Assignment

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

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt

import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
import scipy
from scipy import sparse
from sklearn.feature_extraction.text import CountVectorizer
from tqdm import tqdm
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
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import normalize
```

1. Add bigrams to byte files

In [8]:
```
#unigrams
byte_features=pd.read_csv("result_with_size.csv")
byte_features['ID']  = byte_features['ID'].str.split('.').str[0]
byte_features.head(2)
```

Out[8]:

	Unnamed: 0	ID	0	1	2	3	4	5	6	7	...	
0	0	01azqd4InC7m9JpocGv5	601905	3905	2816	3832	3345	3242	3650	3201	...	31(
1	1	01IsoiSMh5gxyDYTl4CB	39755	8337	7249	7186	8663	6844	8420	7589	...	4:

2 rows × 261 columns

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

In [10]:
```
bigram_vocab = byte_bigram()
bigram_vocab[:20]
```

```
length of bigrams  66049
```

Out[10]:
```
['00 00',
 '00 01',
 '00 02',
 '00 03',
 '00 04',
 '00 05',
 '00 06',
 '00 07',
 '00 08',
 '00 09',
 '00 0a',
 '00 0b',
 '00 0c',
 '00 0d',
 '00 0e',
 '00 0f',
 '00 10',
 '00 11',
 '00 12',
 '00 13']
```

In [39]:
```
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

In [71]:
```
source='byteFiles/'
files = os.listdir('byteFiles')
#ID=df['Id'].tolist()
data=np.arange(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'first')
    elif i%5==1:
        shutil.move(source+files[data[i]],'second')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'third')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fifth')
```

In [7]:
```
from multiprocessing import Process# this is used for multithreading
import multiprocessing
def firstprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabul
ary=bigram_vocab)
    bigram_features = len(bigram_vocab)
    bytebigram_vect1 = scipy.sparse.csr_matrix((2174, bigram_features))
    for i, file in tqdm(enumerate(os.listdir('first'))):
        f = open('first/' + file)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        #bytebigram_vect[i:]+= scipy.sparse.csr_matrix(vector.fit_trans
form([f.read().replace('\n', ' ').lower()]))
        bytebigram_vect1[i, :] += scipy.sparse.csr_matrix(vector.fit_tr
ansform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('first_bigrams.npz', bytebigram_vect1)
    print("first process done")
def secondprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabul
ary=bigram_vocab)
    bigram_features = len(bigram_vocab)
    bytebigram_vect2 = scipy.sparse.csr_matrix((2174, bigram_features))
    for i, file in tqdm(enumerate(os.listdir('second'))):
        f = open('second/' + file)
        #bytebigram_vect[i:]+= scipy.sparse.csr_matrix(vector.fit_trans
form([f.read().replace('\n', ' ').lower()]))
        bytebigram_vect2[i, :] += scipy.sparse.csr_matrix(vector.fit_tr
ansform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('second_bigrams.npz', bytebigram_vect2)
    print("second process done")
def thirdprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabul
ary=bigram_vocab)
    bigram_features = len(bigram_vocab)
    bytebigram_vect3 = scipy.sparse.csr_matrix((2174, bigram_features))
    for i, file in tqdm(enumerate(os.listdir('third'))):
        f = open('third/' + file)
        #bytebigram_vect[i:]+= scipy.sparse.csr_matrix(vector.fit_trans
form([f.read().replace('\n', ' ').lower()]))
        bytebigram_vect3[i, :] += scipy.sparse.csr_matrix(vector.fit_tr
ansform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('third_bigrams.npz', bytebigram_vect3)
    print("third process done")
def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present Sys
tem
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
    p3=Process(target=thirdprocess)
    #p4=Process(target=fourthprocess)
    #p5=Process(target=fifthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    #p4.start()
    #p5.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    #p4.join()
    #p5.join()

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

```
2174it [2:12:39,  3.66s/it]
2174it [2:13:54,  3.70s/it]
2160it [2:13:54,  6.21s/it]
```

```
third process done
```

```
2172it [2:15:09,  6.90s/it]
```

```
second process done
```

```
2174it [2:15:22,  3.74s/it]
```

```
first process done
```

In [8]:

```
def fourthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabul
ary=bigram_vocab)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    bigram_features = len(bigram_vocab)
    bytebigram_vect4 = scipy.sparse.csr_matrix((2173, bigram_features))
    for i, file in tqdm(enumerate(os.listdir('fourth'))):
        f = open('fourth/' + file)
        #bytebigram_vect[i:]+= scipy.sparse.csr_matrix(vector.fit_trans
form([f.read().replace('\n', ' ').lower()]))
        bytebigram_vect4[i, :] += scipy.sparse.csr_matrix(vector.fit_tr
ansform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('fourth_bigrams.npz', bytebigram_vect4)
    print("fourth process done")
def fifthprocess():
    vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabul
ary=bigram_vocab)
    bigram_features = len(bigram_vocab)
    bytebigram_vect5 = scipy.sparse.csr_matrix((2173, bigram_features))
    for i, file in tqdm(enumerate(os.listdir('fifth'))):
        f = open('fifth/' + file)
        #bytebigram_vect[i:]+= scipy.sparse.csr_matrix(vector.fit_trans
form([f.read().replace('\n', ' ').lower()]))
        bytebigram_vect5[i, :] += scipy.sparse.csr_matrix(vector.fit_tr
ansform([f.read().replace('\n', ' ').lower()]))
        f.close()
    scipy.sparse.save_npz('fifth_bigrams.npz', bytebigram_vect5)
    print("fifth process done")

def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present Sys
tem
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    #p1=Process(target=firstprocess)
    #p2=Process(target=secondprocess)
    #p3=Process(target=thirdprocess)
    p4=Process(target=fourthprocess)
    p5=Process(target=fifthprocess)
    #p1.start() is used to start the thread execution
    #p1.start()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    #p2.start()
    #p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joined
    #p1.join()
    #p2.join()
    #p3.join()
    p4.join()
    p5.join()

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

```
2173it [2:13:51,  3.70s/it]
2173it [2:14:35,  3.72s/it]
```

```
fourth process done
fifth process done
```

In [7]:
```
first= sparse.load_npz("first_bigrams.npz")
second= sparse.load_npz("second_bigrams.npz")
third= sparse.load_npz("third_bigrams.npz")
fourth= sparse.load_npz("fourth_bigrams.npz")
fifth= sparse.load_npz("fifth_bigrams.npz")
bigrams = sparse.vstack([first,second,third,fourth,fifth])
scipy.sparse.save_npz('bigrams.npz', bigrams)
```

Normalise the bigrams

In [9]:
```
bigram_norm = normalize(scipy.sparse.load_npz('bigrams.npz'), axis = 0)
```

Add column names to the dataframe(i.e bigram vocabulary)

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [10]:
```
bigram.columns = list(bigram_vocab)
```

In [11]:
```
first_folder_file_ids=os.listdir('first')
second_folder_file_ids=os.listdir('second')
third_folder_file_ids=os.listdir('third')
fourth_folder_file_ids=os.listdir('fourth')
fifth_folder_file_ids=os.listdir('fifth')

cleaned_first_id=[]
cleaned_second_id=[]
cleaned_third_id=[]
cleaned_fourth_id=[]
cleaned_fifth_id=[]

for i in first_folder_file_ids:
    cleaned_first_id.append(i[:-6])
for i in second_folder_file_ids:
    cleaned_second_id.append(i[:-6])
for i in third_folder_file_ids:
    cleaned_third_id.append(i[:-6])
for i in fourth_folder_file_ids:
    cleaned_fourth_id.append(i[:-6])
for i in fifth_folder_file_ids:
    cleaned_fifth_id.append(i[:-6])

cleaned_id = cleaned_first_id+cleaned_second_id+cleaned_third_id+cleane
d_fourth_id+cleaned_fifth_id

id1 = pd.DataFrame(cleaned_id,columns=['ID'])
#bigrams_with_id = pd.concat([id,bigram],axis=1)

#shuffled_id=pd.DataFrame(cleaned_id,columns=['ID'])
class1 = byte_features[['ID','Class']]

bigram_suffled_id_class = pd.merge(id1,class1,on='ID')
```

In [12]:
```
def imp_features(input_values,output_values,features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    rf.fit(input_values, output_values)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    return imp_feature_indx[:keep]
```

In [13]:
```
byte_bi_indxes = imp_features(bigram_norm,bigram_suffled_id_class['Class'],bigram_vocab, 300)
```

In [77]:
```
scipy.sparse.save_npz('bigramsnorma.npz', bigram_norm)
```

In [4]:
```
bigram_norm= sparse.load_npz("bigramsnorma.npz")
```

In [5]:
```
bigram_dense=bigram_norm.todense()
```

In [6]:
```
bigram_df = pd.DataFrame(bigram_dense)
```

In [43]:
```
bigram_top_features=bigram_df.take(byte_bi_indxes,axis=1)
bigram_top_features.head(2)
```

Out[43]:

	52725	60991	52201	7915	36132	29278	38026	60178	23068	2
0	0.001281	0.01027	0.003833	0.004145	0.004929	0.000115	0.021735	0.004653	0.000989	0.00
1	0.000075	0.00000	0.000000	0.000000	0.000469	0.000014	0.002173	0.000582	0.000433	0.00

2 rows × 300 columns

Mapping index with feature names

In [26]:
```
index_feature=dict()
index=np.arange(66049)
for i in index:
    index_feature[i]=bigram_vocab[i]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [27]:
```
top_features=[]
for i in byte_bi_indxes:
    top_features.append(index_feature[i])
```

In [50]:
```
bigram_top_features.to_csv("top_bigram_features",header=top_features)
```

In [60]:
```
data=pd.read_csv("top_bigram_features")
data.head(2)
```

Out[60]:

	Unnamed: 0	cd 28	ed 52	cb 1e	1e cd	8c 98	71 ed	93 f7	ea 28	
0	0	0.001281	0.01027	0.003833	0.004145	0.004929	0.000115	0.021735	0.004653	0.0
1	1	0.000075	0.00000	0.000000	0.000000	0.000469	0.000014	0.002173	0.000582	0.0

2 rows × 301 columns

In [61]:
```
data=data.drop("Unnamed: 0",axis=1)
data.head(2)
```

Out[61]:

	cd 28	ed 52	cb 1e	1e cd	8c 98	71 ed	93 f7	ea 28	59 c3	
0	0.001281	0.01027	0.003833	0.004145	0.004929	0.000115	0.021735	0.004653	0.000989	0.00
1	0.000075	0.00000	0.000000	0.000000	0.000469	0.000014	0.002173	0.000582	0.000433	0.00

2 rows × 300 columns

In [62]:
```
data = pd.concat([id1,data,bigram_suffled_id_class['Class']],axis=1)
data.head(2)
```

Out[62]:

	ID	cd 28	ed 52	cb 1e	1e cd	8c 98	71 ed	93 f7
0	fjAbOtkGQo9nRYiysBIP	0.001281	0.01027	0.003833	0.004145	0.004929	0.000115	0.021735

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	ID	cd 28	ed 52	cb 1e	1e cd	8c 98	71 ed	93 f7
1	HV0ctLUKfW1ozkmC7BMJ	0.000075	0.00000	0.000000	0.000000	0.000469	0.000014	0.002173

2 rows × 302 columns

In [63]:
```
data.shape
```

Out[63]:
```
(10868, 302)
```

In [64]:
```
data.to_csv("top_bigrams_with_id_class.csv")
```

In [ ]:
```
data_unigram = pd.read_csv("result_with_size.csv")
data_bigram - pd.read_csv("top_bigrams_with_id_class.csv")
```

Train Test Split

In [2]:
```
result=pd.read_csv("top_bigrams_with_id_class.csv")
data_y=result["Class"]
# split the data into test and train by maintaining same distribution o
f output varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID',
'Class'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,strati
fy=y_train,test_size=0.20)
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

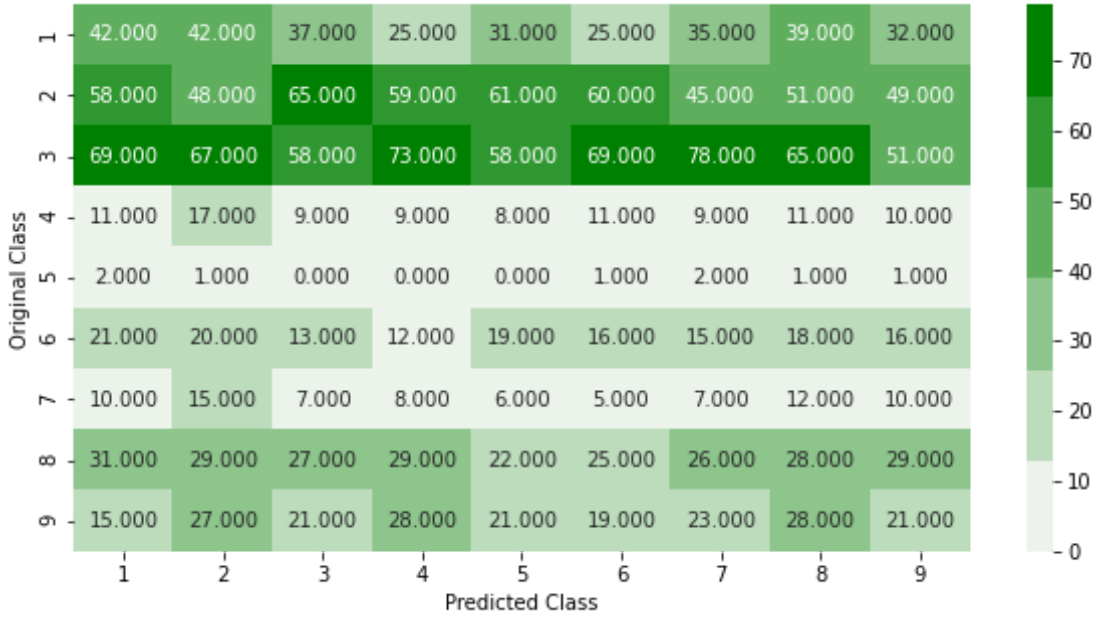
In [2]:
```
def plot_confusion_matrix(test_y, predict_y):
    %matplotlib inline
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/l
en(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

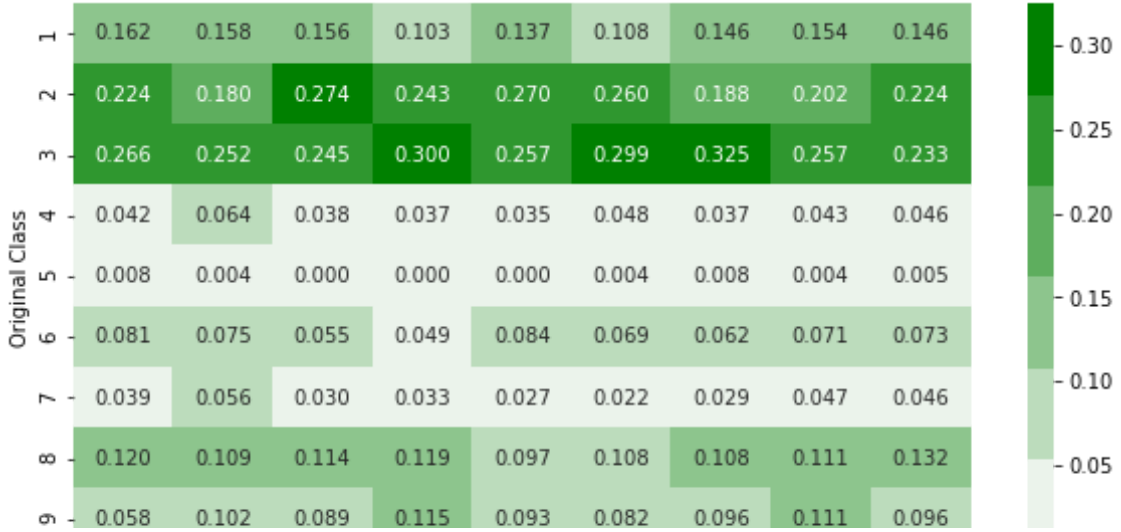
In [71]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.49069599975710
56
Log loss on Test Data using Random Model 2.5219534713142044
Number of misclassified points  89.46642134314628
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

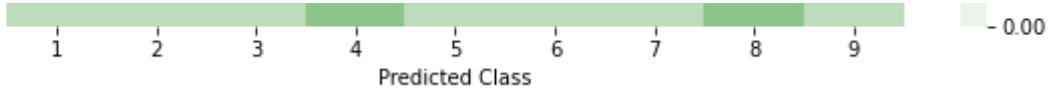
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



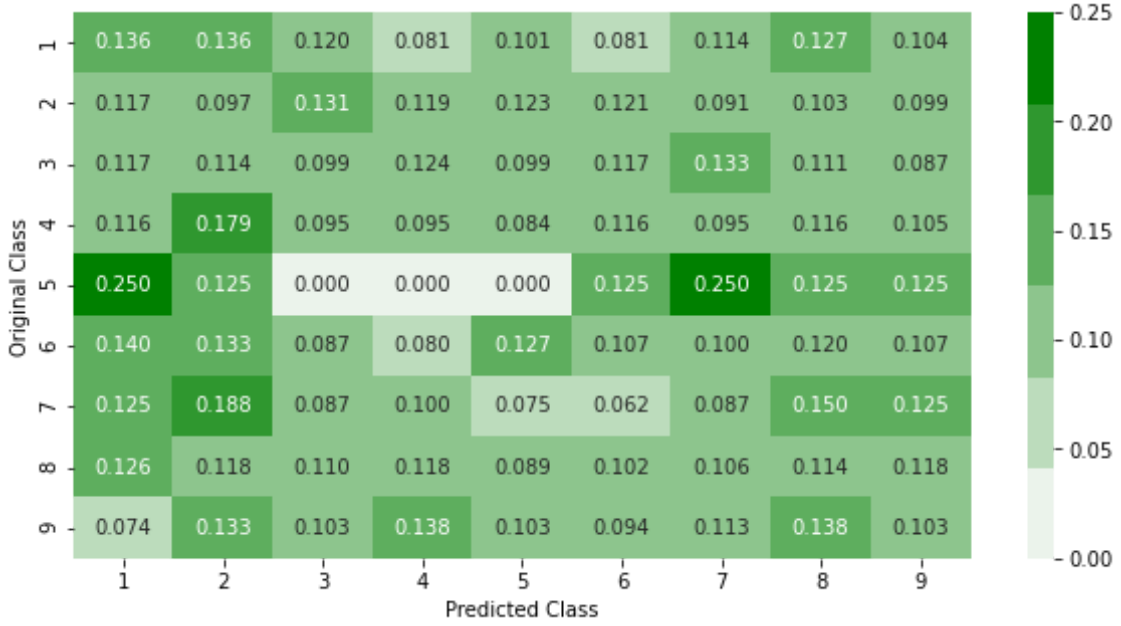
---------- Precision matrix ----------



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
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 [72]:
```
#KNN
```

In [73]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

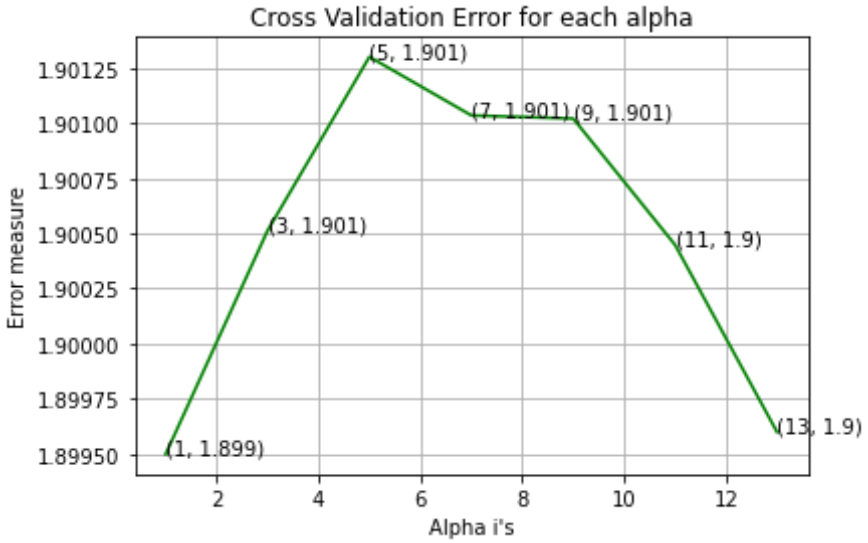
```
# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-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


