn.preprocessing import StandardScaler
scale=StandardScaler()
asm_x=scale.fit_transform(asm_x)
```

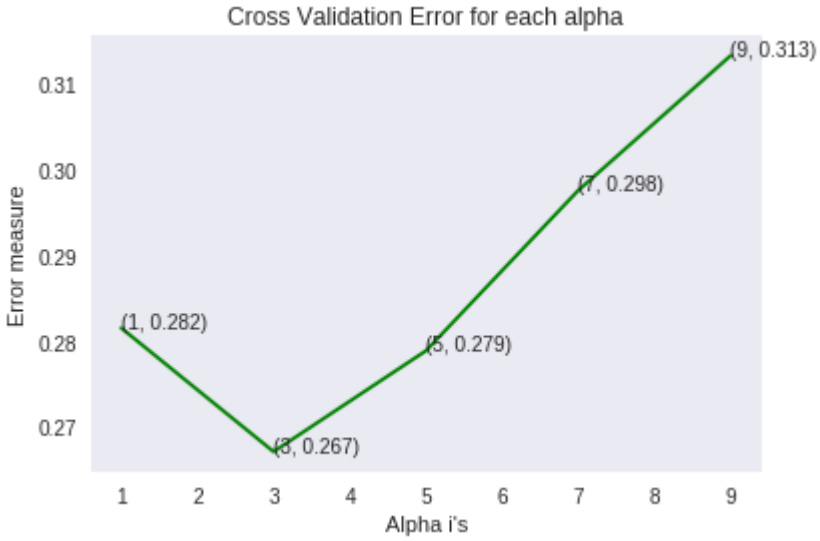
```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:645: DataC
onversionWarning: Data with input dtype int64, float64 were all converted to fl
oat64 by StandardScaler.
  return self.partial_fit(X, y)
/usr/local/lib/python3.6/dist-packages/sklearn/base.py:464: DataConversionWarni
ng: Data with input dtype int64, float64 were all converted to float64 by Stand
ardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [0]:
```
from sklearn.model_selection import train_test_split
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_tr
```

KNN(k nearest neighbors)

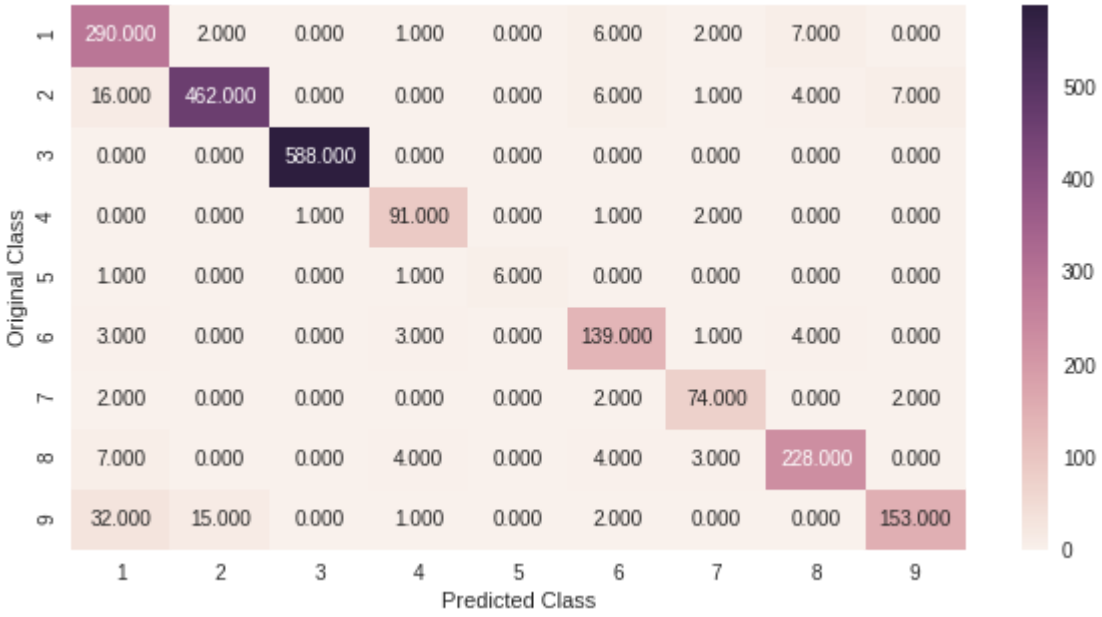
In [0]:
```
alpha = [x for x in range(1, 10,2)]
cv_log_error_array=[]
for i in alpha:
    print(i)
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_

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

best_alpha = np.argmin(cv_log_error_array)

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

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


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

```
1
3
5
7
9
log_loss for k =  1 is 0.2815193019820606
log_loss for k =  3 is 0.2671040855888815
log_loss for k =  5 is 0.27886597408651553
log_loss for k =  7 is 0.29756526031536856
log_loss for k =  9 is 0.31326949294254613
```

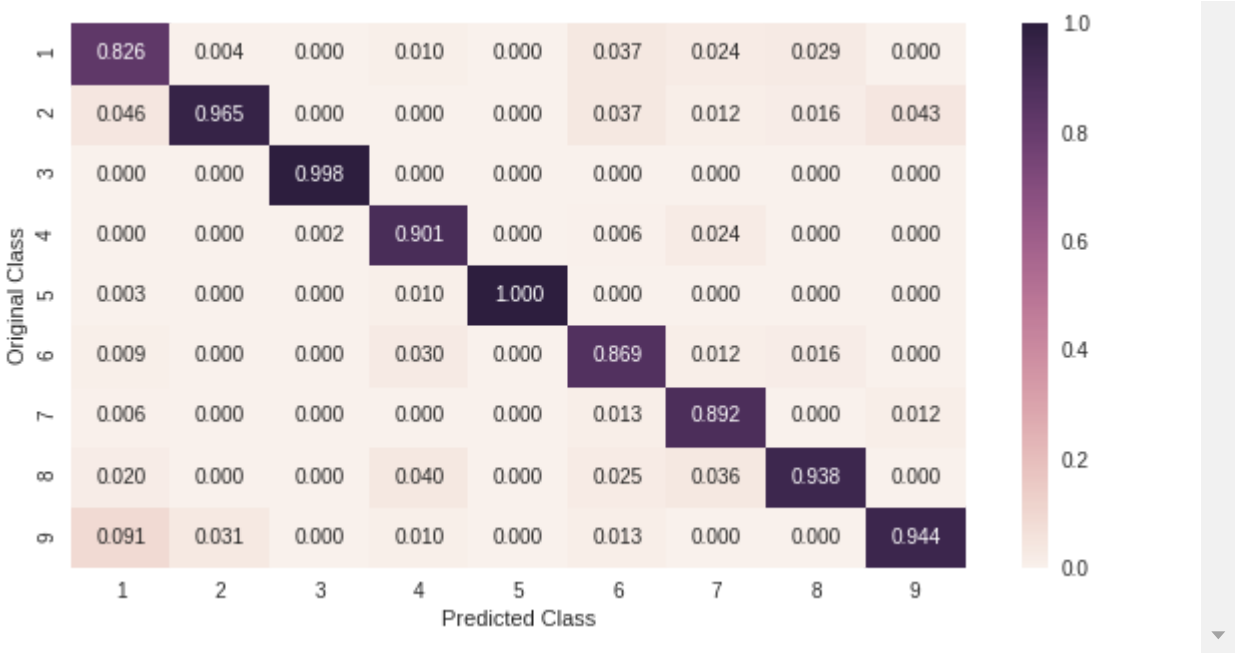