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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

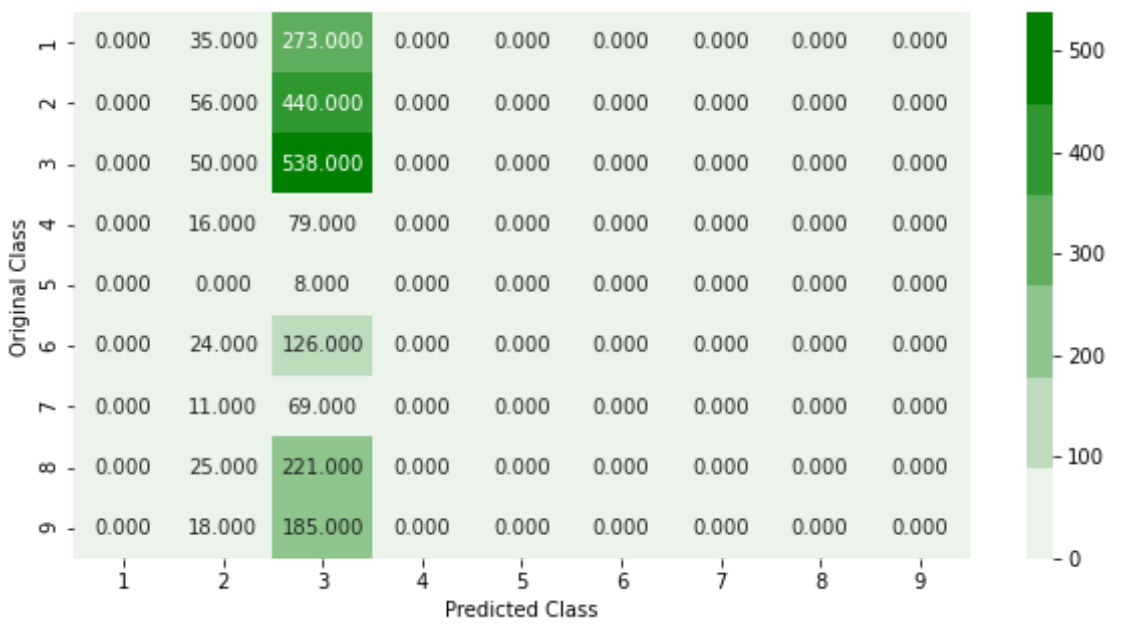
```
log_loss for k =  1 is 1.8994977705165401
log_loss for k =  3 is 1.900512642718071
log_loss for k =  5 is 1.901301609227919
log_loss for k =  7 is 1.9010357538052964
log_loss for k =  9 is 1.9010214058283499
log_loss for k =  11 is 1.9004475276618655
log_loss for k =  13 is 1.8995984989427965
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

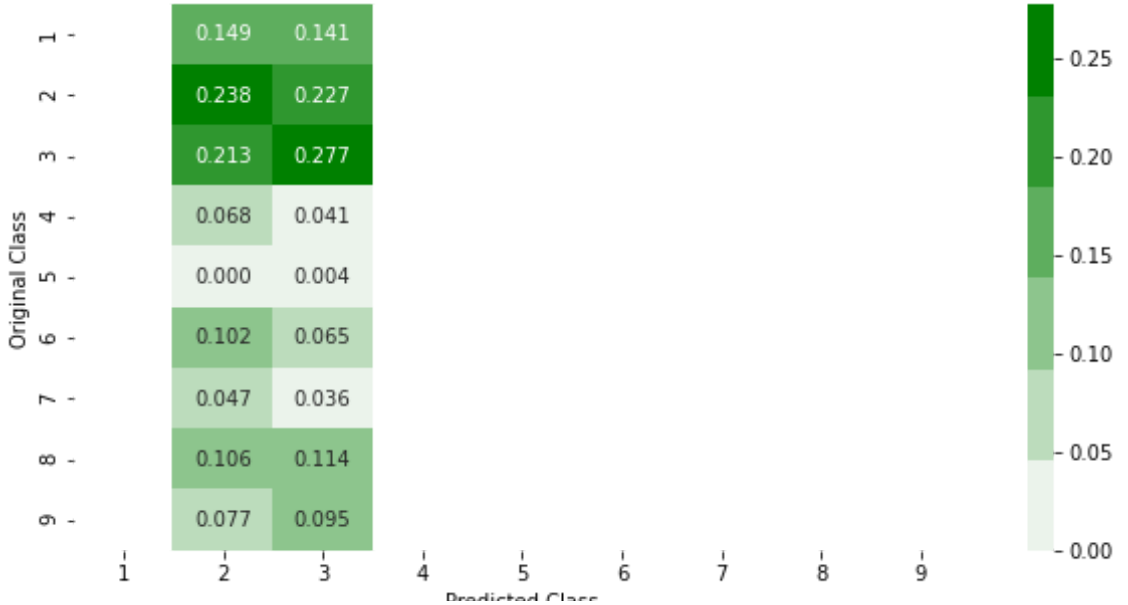



```
For values of best alpha =  1 The train log loss is: 1.82660857082961
74
For values of best alpha =  1 The cross validation log loss is: 1.899
4977705165401
For values of best alpha =  1 The test log loss is: 1.896112223634373
Number of misclassified points  72.67709291628334
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



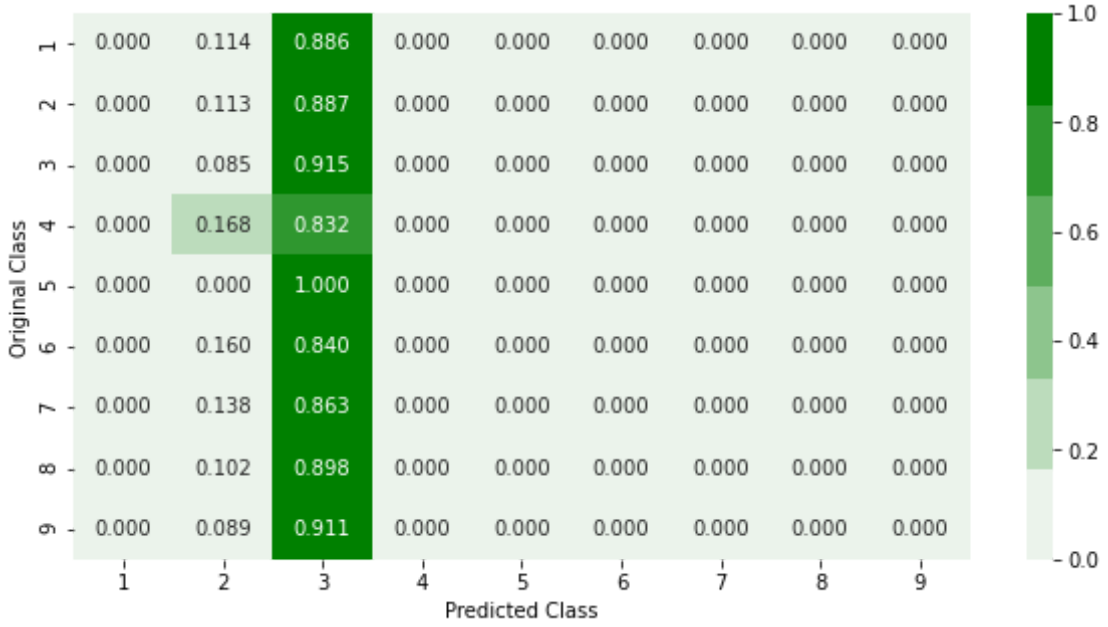
---------- Precision matrix -



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Predicted Class

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

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




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

In [74]:
```
#LR
```

In [75]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

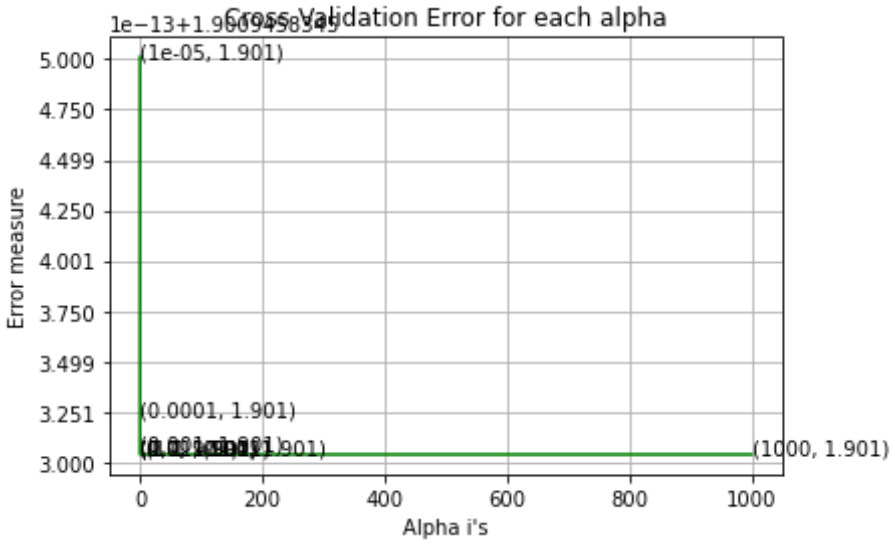
```
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balance
d')
    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=logistic
R.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

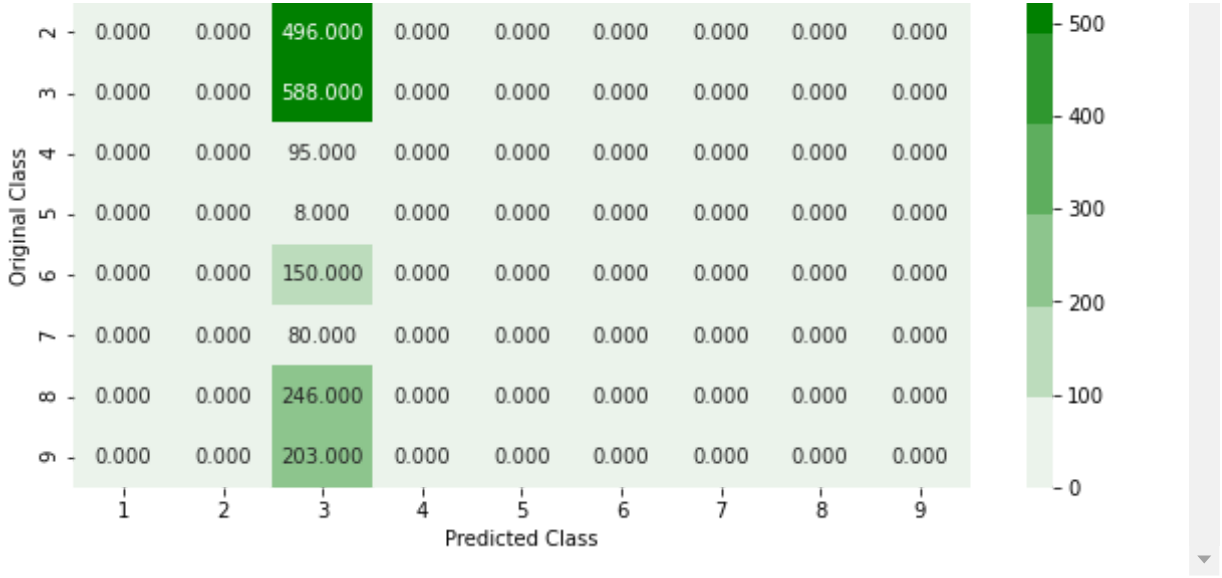
```
log_loss for c =  1e-05 is 1.9009458345125008
log_loss for c =  0.0001 is 1.9009458345123238
log_loss for c =  0.001 is 1.9009458345123063
log_loss for c =  0.01 is 1.9009458345123045
log_loss for c =  0.1 is 1.900945834512304
log_loss for c =  1 is 1.900945834512304
log_loss for c =  10 is 1.900945834512304
log_loss for c =  100 is 1.900945834512304
log_loss for c =  1000 is 1.900945834512304
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