```
log loss for train data 0.1466472782761681
log loss for cv data 0.2671040855888815
log loss for test data 0.2629322790620862
Number of misclassified points  6.5777368905243785
---------------------------------------------------- Confusion matrix -----------
----------------------------------------
```

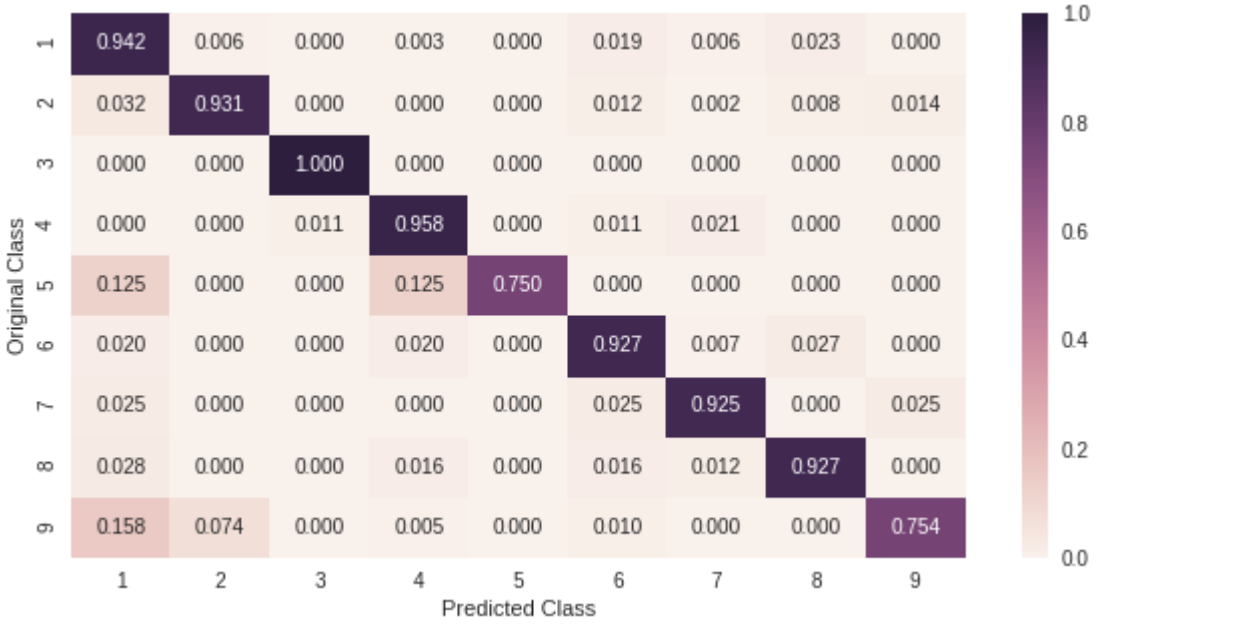
<Figure size 1080x576 with 0 Axes>




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




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

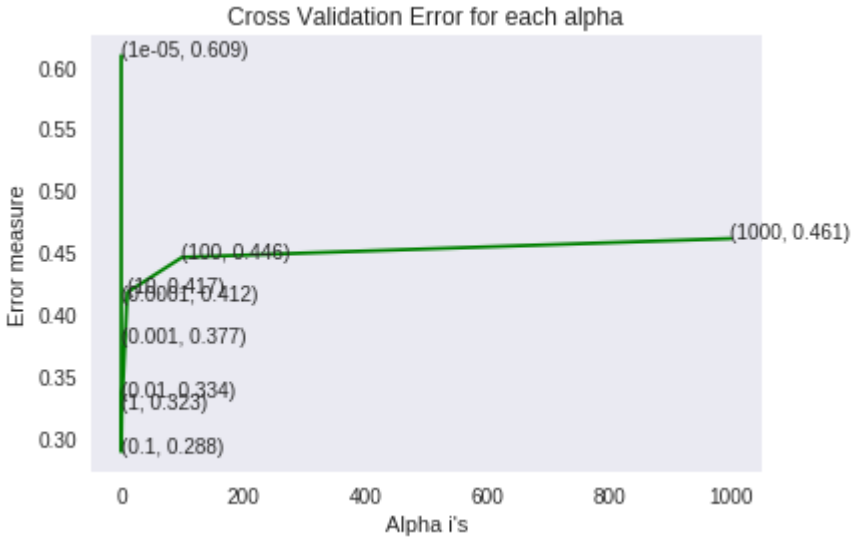



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

#LOGISTIC REGRESSION

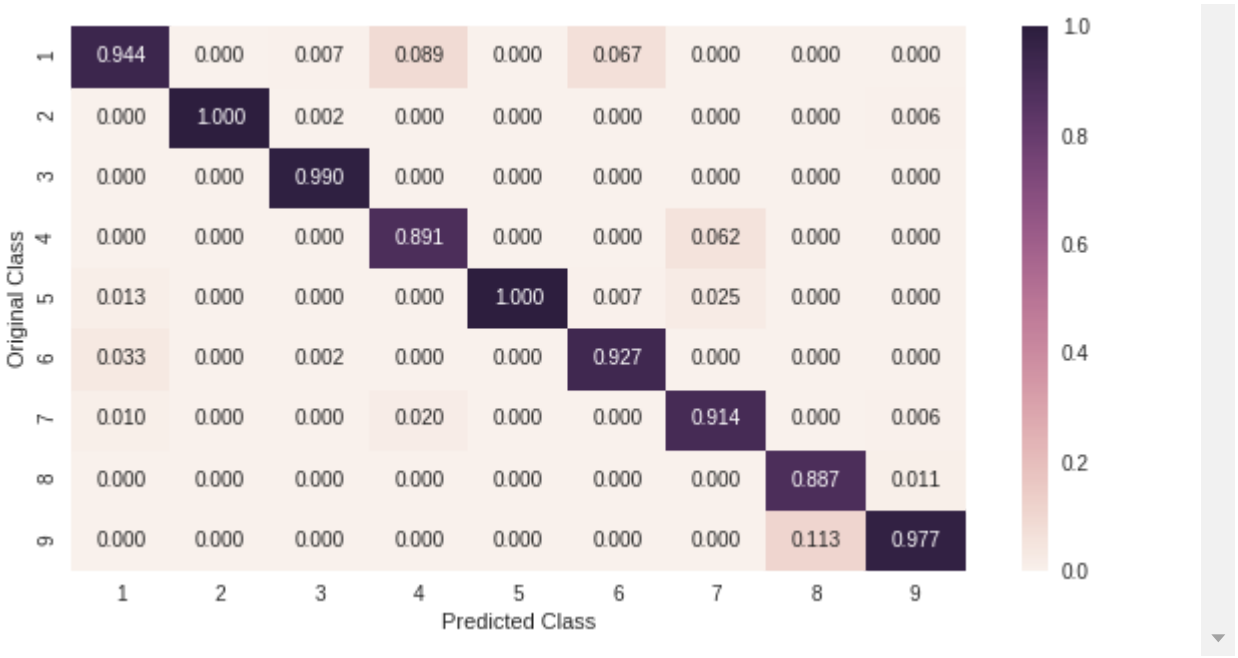
In [0]:
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, 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.clas

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

best_alpha = np.argmin(cv_log_error_array)

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

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balan
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=logisti
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.cla
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  1e-05 is 0.6090034322994382
log_loss for c =  0.0001 is 0.41189347494321016
log_loss for c =  0.001 is 0.3774405398785833
log_loss for c =  0.01 is 0.33410836190450405
log_loss for c =  0.1 is 0.28843301372023583
log_loss for c =  1 is 0.32343885524222354
log_loss for c =  10 is 0.4174086805596062
log_loss for c =  100 is 0.44578402193842614
log_loss for c =  1000 is 0.46095090847608494
```