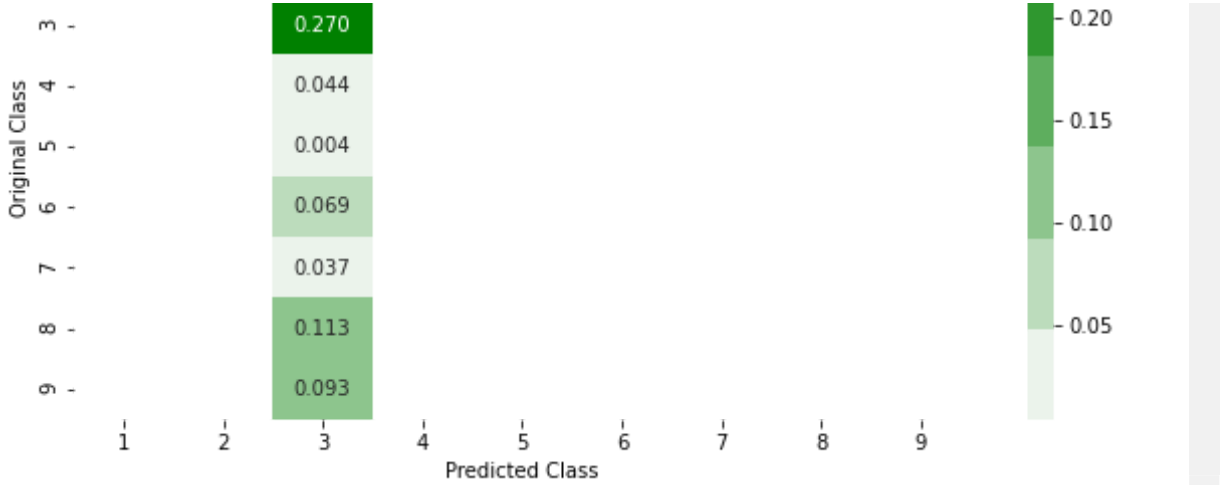

```
log loss for train data 1.8988433646420713
log loss for cv data 1.900945834512304
log loss for test data 1.8996951311183299
Number of misclassified points  72.95308187672494
---------------------------------------------------- Confusion matrix ---
-------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API
PDFCROWD

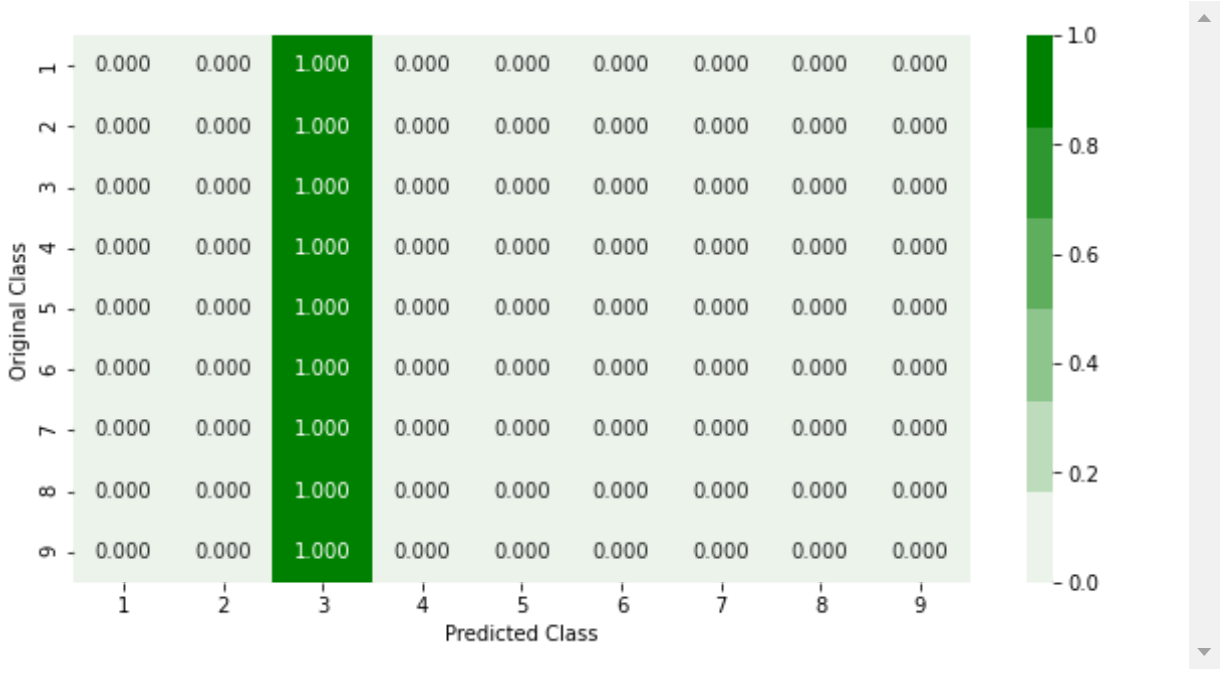


---------- Precision matrix ---

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

In [76]:

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

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

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

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

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

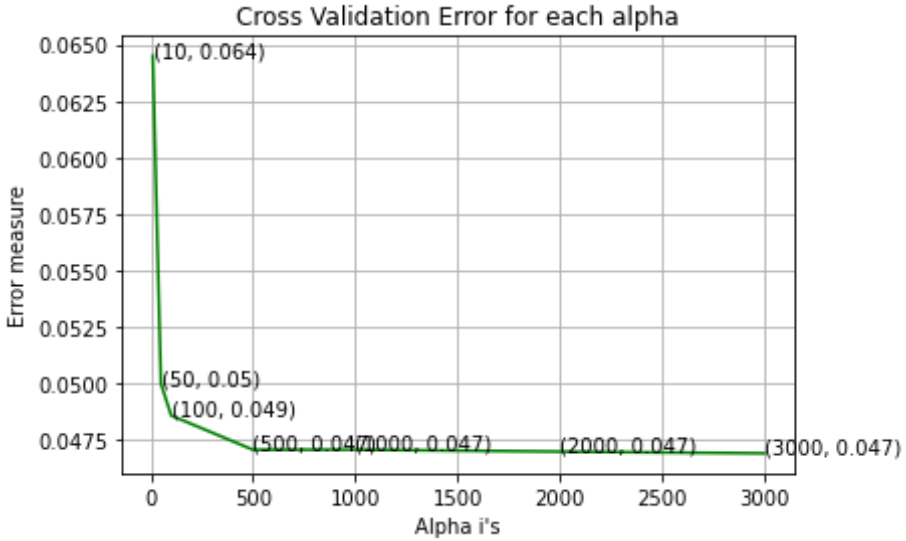
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
asses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_stat
e=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

```
log_loss for c =  10 is 0.06449873222207841
log_loss for c =  50 is 0.04997887544221177
log_loss for c =  100 is 0.04856408549644968
```

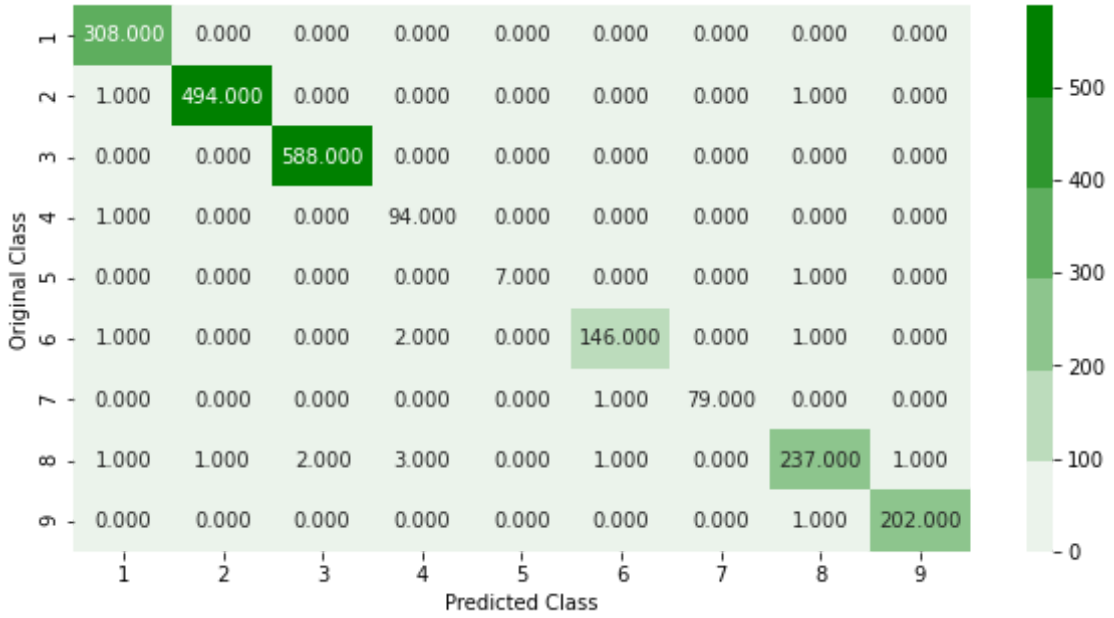
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
log_loss for c =  500 is 0.04704476131042981
log_loss for c =  1000 is 0.04704914874064433
log_loss for c =  2000 is 0.04696314810894241
log_loss for c =  3000 is 0.04688680429039253
```



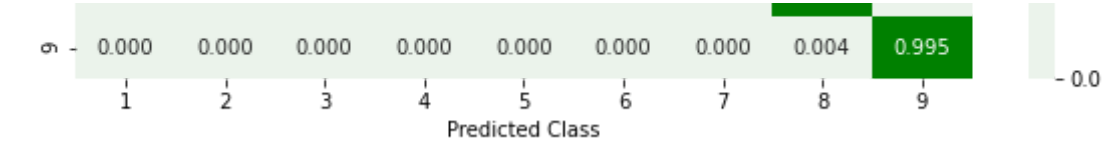

```
For values of best alpha =  3000 The train log loss is: 0.01711271032
780758
For values of best alpha =  3000 The cross validation log loss is: 0.
04688680429039253
For values of best alpha =  3000 The test log loss is: 0.046737280519
71073
Number of misclassified points  0.8739650413983441
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

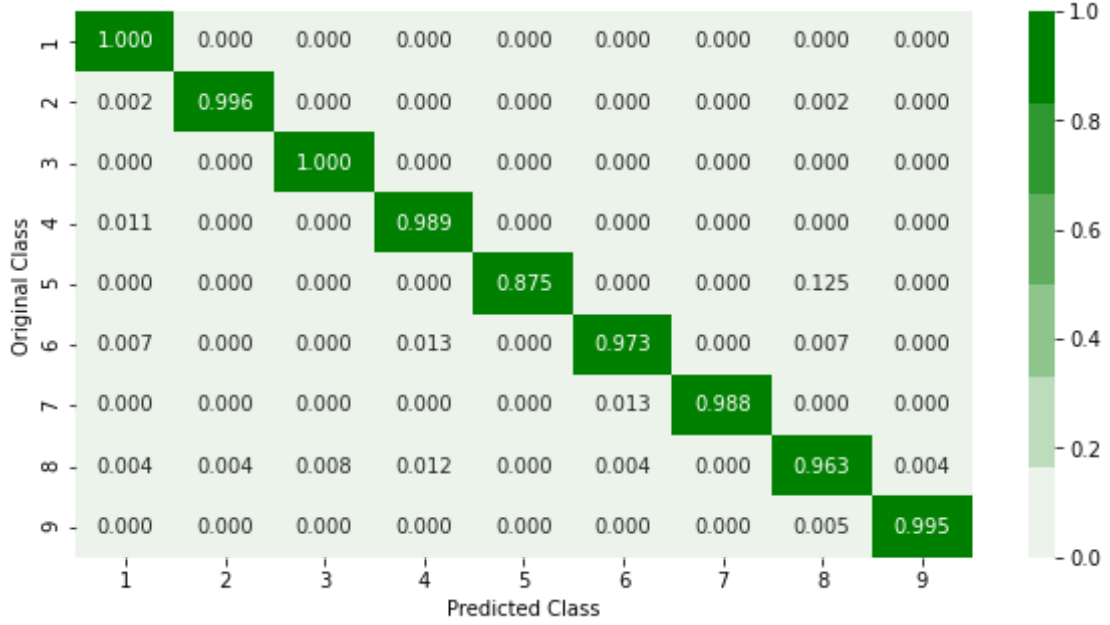


---------- Precision matrix -

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
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 [77]:
```
#xgboost
```

In [4]:
```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
100%|██████████| 6/6 [1:03:07<00:00, 631.27s/it]
```

```
log_loss for c =  10 is 0.06216189360188632
log_loss for c =  50 is 0.04303777952998256
log_loss for c =  100 is 0.042422409209431314
log_loss for c =  500 is 0.043201983912785555
log_loss for c =  1000 is 0.04334610368382352
log_loss for c =  2000 is 0.043345712723028
```

```
For values of best alpha =  100 The train log loss is: 0.01648544385635
363
For values of best alpha =  100 The cross validation log loss is: 0.042
422409209431314
For values of best alpha =  100 The test log loss is: 0.044795136054546
894
```

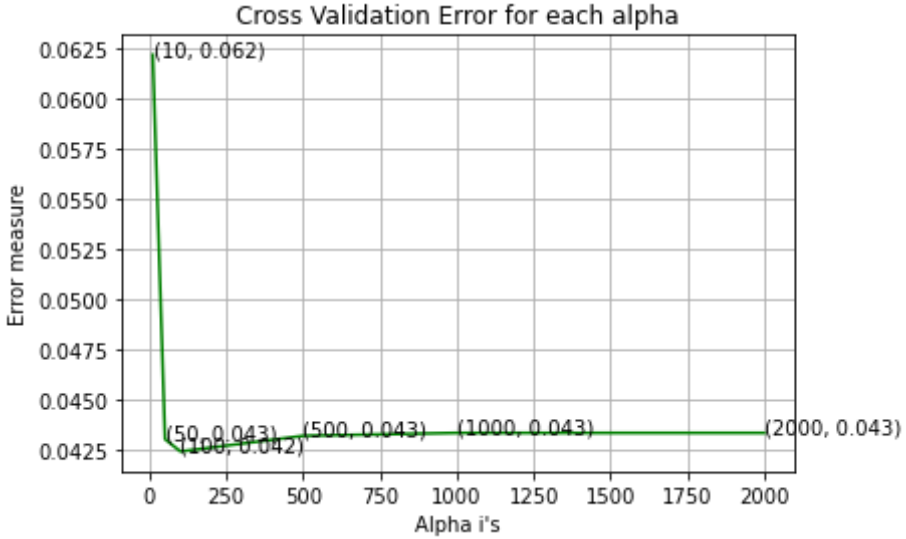
```
---------------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-4-d82be15a853c> in <module>
     56 predict_y = sig_clf.predict_proba(X_test)
     57 print('For values of best alpha = ', alpha[best_alpha], "The te
st log loss is:",log_loss(y_test, predict_y))
---> 58 plot_confusion_matrix(y_test, sig_clf.predict(X_test))

NameError: name 'plot_confusion_matrix' is not defined
```

In [8]:

```
best_alpha = np.argmin(cv_log_error_array)
%matplotlib inline
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[i]))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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



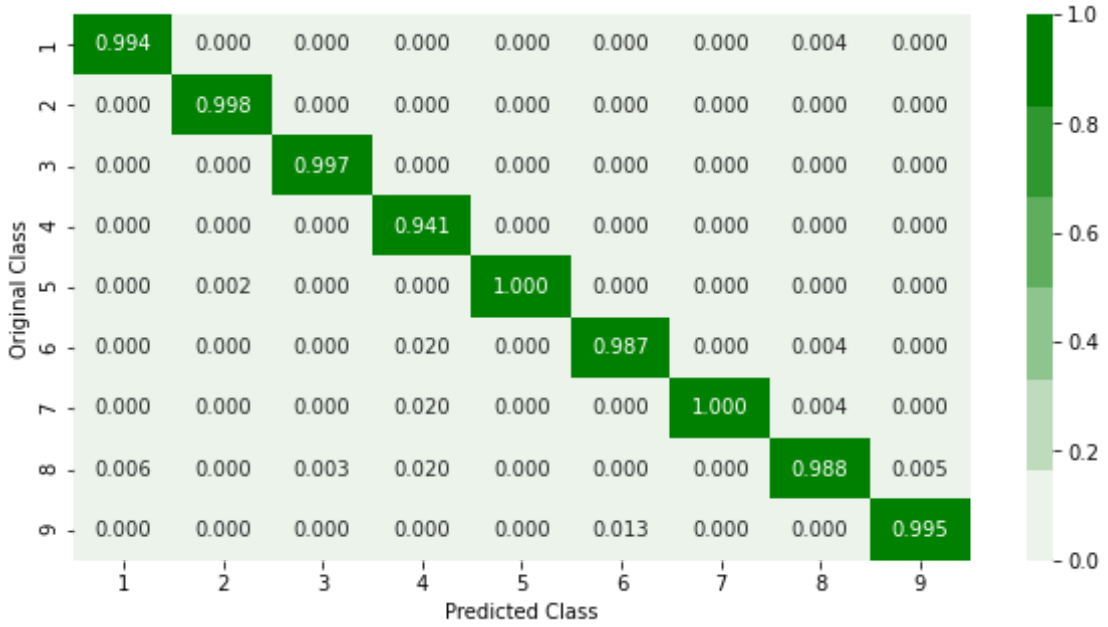

```
For values of best alpha =  100 The train log loss is: 0.01648544385635
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

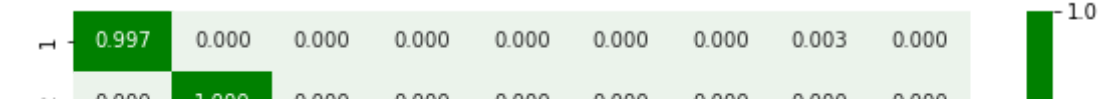
```
363
For values of best alpha =  100 The cross validation log loss is: 0.042
422409209431314
For values of best alpha =  100 The test log loss is: 0.044795136054546
894
Number of misclassified points  0.78196872125115
---------------------------------------------------- Confusion matrix ---
--------------------------------------------------
```


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

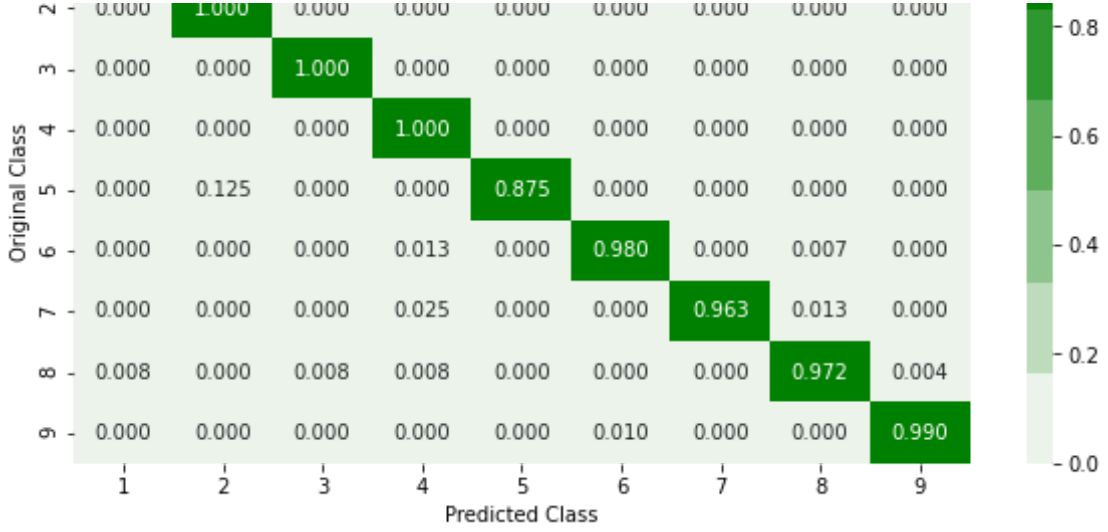
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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

---------- Bigrams on byte features ----------

Model	Hyper parameter	Test Loss	Miss-classification
Random Model	-	2.52195	89.4 %
KNN	1	1.89611	72.6 %
Logistic Regression	0.1	1.89969	72.9 %

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Model	Hyper parameter	Test Loss	Miss-classification
Random Forest Classifier	3000	0.04673	0.87 %
XGBoost Classsifier	100	0.04479	0.78 %

Byte_unigrams + Byte_bigrams

In [203]:
```
data1 = pd.read_csv("result.csv")
data1.head(3)
```

Out[203]:

	ID	0	1	2	3	4	5	6	7	8	...	f7
0	01azqd4InC7m9JpocGv5.txt	601905	3905	2816	3832	3345	3242	3650	3201	2965	...	2804
1	01IsoiSMh5gxyDYTl4CB.txt	39755	8337	7249	7186	8663	6844	8420	7589	9291	...	451
2	01jsnpXSAlgw6aPeDxrU.txt	93506	9542	2568	2438	8925	9330	9007	2342	9107	...	2325

3 rows × 258 columns

In [204]:
```
id=data1['ID']
data1=data1.drop("ID",axis=1)