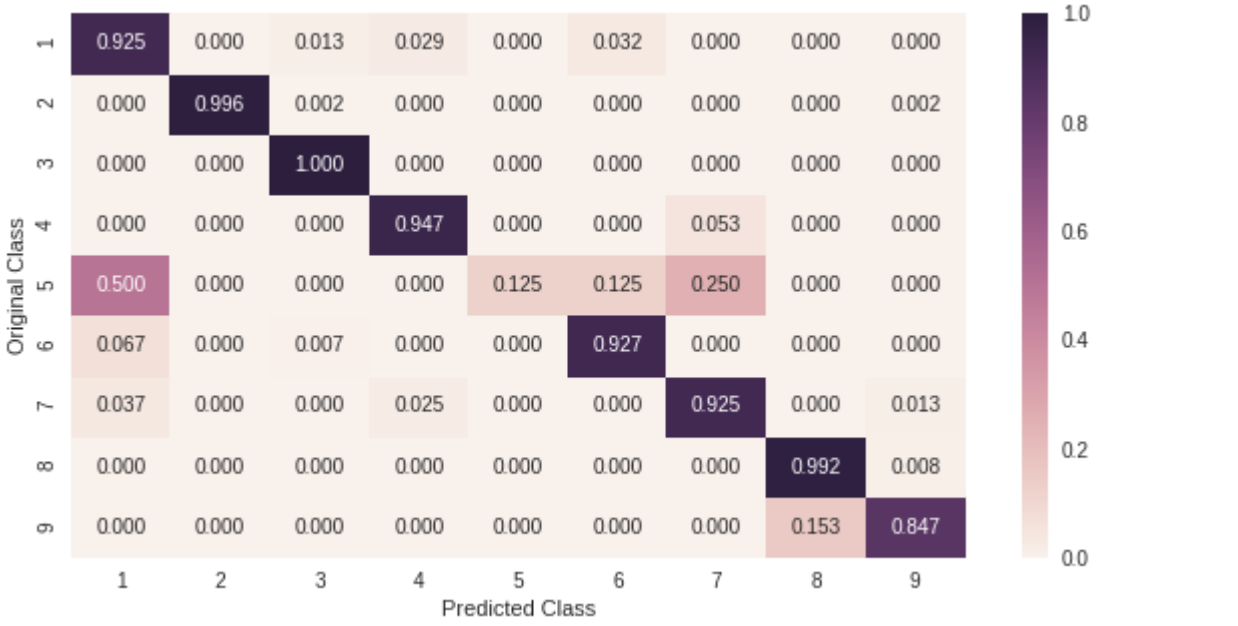



```
log loss for train data 0.2640092989362992
log loss for cv data 0.28843301372023583
log loss for test data 0.282203111170429
Number of misclassified points  4.001839926402944
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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

multinomial naive bayes does not work on negative data we used normalizer instead of using the standard scaler

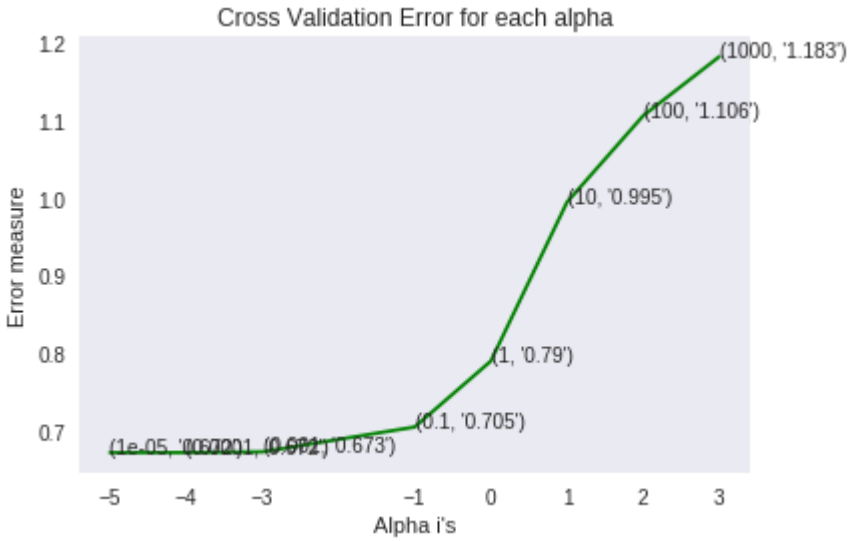
In [0]:
```
from sklearn.preprocessing import Normalizer
norm=Normalizer()
asm_x=norm.fit_transform(asm_x)
```

In [0]:
```
from sklearn.model_selection import train_test_split
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_tr
```

In [0]:
```
from sklearn.naive_bayes import MultinomialNB
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_asm,y_train_asm)
    sig_clf_probs = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, sig_clf_probs, labels=clf.classe
    # to avoid rounding error while multiplying probabilites we use log-probabili
    print("Log Loss :",log_loss(y_cv_asm, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(clf, 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:",
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

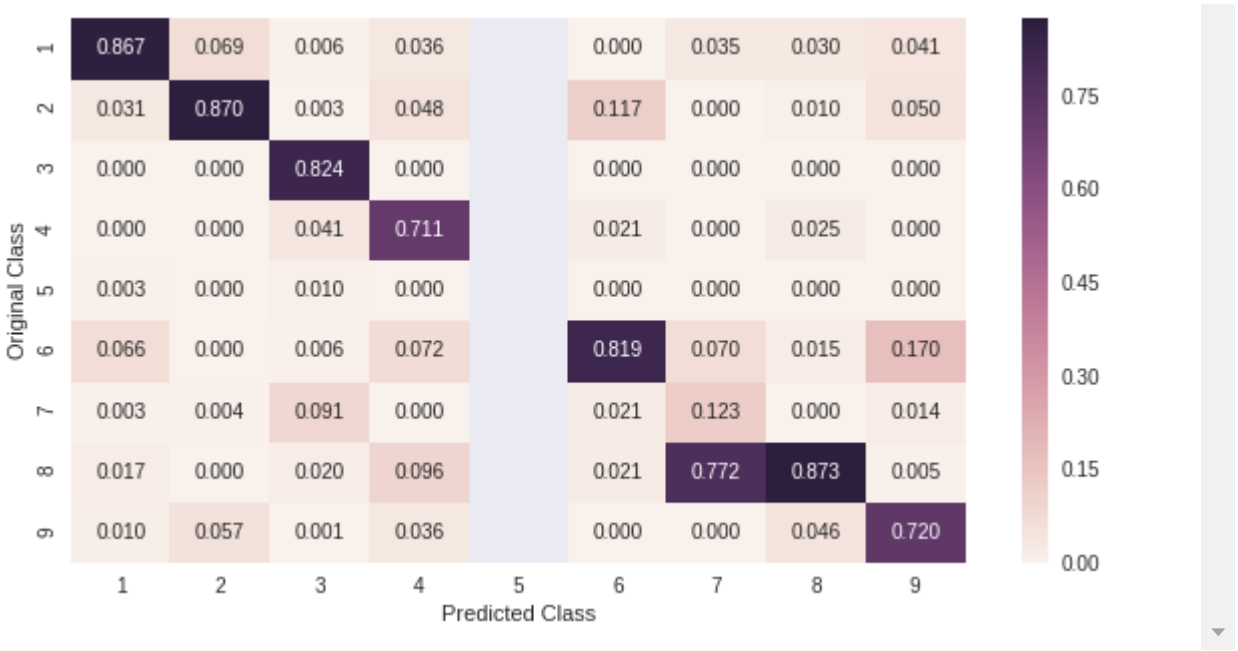
```
for alpha = 1e-05
Log Loss : 0.6715329427802057
for alpha = 0.0001
Log Loss : 0.6716322192344913
for alpha = 0.001
Log Loss : 0.6725542641805241
for alpha = 0.1
Log Loss : 0.7046067230160736
for alpha = 1
Log Loss : 0.7896935980535442
for alpha = 10
Log Loss : 0.9947952571443085
for alpha = 100
Log Loss : 1.1063123268802026
```

```
for alpha = 1000
Log Loss : 1.182612374427028
```