cid=[]
for i in id:
    cid.append(i.split(".")[0])

data1["ID"] = cid
data1.head(3)
```

Out[204]:

	0	1	2	3	4	5	6	7	8	9	...	f8	f9	fa	fb
0	601905	3905	2816	3832	3345	3242	3650	3201	2965	3205	...	3687	3101	3211	3097
1	39755	8337	7249	7186	8663	6844	8420	7589	9291	358	...	6536	439	281	302

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	0	1	2	3	4	5	6	7	8	9	...	f8	f9	fa	fb
2	93506	9542	2568	2438	8925	9330	9007	2342	9107	2457	...	2358	2242	2885	2863

3 rows × 258 columns

In [208]:
```
data2 = pd.read_csv("top_bigrams_with_id_class.csv")
data2.head(2)
```

Out[208]:

	Unnamed: 0	ID	cd 28	ed 52	cb 1e	1e cd	8c 98	71 ed
0	0	fjAbOtkGQo9nRYiysBIP	0.001281	0.01027	0.003833	0.004145	0.004929	0.000115
1	1	HV0ctLUKfW1ozkmC7BMJ	0.000075	0.00000	0.000000	0.000000	0.000469	0.000014

2 rows × 303 columns

In [209]:
```
data2 = data2.drop("Unnamed: 0",axis=1)
data = pd.merge(data2,data1,on="ID")
data.head(3)
```

Out[209]:

	ID	cd 28	ed 52	cb 1e	1e cd	8c 98	71 ed	93 f7
0	fjAbOtkGQo9nRYiysBIP	0.001281	0.010270	0.003833	0.004145	0.004929	0.000115	0.021735
1	HV0ctLUKfW1ozkmC7BMJ	0.000075	0.000000	0.000000	0.000000	0.000469	0.000014	0.002173
2	3RiBw0ntobeCh5VEQrY1	0.000151	0.004565	0.001095	0.000000	0.000235	0.000014	0.002173

3 rows × 559 columns

Train Test Split

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [210]:
```
data_y = data['Class']
# split the data into test and train by maintaining same distribution o
f output varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(data.drop(['ID','Cl
ass'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,strati
fy=y_train,test_size=0.20)
```

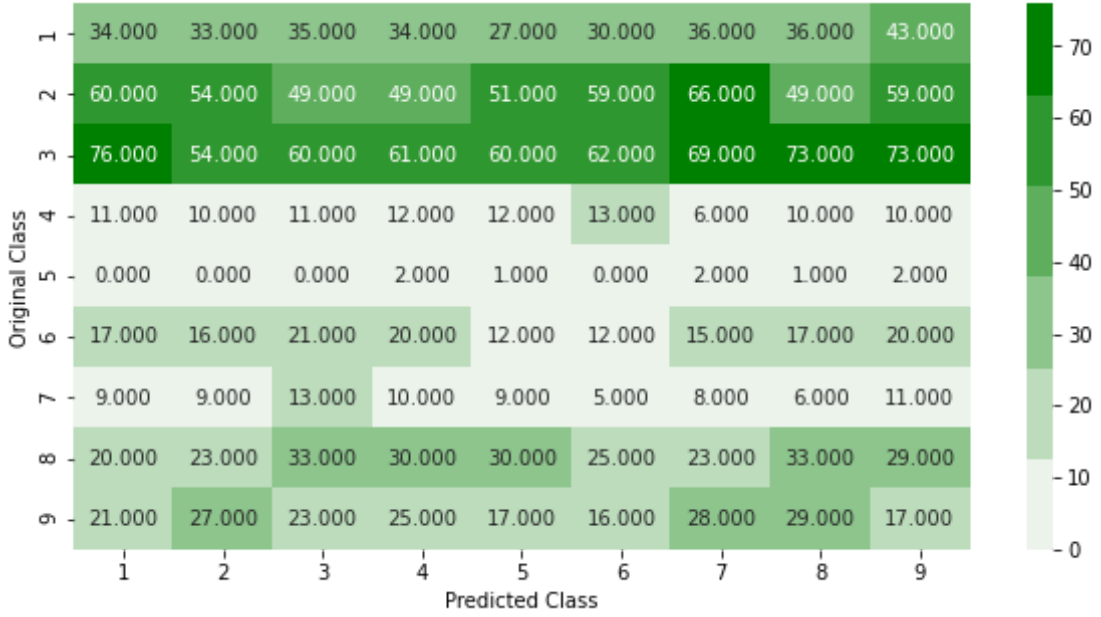
Random Model

In [212]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

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


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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

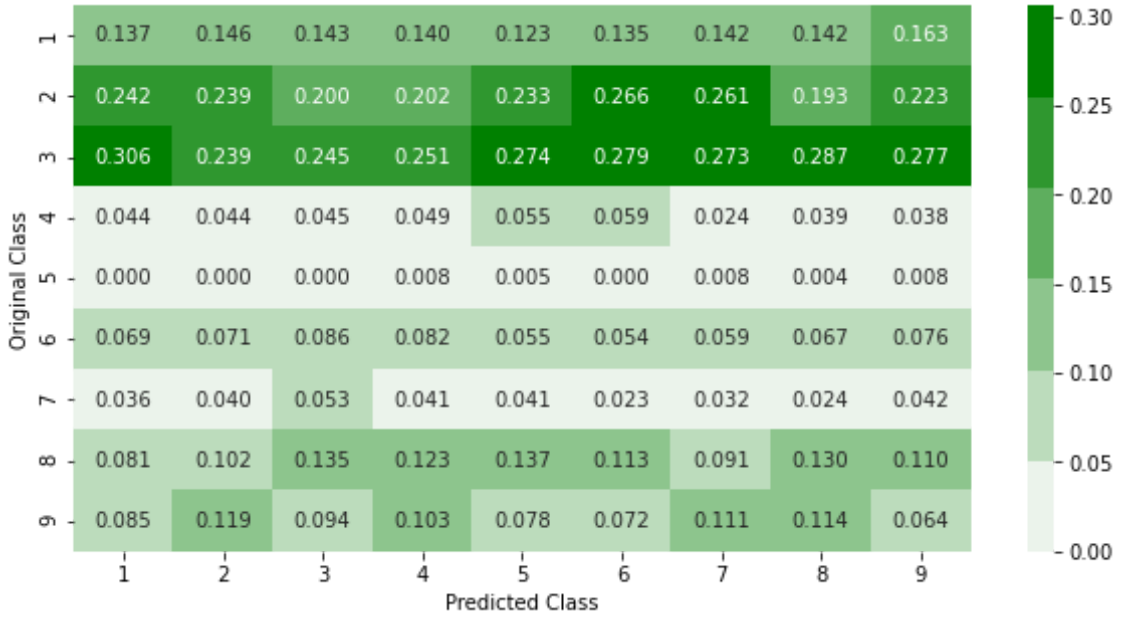
```
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.50665252710059
86
Log loss on Test Data using Random Model 2.5241505483193247
Number of misclassified points  89.37442502299908
------------------------------------------------------ Confusion matrix -
------------------------------------------------------
```



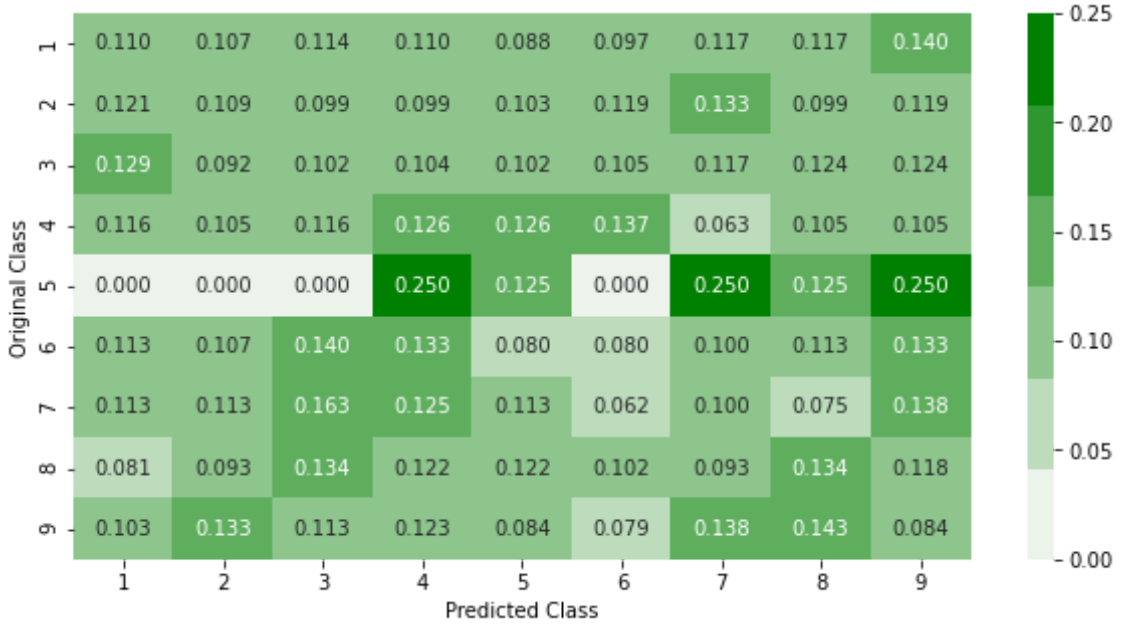

```
------------------------------------------------------ Precision matrix -
------------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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

KNN

In [214]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

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

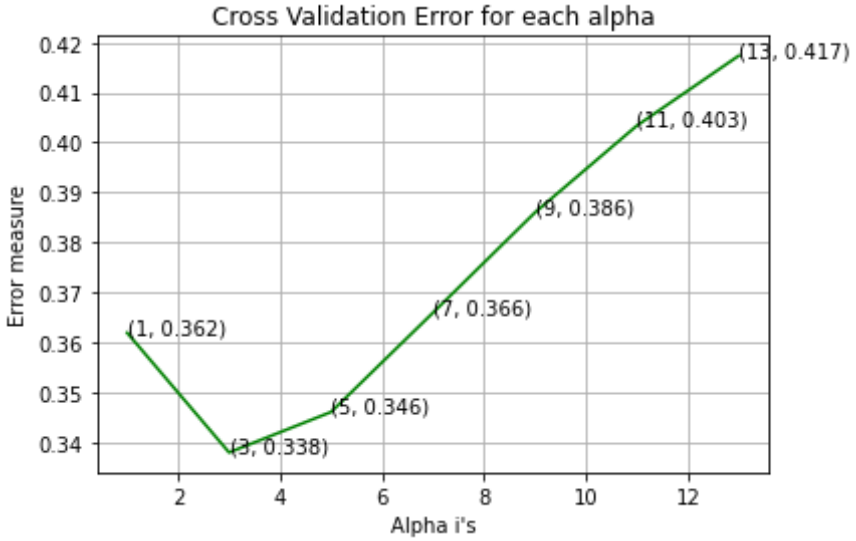
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
asses_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

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

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

```
log_loss for k =  1 is 0.36189200802888577
log_loss for k =  3 is 0.3379356450066001
log_loss for k =  5 is 0.3460382981457452
log_loss for k =  7 is 0.3659092189335617
log_loss for k =  9 is 0.3859599442500066
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

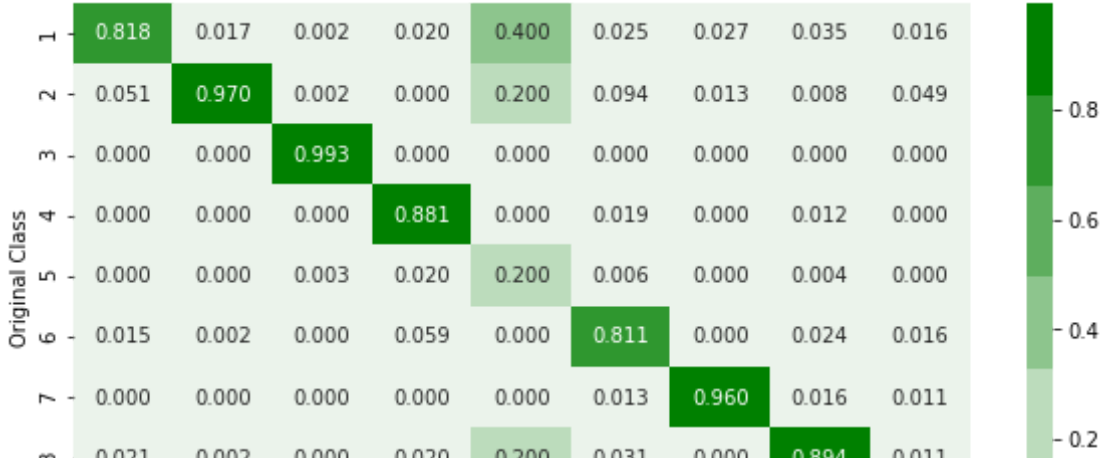
```
log_loss for k =  11 is 0.4033587410259244
log_loss for k =  13 is 0.4173713719615923
```



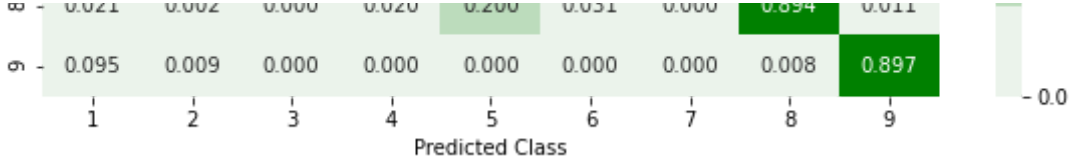

```
For values of best alpha =  3 The train log loss is: 0.16997650998100
303
For values of best alpha =  3 The cross validation log loss is: 0.337
9356450066001
For values of best alpha =  3 The test log loss is: 0.320859639693721
83
Number of misclassified points  8.187672493100276
---------------------------------------------------- Confusion matrix -
-----------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