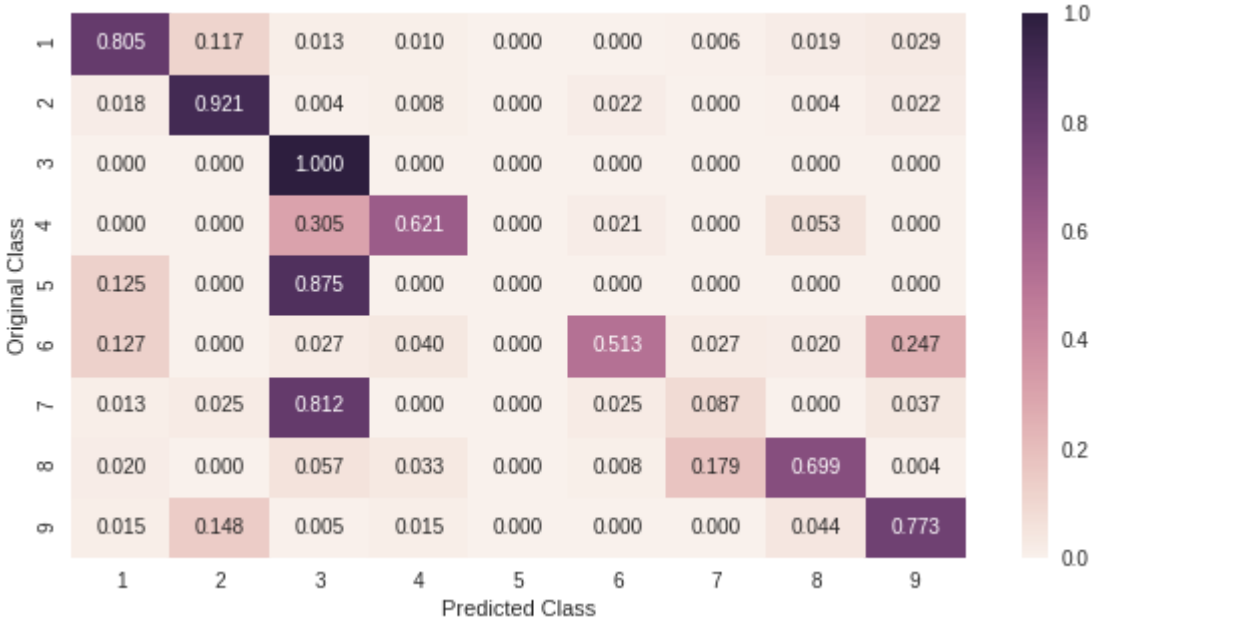



```
For values of best alpha =  1e-05 The train log loss is: 0.6639536954595033
For values of best alpha =  1e-05 The cross validation log loss is: 0.671532942
7802057
For values of best alpha =  1e-05 The test log loss is: 0.6789986580234393
Number of misclassified points  18.813247470101196
---------------------------------------------------- Confusion matrix -----------
----------------------------------------

<Figure size 1080x576 with 0 Axes>
```


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




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

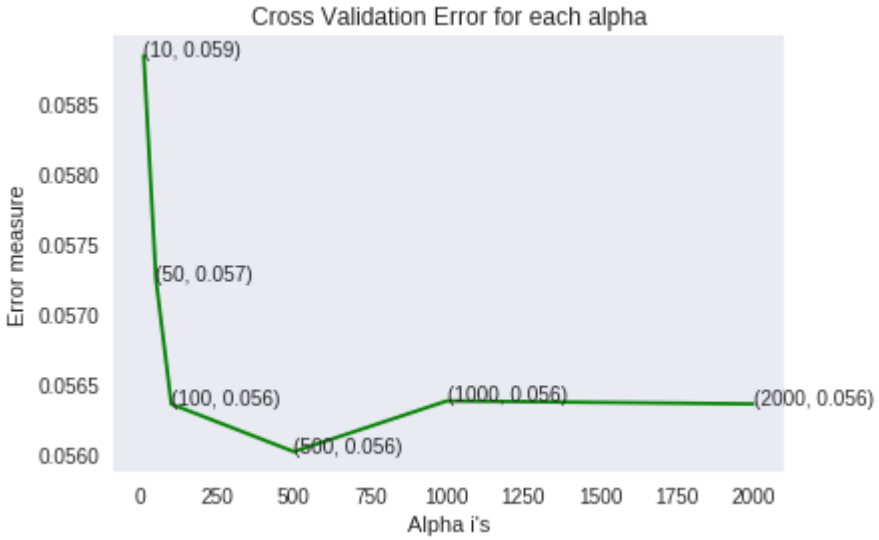



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

Random Forest Classifier

In [0]:
```
alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    print('current value of alpha is',i)
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm,y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_

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


best_alpha = np.argmin(cv_log_error_array)

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


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_job
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 ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
current value of alpha is 10
current value of alpha is 50
current value of alpha is 100
current value of alpha is 500
current value of alpha is 1000
current value of alpha is 2000
log_loss for c =  10 is 0.058836192465336654
log_loss for c =  50 is 0.057245687441351355
log_loss for c =  100 is 0.05635909048492949
log_loss for c =  500 is 0.056020994169041605
log_loss for c =  1000 is 0.056381575085719636
log_loss for c =  2000 is 0.05636034947761574
```




```
For values of best alpha =  500 The train log loss is: 0.020302926521527122
For values of best alpha =  500 The cross validation log loss is: 0.05602099416
9041605
For values of best alpha =  500 The test log loss is: 0.05165091509556082
```

XGB CLASSIFIER WITH RANDOM SEARCH CV

In [0]:
```
x_cfl=XGBClassifier()
from sklearn.model_selection import GridSearchCV

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000,3000],
     'max_depth':[3,5,10,20,30],
    '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,scoring='neg_log_lo
random_cfl.fit(X_train_asm,y_train_asm)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 38.0min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 106.5min finished
```

Out[30]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_b
ylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', 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, 3000], 'max_depth': [3, 5, 1
0, 20, 30], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'subsample': [0.1, 0.3, 0.
5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring='neg_log_loss', verbose=5)
```

In [0]:
```
print (random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 500, 'max_depth': 3, 'learning_rate': 0.2,
'colsample_bytree': 0.5}
```

In [0]:
```
x_cfl=XGBClassifier(n_estimators=500,subsample=0.5,learning_rate=0.2,colsample_by
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid',cv=3)
c_cfl.fit(X_train_asm,y_train_asm)
```

Out[32]:
```
CalibratedClassifierCV(base_estimator=XGBClassifier(base_score=0.5, booster='gb
tree', colsample_bylevel=1,
       colsample_bytree=0.5, gamma=0, learning_rate=0.2, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=500,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.5),
            cv=3, method='sigmoid')
```

In [0]:
```
predict_y = c_cfl.predict_proba(X_train_asm)
print ('train loss',log_loss(y_train_asm, predict_y))
```

```
train loss 0.010377843567128
```

In [0]:
```
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
```

```
cv loss 0.013632749128642
```

In [0]:
```
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict_y))
```

```
test loss 0.01431311263546474
```

In [0]:
```
print(X_train_asm.shape)
print(y_train_asm.shape)
```

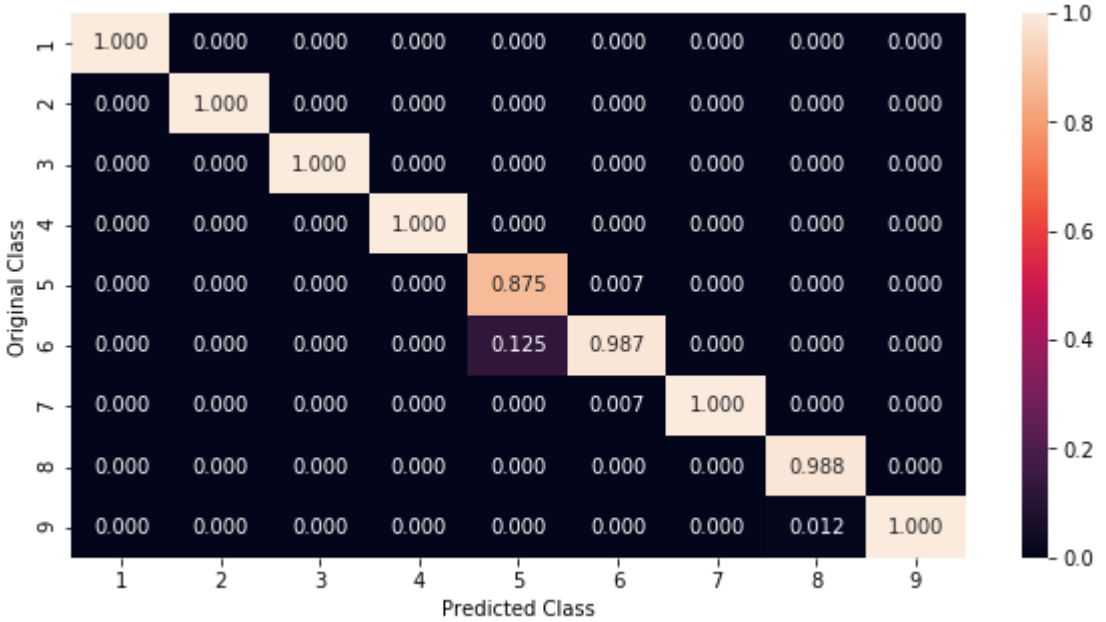
```
(6955, 1259)
(6955,)
```

In [0]:
```
%matplotlib inline
plot_confusion_matrix(y_test_asm,c_cfl.predict(X_test_asm))
```

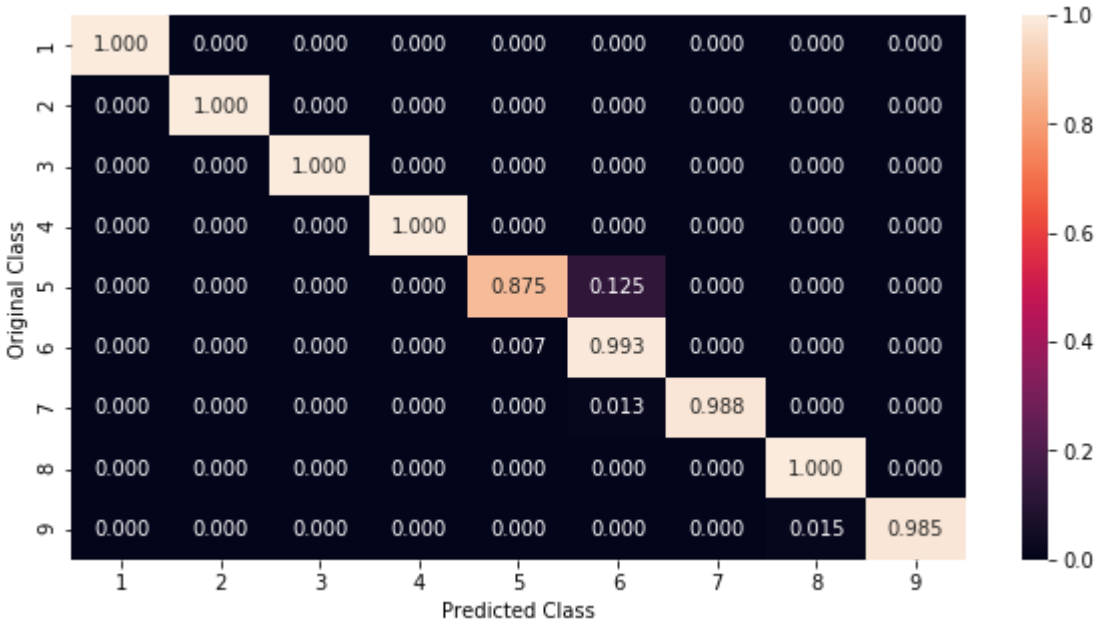
```
Number of misclassified points  0.27598896044158233
---------------------------------------------------- Confusion matrix ----------
----------------------------------------
```