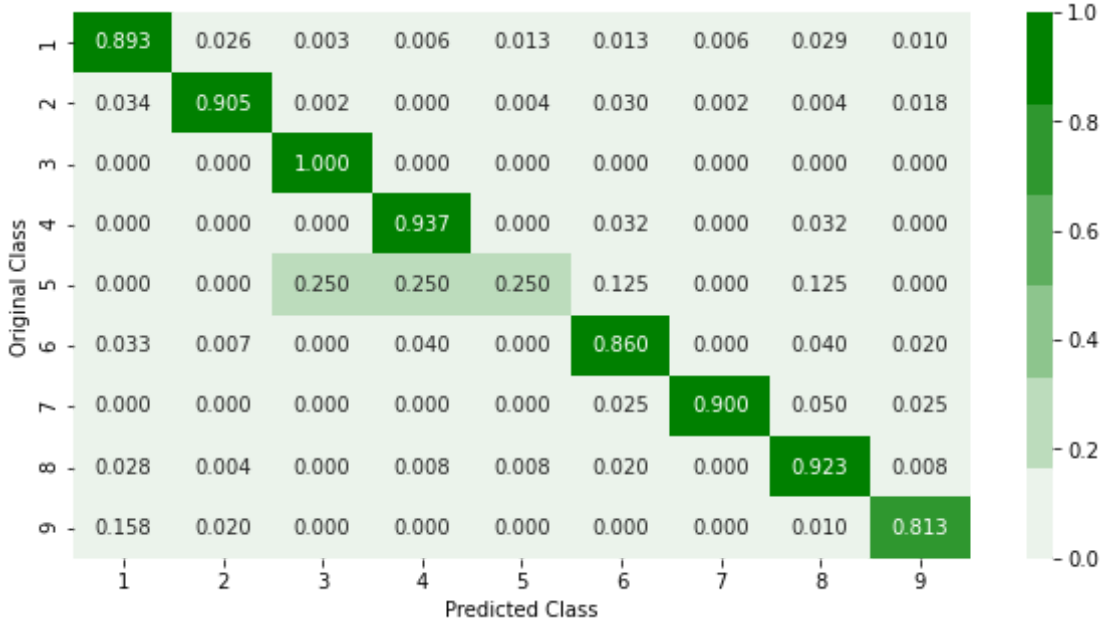
---------- Precision matrix -



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
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 [216]:
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
```

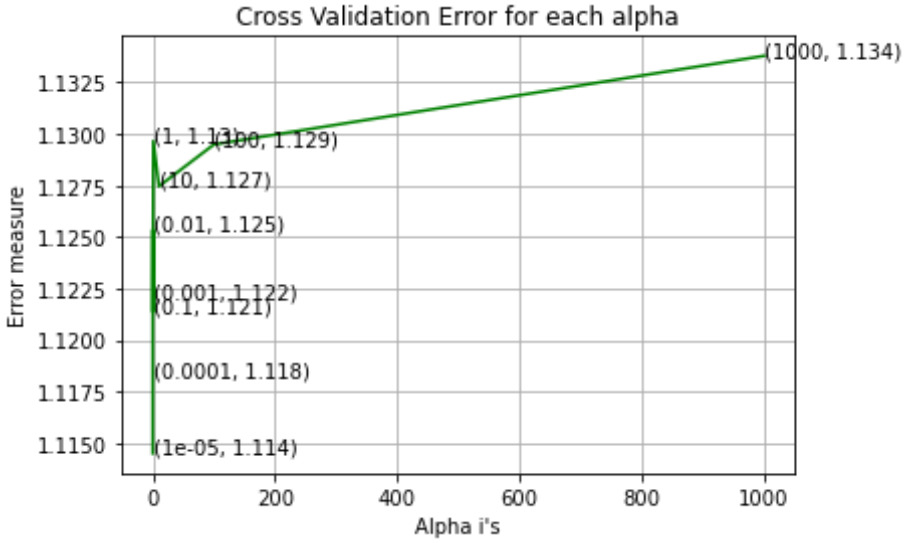
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balance
d')
    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=logistic
R.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_wei
ght='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=lo
gisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logistic
R.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

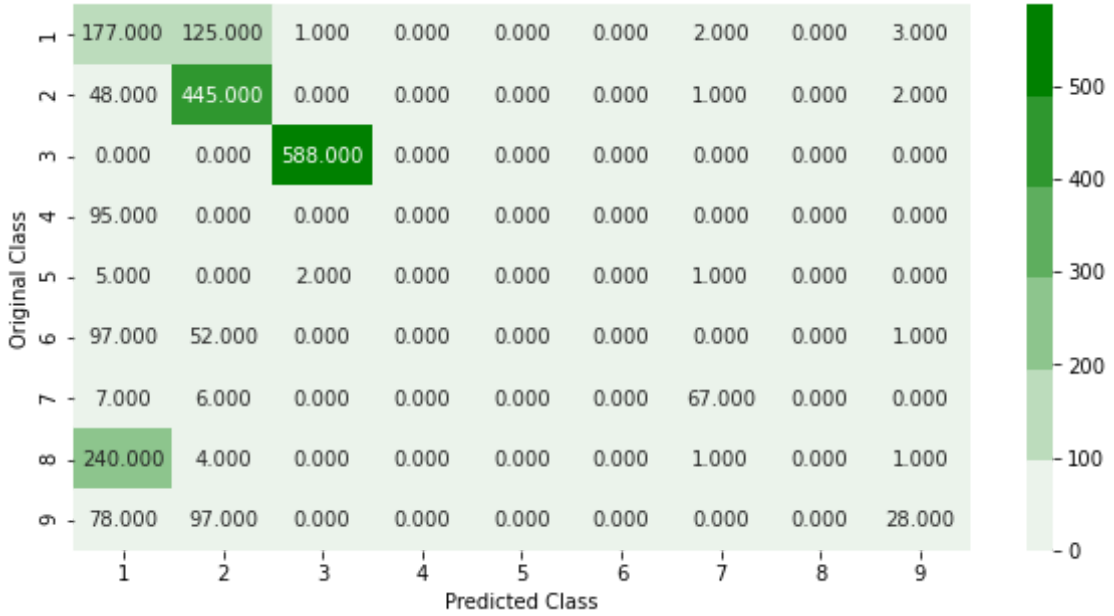
```
print ('log loss for test data',log_loss(y_test, predict_y, labels=logi
sticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.1144984166528842
log_loss for c =  0.0001 is 1.1182057650174002
log_loss for c =  0.001 is 1.1220028730847307
log_loss for c =  0.01 is 1.1253515718762572
log_loss for c =  0.1 is 1.1213632453448656
log_loss for c =  1 is 1.1296613792390513
log_loss for c =  10 is 1.1274568436210595
log_loss for c =  100 is 1.1294894530839814
log_loss for c =  1000 is 1.1337929311684898
```

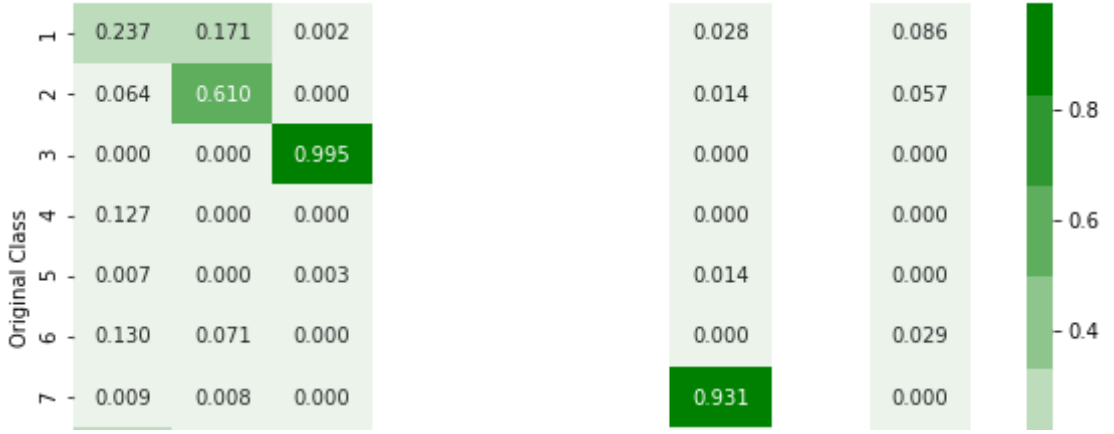



```
log loss for train data 1.113963894298774
log loss for cv data 1.1144984166528842
log loss for test data 1.1275932361592873
Number of misclassified points  39.97240110395584
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

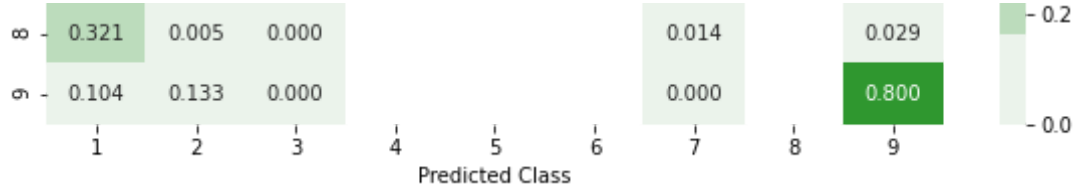
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



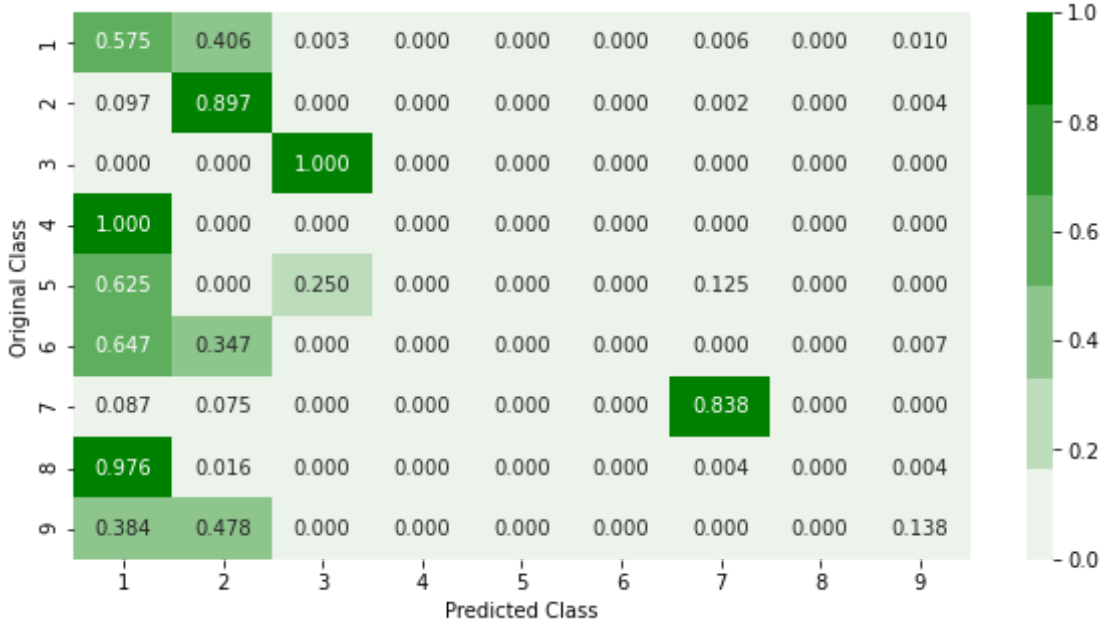
---------- Precision matrix -



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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




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

Random Forest Classifier

In [218]:
```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
```

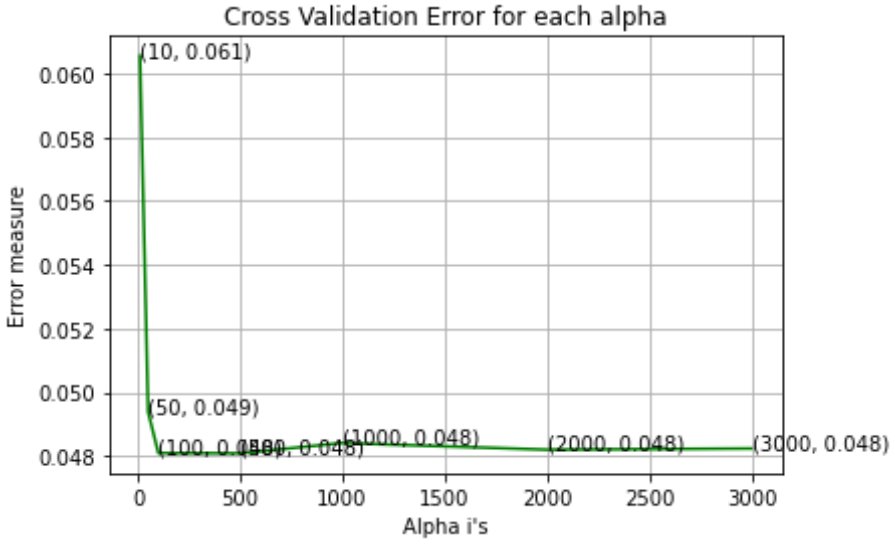
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=
-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_stat
e=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

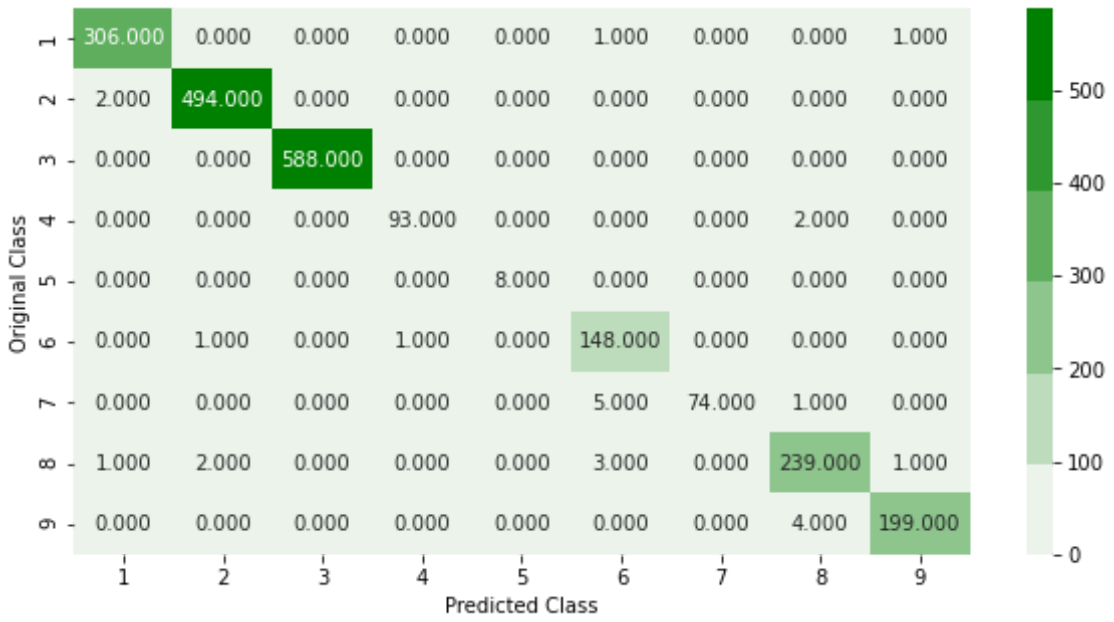
```
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y))
plot_confusion_matrix(y_test,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

```
log_loss for c =  10 is 0.06056252002324091
log_loss for c =  50 is 0.04937864548103391
log_loss for c =  100 is 0.048086847443930276
log_loss for c =  500 is 0.048080654560957385
log_loss for c =  1000 is 0.04840568186591602
log_loss for c =  2000 is 0.04819530449670029
log_loss for c =  3000 is 0.04823204459912968
```



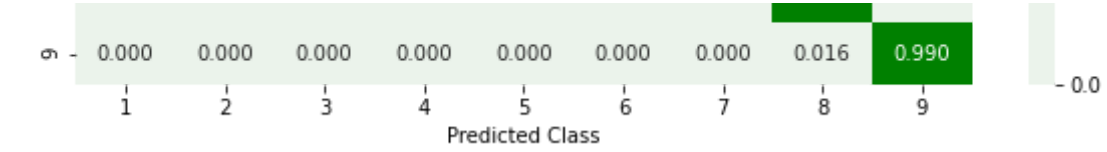

```
For values of best alpha =  500 The train log loss is: 0.016769498718
904104
For values of best alpha =  500 The cross validation log loss is: 0.0
48080654560957385
For values of best alpha =  500 The test log loss is: 0.0471762479840
4287
Number of misclassified points  1.1499540018399264
---------------------------------------------------- Confusion matrix -
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

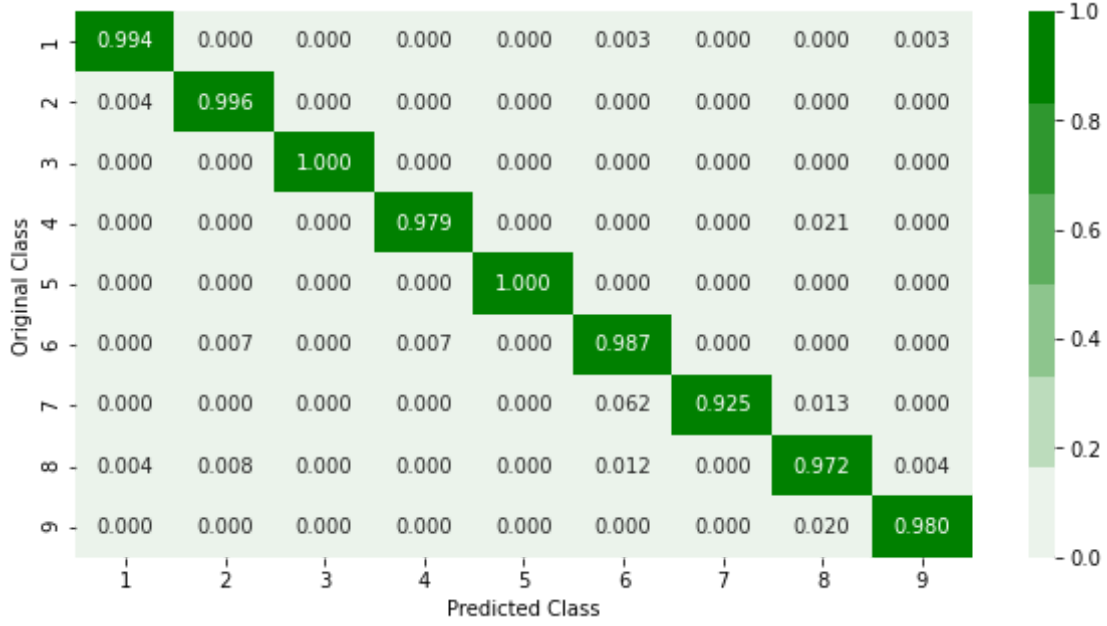


---------- Precision matrix -

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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




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

XGboost Classifier

In [221]:
```
alpha=[10,50,100]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

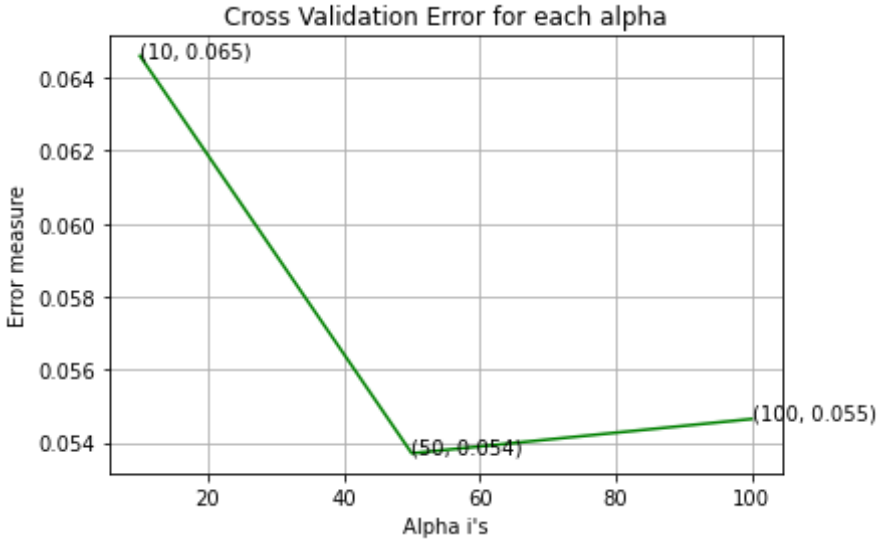
```
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

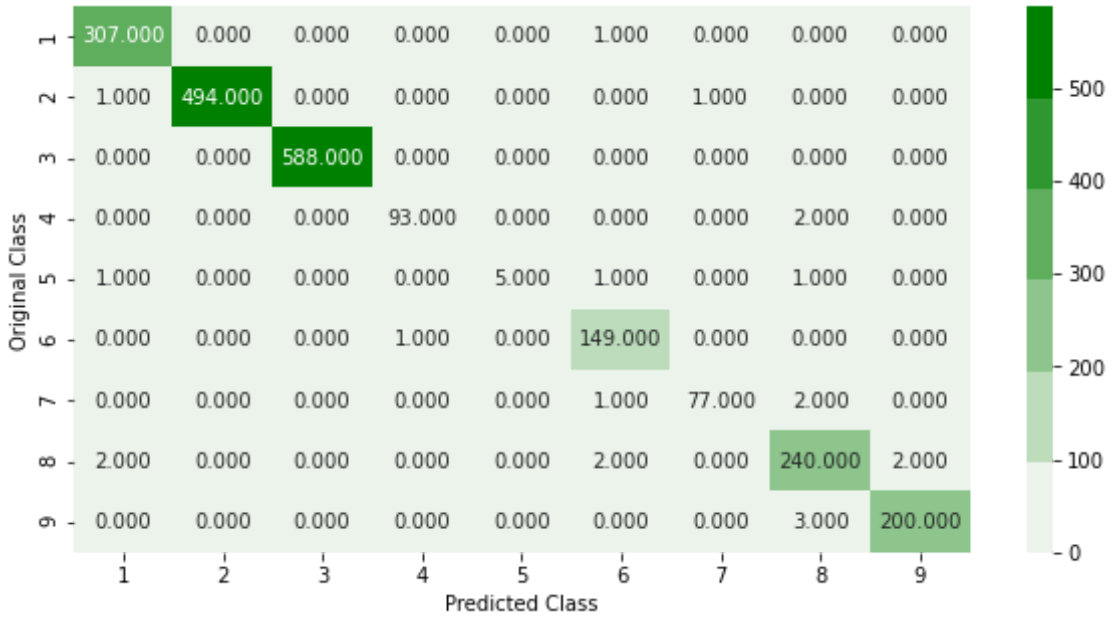
```
100%|██████████| 3/3 [15:58<00:00, 319.51s/it]
```

```
log_loss for c =  10 is 0.06460072610566713
log_loss for c =  50 is 0.053702954765675055
log_loss for c =  100 is 0.05464395516745456
```



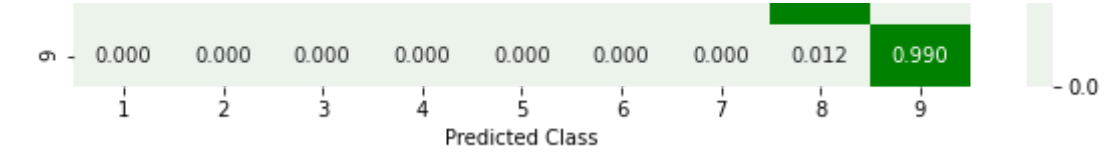

```
For values of best alpha =  50 The train log loss is: 0.0160066191629
27747
For values of best alpha =  50 The cross validation log loss is: 0.05
3702954765675055
For values of best alpha =  50 The test log loss is: 0.04806746765243
189
Number of misclassified points  0.9659613615455382
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



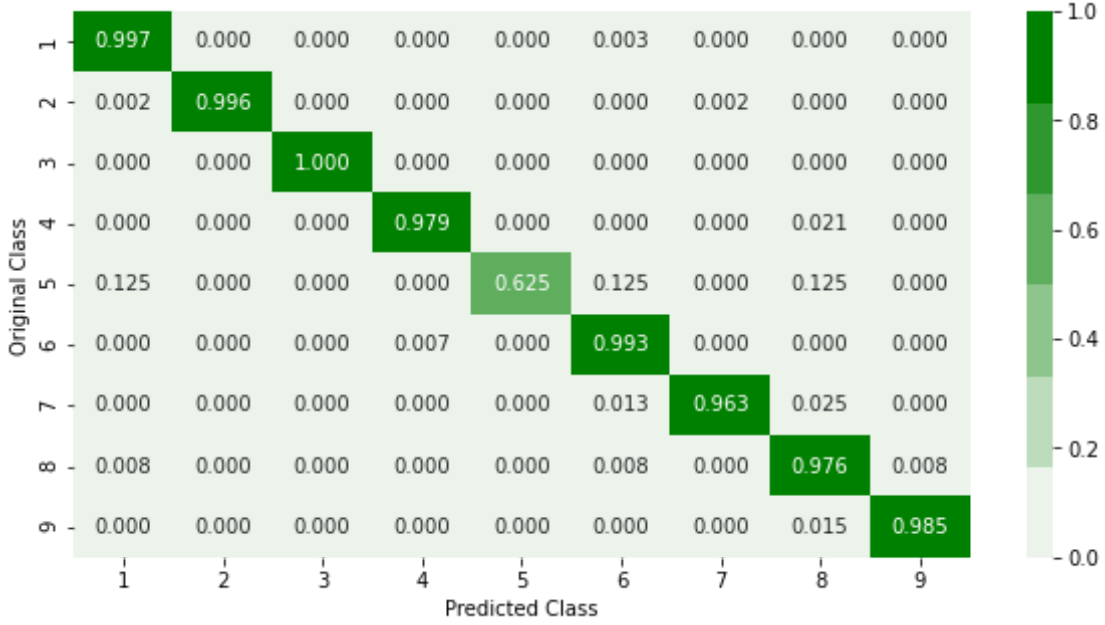
---------- Precision matrix -

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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.]

----------Byte-unigrams+Byte-bigrams ----------

Model	Test Loss	Miss-classification

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Model	Test Loss	Miss-classification
Random Model	2.52195	89.3 %
KNN	0.32085	8.18 %
Logistic Regression	1.1275	39.9 %
Random Forest Classifier	0.04717	1.149 %
XGBoost Classsifier	0.048067	0.96 %

2.Include pixel intensity features to improve the logloss

In [21]:
```
import array
def collect_img_asm():
    for asmfile in tqdm(os.listdir("asmFiles")[:10]):
        filename = asmfile.split('.')[0]
        file = codecs.open("asmFiles/" + asmfile, 'rb')
        filelen = os.path.getsize("asmFiles/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('asmimage/' + filename + '.png',reshaped)
```

In [22]:
```
collect_img_asm()
```

```
100%|██████████| 10/10 [00:27<00:00,  2.70s/it]
```

In [23]:
```
import cv2
imagefeatures = np.zeros((10, 200))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [17]:
```
from imageio import imwrite
```

As the size of asm files is 150 GB, to do processing faster making use of multi-processing with 20 sub processes

In [28]:
```
folder_1 ='a1'
folder_2 = 'a2'
folder_3 ='a3'
folder_4 ='a4'
folder_5 ='a5'
folder_6 = 'a6'
folder_7 ='a7'
folder_8 = 'a8'
folder_9 ='a9'
folder_10 ='a10'
folder_11 ='a11'
folder_12 = 'a12'
folder_13 ='a13'
folder_14 ='a14'
folder_15 ='a15'
folder_16 = 'a16'
folder_17 ='a17'
folder_18 = 'a18'
folder_19 ='a19'
folder_20 ='a20'

for i in [folder_1,folder_2,folder_3,folder_4,folder_5,folder_6,folder_
7,folder_8,folder_9,folder_10,
          folder_11,folder_12,folder_13,folder_14,folder_15,folder_16,f
older_17,folder_18,folder_19,folder_20]:
    if not os.path.isdir(i):
        os.makedirs(i)
```

In [30]:
```
source='asmFiles/'
files = os.listdir('asmFiles')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
#ID=df['Id'].tolist()
data=np.arange(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 20==0:
        shutil.move(source+files[data[i]],'a1')
    elif i%20==1:
        shutil.move(source+files[data[i]],'a2')
    elif i%20 ==2:
        shutil.move(source+files[data[i]],'a3')
    elif i%20 ==3:
        shutil.move(source+files[data[i]],'a4')
    elif i%20==4:
        shutil.move(source+files[data[i]],'a5')
    elif i%20==5:
        shutil.move(source+files[data[i]],'a6')
    elif i%20 ==6:
        shutil.move(source+files[data[i]],'a7')
    elif i%20 ==7:
        shutil.move(source+files[data[i]],'a8')
    elif i%20==8:
        shutil.move(source+files[data[i]],'a9')
    elif i%20==9:
        shutil.move(source+files[data[i]],'a10')
    elif i%20 ==10:
        shutil.move(source+files[data[i]],'a11')
    elif i%20 ==11:
        shutil.move(source+files[data[i]],'a12')
    elif i%20==12:
        shutil.move(source+files[data[i]],'a13')
    elif i%20==13:
        shutil.move(source+files[data[i]],'a14')
    elif i%20 ==14:
        shutil.move(source+files[data[i]],'a15')
    elif i%20 ==15:
        shutil.move(source+files[data[i]],'a16')
    elif i%20==16:
        shutil.move(source+files[data[i]],'a17')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    elif i%20==17:
        shutil.move(source+files[data[i]],'a18')
    elif i%20 ==18:
        shutil.move(source+files[data[i]],'a19')
    elif i%20 ==19:
        shutil.move(source+files[data[i]],'a20')
```

In [37]:
```
os.mkdir("a1_img")
os.mkdir("a2_img")
os.mkdir("a3_img")
os.mkdir("a4_img")
os.mkdir("a5_img")
os.mkdir("a6_img")
os.mkdir("a7_img")
os.mkdir("a8_img")
os.mkdir("a9_img")
os.mkdir("a10_img")
os.mkdir("a11_img")
os.mkdir("a12_img")
os.mkdir("a13_img")
os.mkdir("a14_img")
os.mkdir("a15_img")
os.mkdir("a16_img")
os.mkdir("a17_img")
os.mkdir("a18_img")
os.mkdir("a19_img")
os.mkdir("a20_img")
```

In [38]:
```
def ext_img_asm1():
    for asmfile in tqdm(os.listdir("a1")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a1/" + asmfile, 'rb')
        filelen = os.path.getsize("a1/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a1_img/' + filename + '.png',reshaped)

def ext_img_asm2():
    for asmfile in tqdm(os.listdir("a2")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a2/" + asmfile, 'rb')
        filelen = os.path.getsize("a2/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a2_img/' + filename + '.png',reshaped)

def ext_img_asm3():
    for asmfile in tqdm(os.listdir("a3")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a3/" + asmfile, 'rb')
        filelen = os.path.getsize("a3/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a3_img/' + filename + '.png',reshaped)

def ext_img_asm4():
    for asmfile in tqdm(os.listdir("a4")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a4/" + asmfile, 'rb')
        filelen = os.path.getsize("a4/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a4_img/' + filename + '.png',reshaped)

def ext_img_asm5():
    for asmfile in tqdm(os.listdir("a5")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a5/" + asmfile, 'rb')
        filelen = os.path.getsize("a5/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a5_img/' + filename + '.png',reshaped)

def ext_img_asm6():
    for asmfile in tqdm(os.listdir("a6")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a6/" + asmfile, 'rb')
        filelen = os.path.getsize("a6/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a6_img/' + filename + '.png',reshaped)

def ext_img_asm7():
    for asmfile in tqdm(os.listdir("a7")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a7/" + asmfile, 'rb')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        filelen = os.path.getsize("a7/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a7_img/' + filename + '.png',reshaped)

def ext_img_asm8():
    for asmfile in tqdm(os.listdir("a8")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a8/" + asmfile, 'rb')
        filelen = os.path.getsize("a8/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a8_img/' + filename + '.png',reshaped)

def ext_img_asm9():
    for asmfile in tqdm(os.listdir("a9")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a9/" + asmfile, 'rb')
        filelen = os.path.getsize("a9/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a9_img/' + filename + '.png',reshaped)

def ext_img_asm10():
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    for asmfile in tqdm(os.listdir("a10")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a10/" + asmfile, 'rb')
        filelen = os.path.getsize("a10/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a10_img/' + filename + '.png',reshaped)
```

In [39]:

```
def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present Sys
tem
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=ext_img_asm1)
    p2=Process(target=ext_img_asm2)
    p3=Process(target=ext_img_asm3)
    p4=Process(target=ext_img_asm4)
    p5=Process(target=ext_img_asm5)
    p6=Process(target=ext_img_asm6)
    p7=Process(target=ext_img_asm7)
    p8=Process(target=ext_img_asm8)
    p9=Process(target=ext_img_asm9)
    p10=Process(target=ext_img_asm10)

    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()

    p6.start()
    p7.start()
    p8.start()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    p9.start()
    p10.start()

    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()

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

```
100%|██████████| 544/544 [13:29<00:00,  1.49s/it]
100%|██████████| 544/544 [14:09<00:00,  1.56s/it]
100%|██████████| 544/544 [14:27<00:00,  1.60s/it]
100%|██████████| 544/544 [14:50<00:00,  1.64s/it]
100%|██████████| 544/544 [14:52<00:00,  1.64s/it]
100%|██████████| 543/543 [14:59<00:00,  1.66s/it]
100%|██████████| 544/544 [15:07<00:00,  1.67s/it]
100%|██████████| 544/544 [15:08<00:00,  1.67s/it]
100%|██████████| 543/543 [15:14<00:00,  1.68s/it]
100%|██████████| 544/544 [15:23<00:00,  1.70s/it]
```

In [40]:

```
def ext_img_asm11():
    for asmfile in tqdm(os.listdir("a11")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a11/" + asmfile, 'rb')
        filelen = os.path.getsize("a11/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a11_img/' + filename + '.png',reshaped)

def ext_img_asm12():
    for asmfile in tqdm(os.listdir("a12")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a12/" + asmfile, 'rb')
        filelen = os.path.getsize("a12/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a12_img/' + filename + '.png',reshaped)

def ext_img_asm13():
    for asmfile in tqdm(os.listdir("a13")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a13/" + asmfile, 'rb')
        filelen = os.path.getsize("a13/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a13_img/' + filename + '.png',reshaped)

def ext_img_asm14():
    for asmfile in tqdm(os.listdir("a14")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a14/" + asmfile, 'rb')
        filelen = os.path.getsize("a14/" + asmfile)
        width = int(filelen ** 0.5)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a14_img/' + filename + '.png',reshaped)

def ext_img_asm15():
    for asmfile in tqdm(os.listdir("a15")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a15/" + asmfile, 'rb')
        filelen = os.path.getsize("a15/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a15_img/' + filename + '.png',reshaped)

def ext_img_asm16():
    for asmfile in tqdm(os.listdir("a16")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a16/" + asmfile, 'rb')
        filelen = os.path.getsize("a16/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a16_img/' + filename + '.png',reshaped)

def ext_img_asm17():
    for asmfile in tqdm(os.listdir("a17")):
        filename = asmfile.split('.')[0]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        file = codecs.open("a17/" + asmfile, 'rb')
        filelen = os.path.getsize("a17/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a17_img/' + filename + '.png',reshaped)

def ext_img_asm18():
    for asmfile in tqdm(os.listdir("a18")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a18/" + asmfile, 'rb')
        filelen = os.path.getsize("a18/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a18_img/' + filename + '.png',reshaped)

def ext_img_asm19():
    for asmfile in tqdm(os.listdir("a19")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a19/" + asmfile, 'rb')
        filelen = os.path.getsize("a19/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a19_img/' + filename + '.png',reshaped)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
def ext_img_asm20():
    for asmfile in tqdm(os.listdir("a20")):
        filename = asmfile.split('.')[0]
        file = codecs.open("a20/" + asmfile, 'rb')
        filelen = os.path.getsize("a20/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        imwrite('a20_img/' + filename + '.png',reshaped)
```

In [41]:

```
def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present Sys
tem
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=ext_img_asm11)
    p2=Process(target=ext_img_asm12)
    p3=Process(target=ext_img_asm13)
    p4=Process(target=ext_img_asm14)
    p5=Process(target=ext_img_asm15)
    p6=Process(target=ext_img_asm16)
    p7=Process(target=ext_img_asm17)
    p8=Process(target=ext_img_asm18)
    p9=Process(target=ext_img_asm19)
    p10=Process(target=ext_img_asm20)

    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()

    p6.start()
    p7.start()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    p8.start()
    p9.start()
    p10.start()

    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

    p6.join()
    p7.join()
    p8.join()
    p9.join()
    p10.join()

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

```
100%|██████████| 543/543 [13:27<00:00,  1.49s/it]
100%|██████████| 543/543 [13:54<00:00,  1.54s/it]
100%|██████████| 543/543 [14:10<00:00,  1.57s/it]
100%|██████████| 543/543 [14:16<00:00,  1.58s/it]
100%|██████████| 543/543 [14:23<00:00,  1.59s/it]
100%|██████████| 543/543 [14:36<00:00,  1.61s/it]
100%|██████████| 543/543 [14:37<00:00,  1.62s/it]
100%|██████████| 543/543 [14:51<00:00,  1.64s/it]
100%|██████████| 543/543 [14:51<00:00,  1.64s/it]
100%|██████████| 543/543 [15:46<00:00,  1.74s/it]
```

In [75]:

```
import cv2
imagefeatures = np.zeros((10868, 300))

folders = ["a1_img","a2_img","a3_img","a4_img","a5_img","a6_img","a7_im
g","a8_img","a9_img","a10_img",
           "a11_img","a12_img","a13_img","a14_img","a15_img","a16_img",
"a17_img","a18_img","a19_img","a20_img"]
count=0
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
for folder in tqdm(folders):
    for asmfile in os.listdir(folder):
        img = cv2.imread(folder+"/" + asmfile.split('.')[0] + '.png')
        img_arr = img.flatten()[:300]
        imagefeatures[count, :] += img_arr
        count+=1
```

```
100%|██████████| 20/20 [36:35<00:00, 109.80s/it]
```

In [150]:
```
imagefeatures
```

Out[150]:
```
array([[46., 46., 46., ..., 45., 45., 45.],
       [72., 72., 72., ..., 45., 45., 45.],
       [72., 72., 72., ..., 45., 45., 45.],
       ...,
       [46., 46., 46., ..., 45., 45., 45.],
       [72., 72., 72., ..., 45., 45., 45.],
       [46., 46., 46., ..., 45., 45., 45.]])
```

In [151]:
```
img_features = normalize(imagefeatures[:,:180],axis=0)
img_features
```

Out[151]:
```
array([[0.00656033, 0.00656033, 0.00656033, ..., 0.00925827, 0.0092582
7,
        0.00925827],
       [0.01026834, 0.01026834, 0.01026834, ..., 0.00968889, 0.0096888
9,
        0.00968889],
       [0.01026834, 0.01026834, 0.01026834, ..., 0.00968889, 0.0096888
9,
        0.00968889],
       ...,
       [0.00656033, 0.00656033, 0.00656033, ..., 0.00925827, 0.0092582
7,
        0.00925827],
       [0.01026834, 0.01026834, 0.01026834, ..., 0.00968889, 0.0096888
9,
        0.00968889],
       [0.00656033, 0.00656033, 0.00656033, ..., 0.00925827, 0.0092582
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
       7,
              0.00925827]])
```

In [93]:
```
cleaned_id=[]
for folder in folders:
    files = os.listdir(folder)
    for file in files:
        cleaned_id.append(file.split(".")[0])

id1 = pd.DataFrame(cleaned_id,columns=['ID'])

#shuffled_id=pd.DataFrame(cleaned_id,columns=['ID'])
data = pd.read_csv("asm_with_size.csv")
id_class = data[['ID','Class']]

asm_suffled_id_class = pd.merge(id1,id_class,on='ID')
asm_suffled_id_class.head(3)
```

Out[93]:

	ID	Class
0	itwvmenE3zjuNpJF0lR8	2
1	IABzVohOWJZtRQXrcdC4	1
2	0Hrfce4X5YGESJPjl9uL	1

In [152]:
```
pixel_intensity_df = pd.DataFrame(img_features)
```

In [153]:
```
pix_intensity=pd.concat([asm_suffled_id_class['ID'],pixel_intensity_df,
asm_suffled_id_class['Class']],axis=1)
pix_intensity.head(3)
```

Out[153]:

	ID	0	1	2	3	4	5	6
0	itwvmenE3zjuNpJF0lR8	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927
1	IABzVohOWJZtRQXrcdC4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320
2	0Hrfce4X5YGESJPjl9uL	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

3 rows × 182 columns

In [ ]:
```
pix_intensity.to_csv("pixel_intensity_features.csv")
```

Train Test Split

In [13]:
```
data_asm = pd.read_csv("pixel_intensity_features.csv")
asm_y = data_asm['Class']
asm_x = data_asm.drop(['ID','Class'], axis=1)
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm
_x,asm_y ,stratify=asm_y,test_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train
_asm, y_train_asm,stratify=y_train_asm,test_size=0.20)
```

KNN

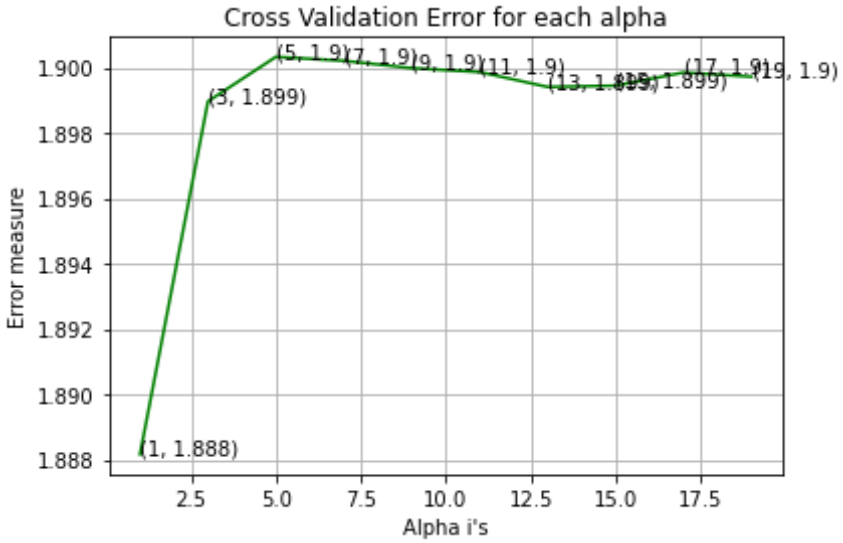
In [5]:
```
%matplotlib inline
alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cf
l.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

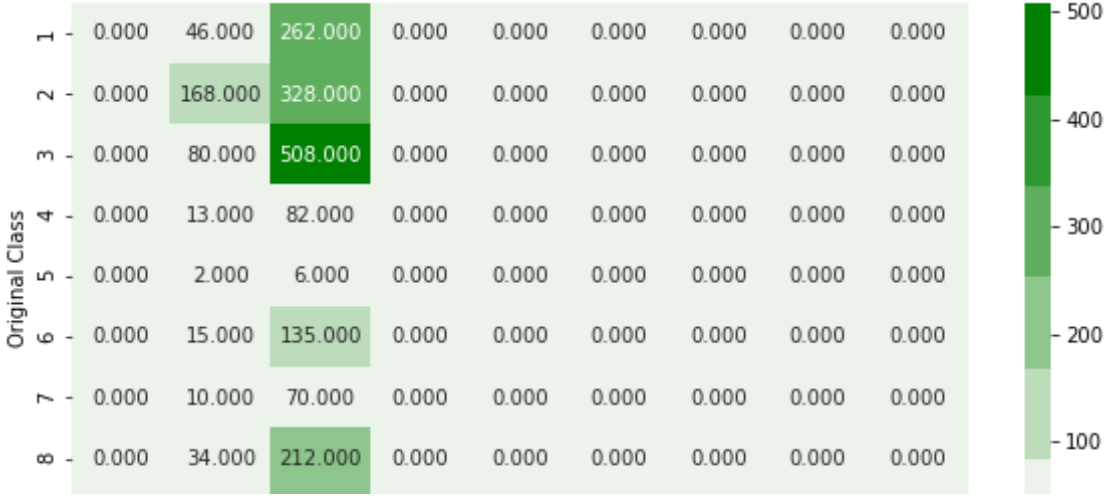
```
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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


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

```
log_loss for k =  1 is 1.8881626568447534
log_loss for k =  3 is 1.898959674184379
log_loss for k =  5 is 1.9003234537755396
log_loss for k =  7 is 1.9002031009181086
log_loss for k =  9 is 1.8999775687523917
log_loss for k =  11 is 1.89985774199873
log_loss for k =  13 is 1.899401917370922
log_loss for k =  15 is 1.899441088783216
log_loss for k =  17 is 1.8998446215592606
log_loss for k =  19 is 1.8997155629334248
```

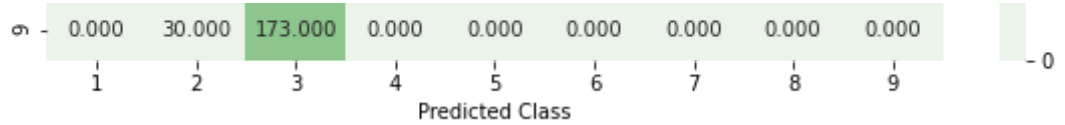
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



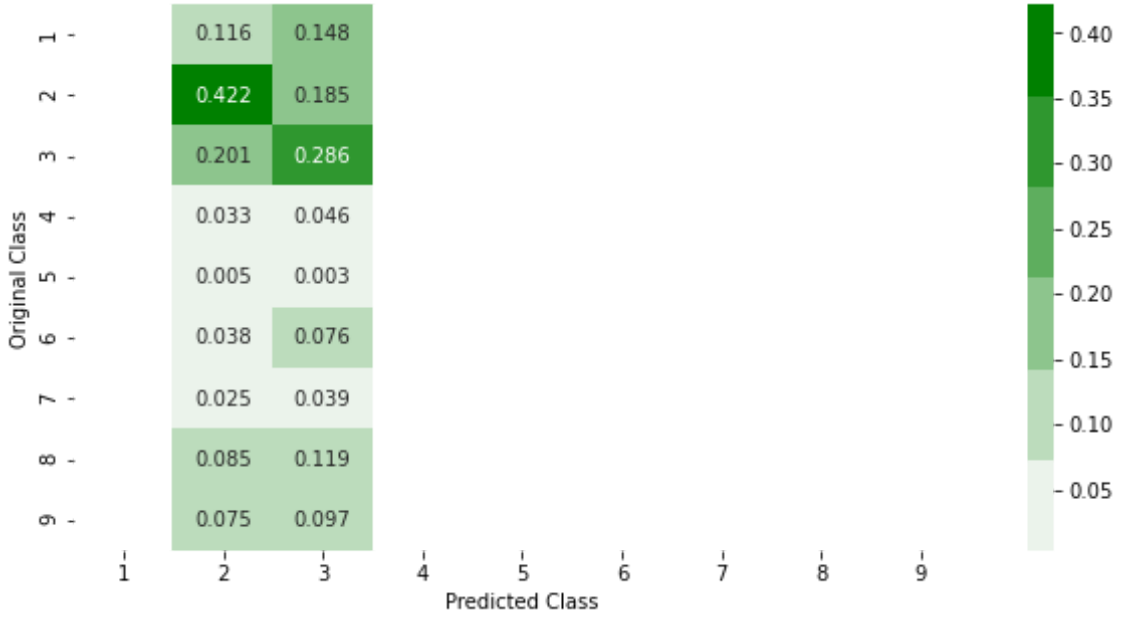

```
log loss for train data 1.8202921958536307
log loss for cv data 1.8881626568447534
log loss for test data 1.8894555078248763
Number of misclassified points  68.90524379024839
-------------------------------------------------- Confusion matrix -
--------------------------------------------------
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



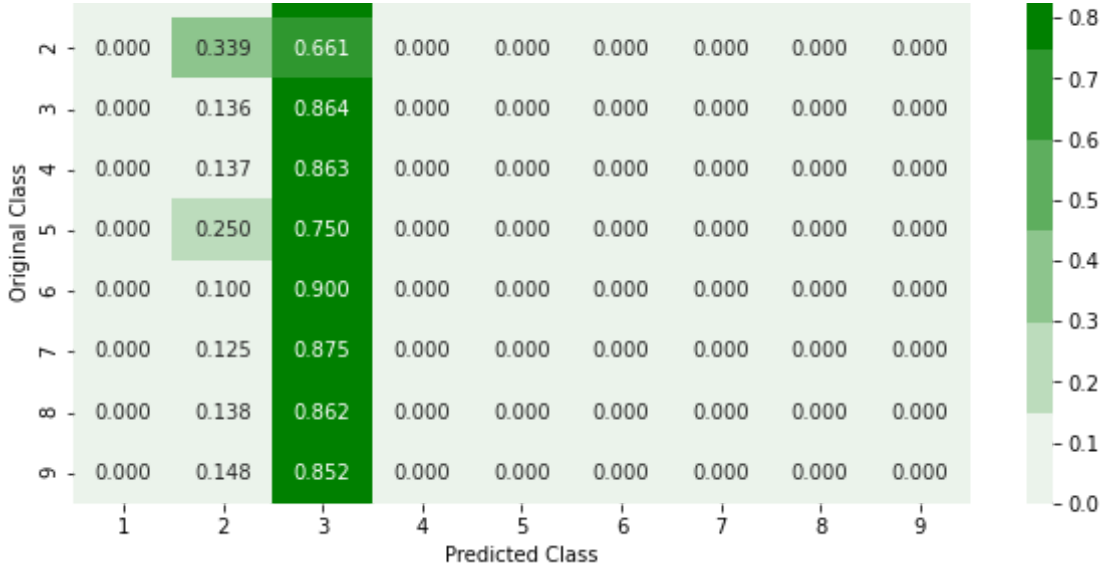
---------- Precision matrix -



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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API
PDFCROWD




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

Logistic Regression

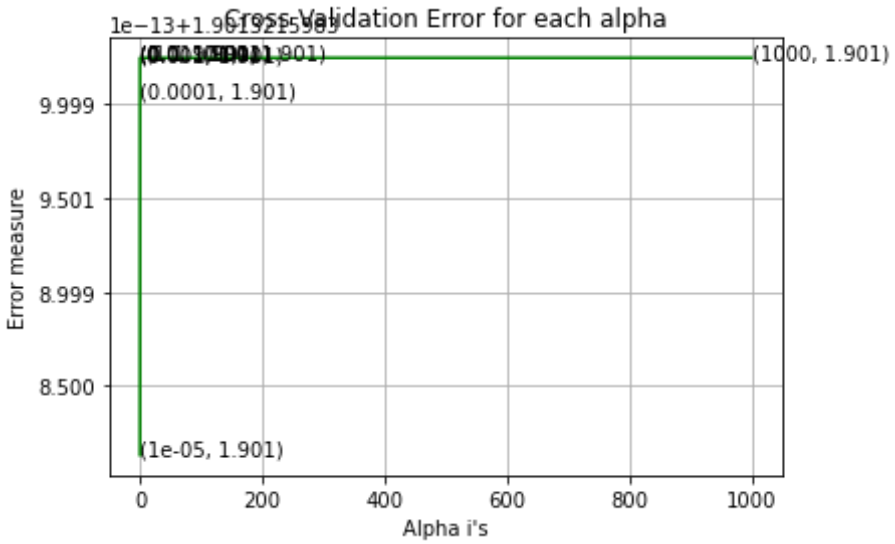
In [6]:
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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_arra
y[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_wei
ght='balanced')
logisticR.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

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

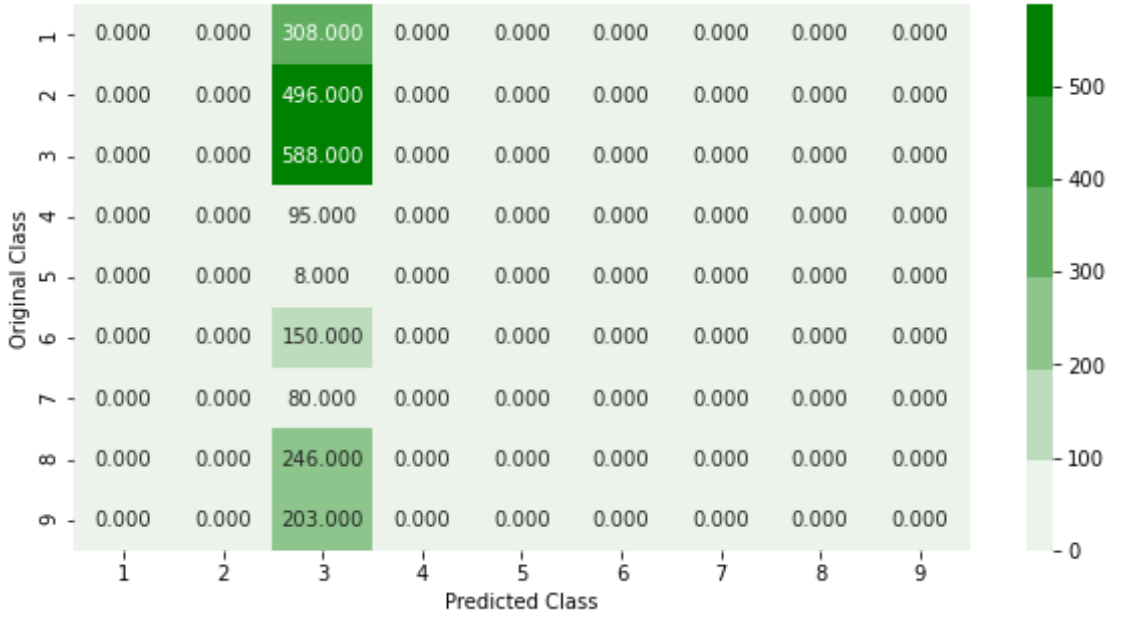
```
log_loss for c =  1e-05 is 1.901321598287813
log_loss for c =  0.0001 is 1.9013215982880036
log_loss for c =  0.001 is 1.9013215982880225
log_loss for c =  0.01 is 1.9013215982880245
log_loss for c =  0.1 is 1.9013215982880245
log_loss for c =  1 is 1.9013215982880245
log_loss for c =  10 is 1.9013215982880245
log_loss for c =  100 is 1.9013215982880245
log_loss for c =  1000 is 1.9013215982880245
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

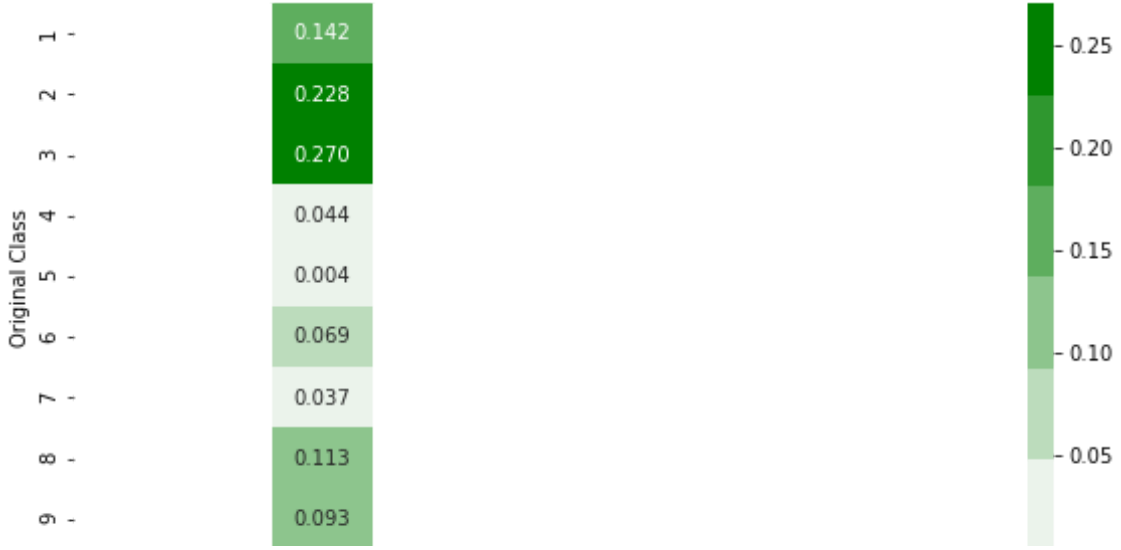



```
log loss for train data 1.898768518875549
log loss for cv data 1.901321598287813
log loss for test data 1.8998128379762445
Number of misclassified points  72.95308187672494
-------------------------------------------------- Confusion matrix -
--------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

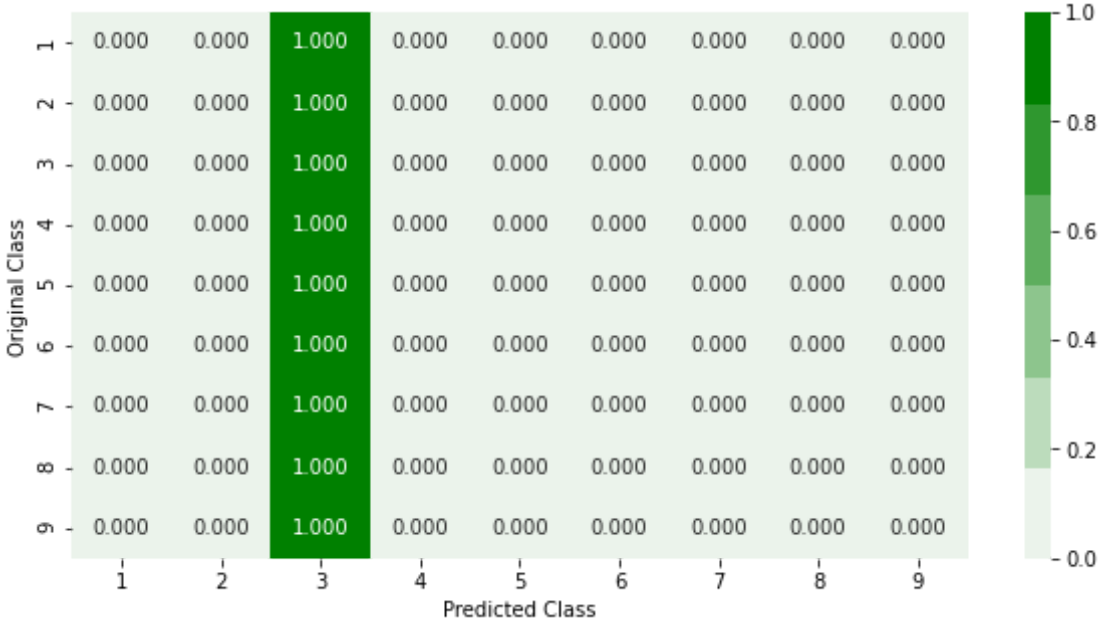


---------- Precision matrix -



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD


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




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Random Forest Classifier

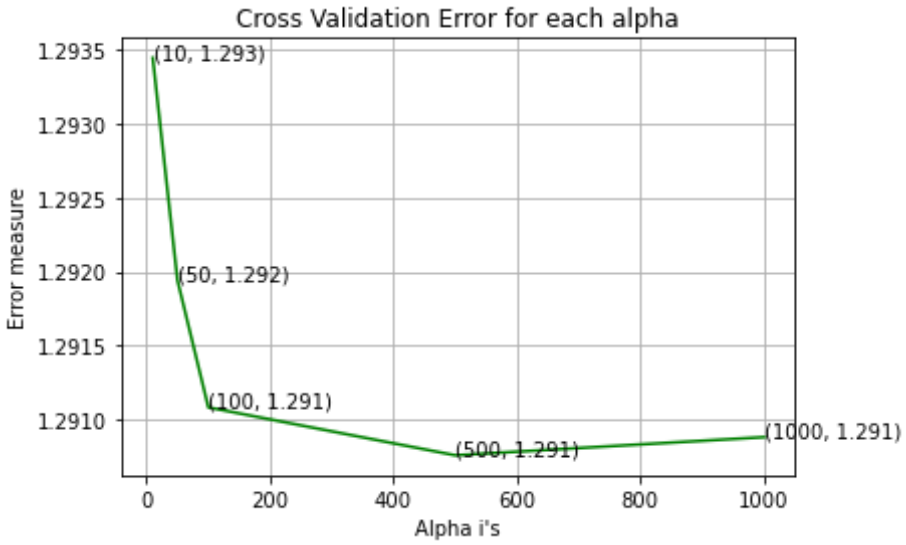
In [7]:
```
alpha=[10,50,100,500,1000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=
-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cf
l.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

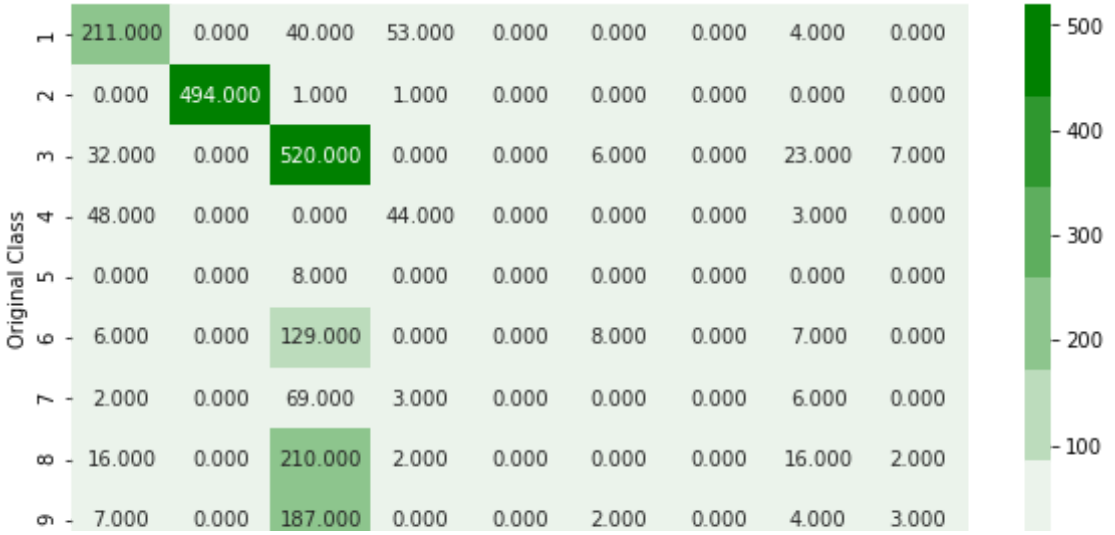
```
plt.ylabel("Error measure")
plt.show()

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

```
log_loss for c =  10 is 1.2934470464647505
log_loss for c =  50 is 1.2919419424851615
log_loss for c =  100 is 1.291081390949035
log_loss for c =  500 is 1.290756878863674
log_loss for c =  1000 is 1.2908798814619318
```

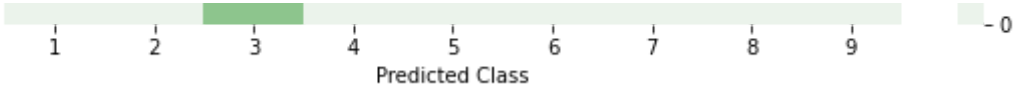
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



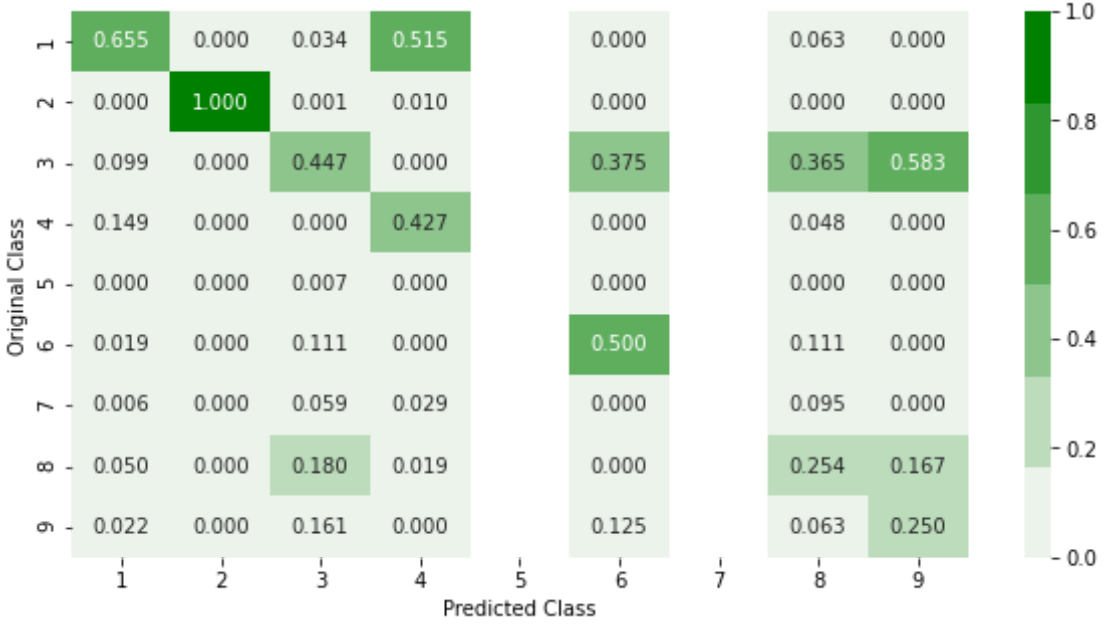

```
log loss for train data 0.8746647312394111
log loss for cv data 1.290756878863674
log loss for test data 1.2756643707323472
Number of misclassified points  40.38638454461822
---------------------------------------------------- Confusion matrix -
-----------------------------------------------------
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



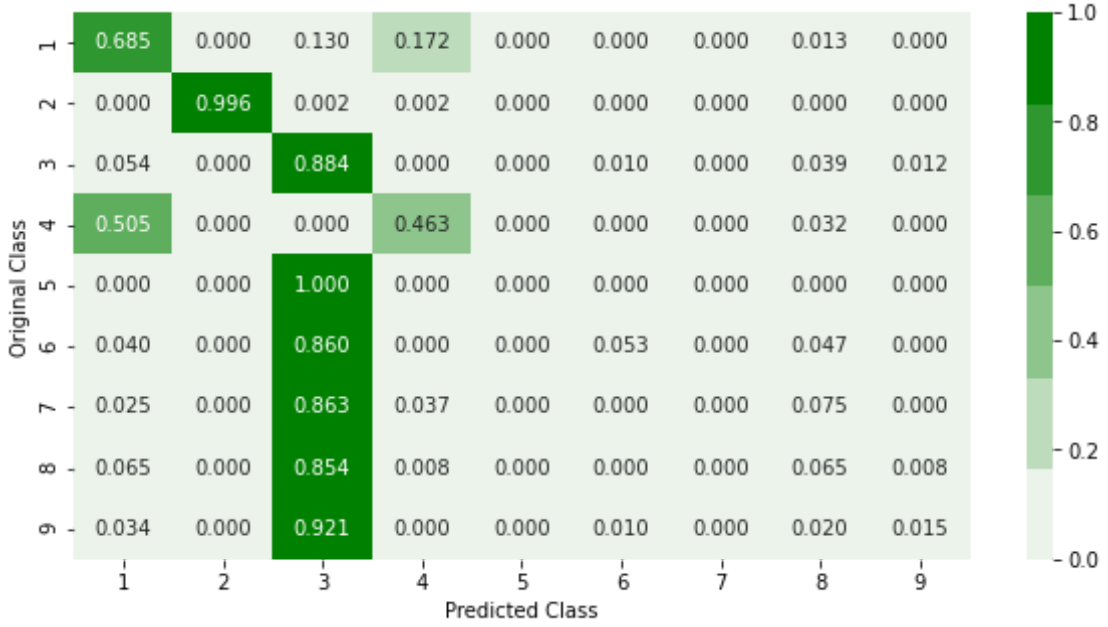
---------- Precision matrix -



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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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




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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

XBGBoost Classification

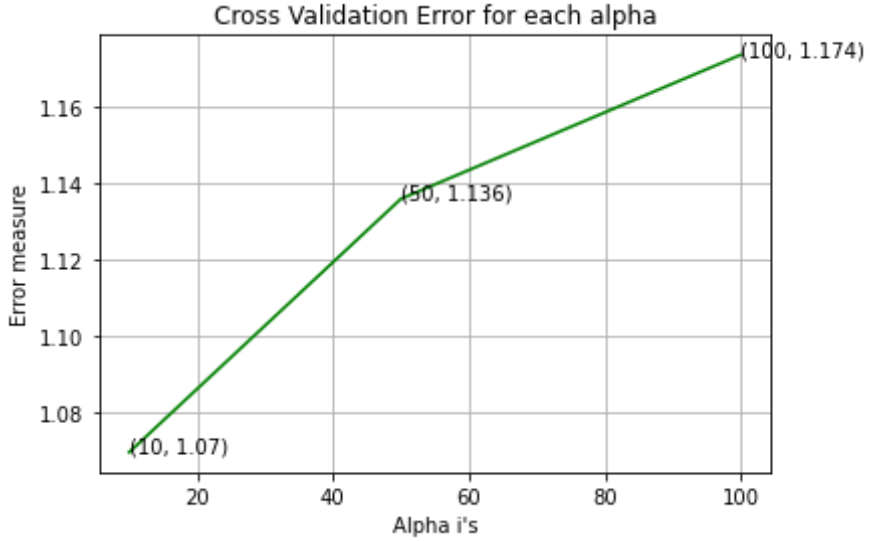
In [18]:
```
alpha=[10,50,100]
cv_log_error_array=[]
for i in tqdm(alpha):
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_cf
l.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
100%|██████████| 3/3 [05:04<00:00, 101.43s/it]
```

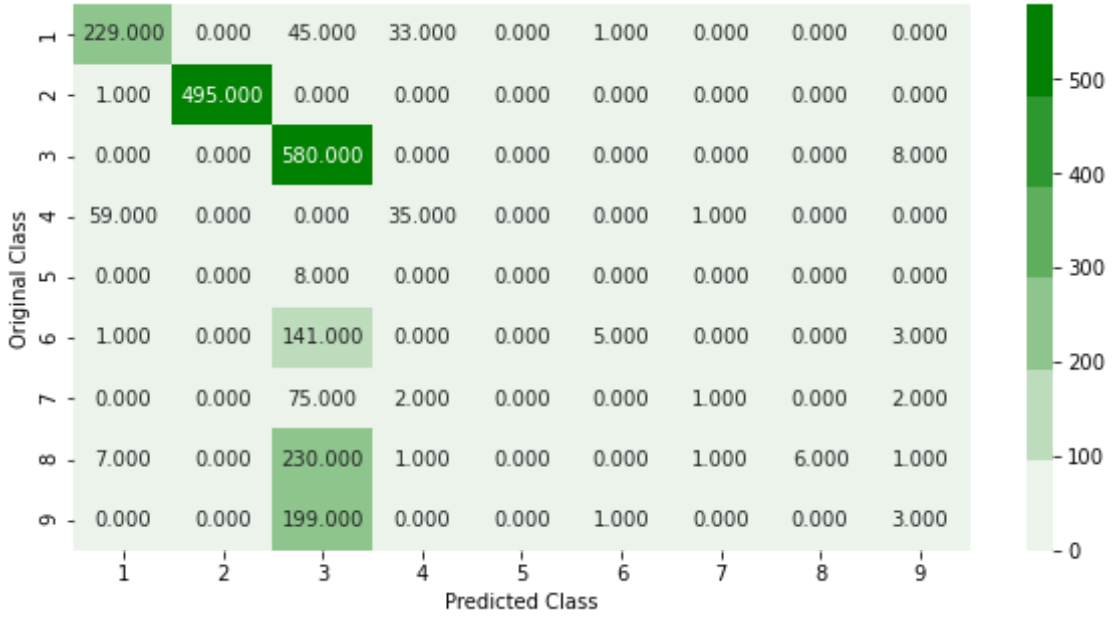
```
log_loss for c =  10 is 1.0695981210235306
log_loss for c =  50 is 1.1359931478870404
log_loss for c =  100 is 1.1737161578471935
```



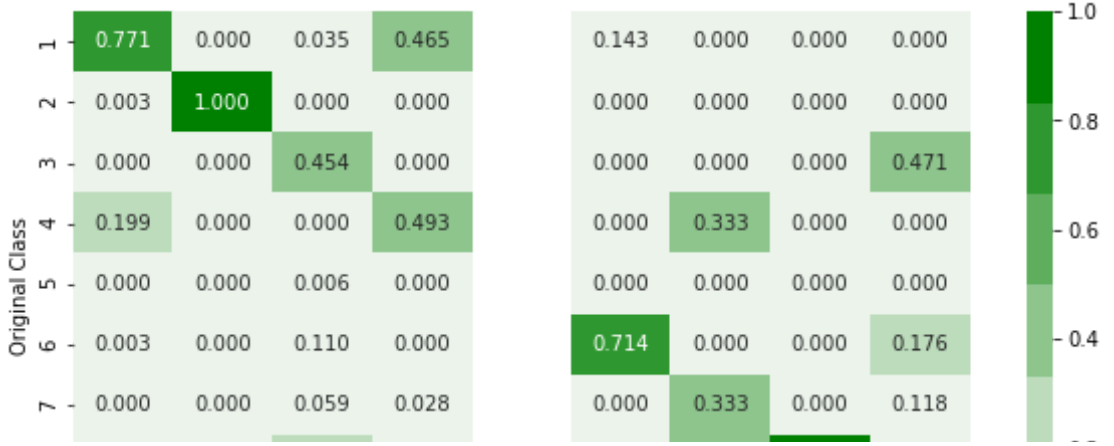

```
For values of best alpha =  10 The train log loss is: 1.0064002364698
77
For values of best alpha =  10 The cross validation log loss is: 1.06
95981210235306
For values of best alpha =  10 The test log loss is: 1.05446014807504
66
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
Number of misclassified points  37.71849126034959
------------------------------------------------------ Confusion matrix -
------------------------------------------------------
```