```
<Figure size 1080x576 with 0 Axes>
```


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

In [0]:
```
# Code to read csv file into Colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
link = 'https://drive.google.com/open?id=12LIFrxVEcAnTSgDDsraUMAi6yzV1zGqk' # The
```

In [0]:
```
fluff, id = link.split('=')
print (id) # Verify that you have everything after '='
```

```
12LIFrxVEcAnTSgDDsraUMAi6yzV1zGqk
```

In [0]:
```
import pandas as pd
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('finalsubmission - microsoftmalwaredetectiontable - She
datafra= pd.read_csv('finalsubmission - microsoftmalwaredetectiontable - Sheet1.c
```

In [0]:
```
datafra[:5]
```

Out[11]:

	MACHINE LEARNING MODEL	BEST HYPERPARAMETER VALUE	TRAIN LOG LOSS	CROSS VALIDATION LOGLOSS	TEST LOG LOSS
0	LOGISTIC REGRESSION	0.1(ALPHA)	0.264	0.288	0.282
1	MULTINOMIAL NAIVE BAYES	0.00001(ALPHA)	0.660	0.670	0.670
2	K NEAREST NEIGHBORS	3(VALUES OF K)	0.140	0.260	0.260
3	RANDOM FOREST CLASSIFIER	500(NUMBER OF ESTIMATORS)	0.020	0.050	0.050
4	XGB WITH RANDOM SEARCH CV	NUMBER OF ESTIMATORS=1000,MAXIMUMDEPTH=10,LEAR...	0.010	0.012	0.014

we have achieved log loss of 0.01 with the xgb classifier with random search cv.

in task2 studing the anlaysis done by dchad by dchad in vision of reducing of logloss

the key take aways are

- use the image features
- using chi square method get the best features
- apply various models and reduce the log loss which are the requrirements that are satisfied above in the task 2 assignment. ***thank you***

using the dchad github account and analysis done by him we are able to achieve the log loss of 0.01 which is very good.

##DOCUMENTATION

- IN TASK 2 WE HAVE TAKEN THE FEATRUES OF .ASM FILES AND .BYTE FILES AND APPLY CHI2 ON TOP OF THE FEATURES AND ANALYSED THE MODELS HOW THEY ARE PERFORMING. THEN WE APPLIED THE IMAGE FEATURES ALONG WITH THE .BYTE FILES AND .ASM FILES APPLIED MODELS ON THEM LIKE
- LOGISTIC REGRESSION
- MULTINOMIAL NAIVE BAYES
- K NEAREST NEIGHBORS
- RANDOM FOREST CLASSIFIER
- XGB WITH RANDOM SEARCH CV

OBTAINED VERY GOOD RESULTS OF TEST LOGLOSS LIKE 0.05 AND 0.01.

In [0]: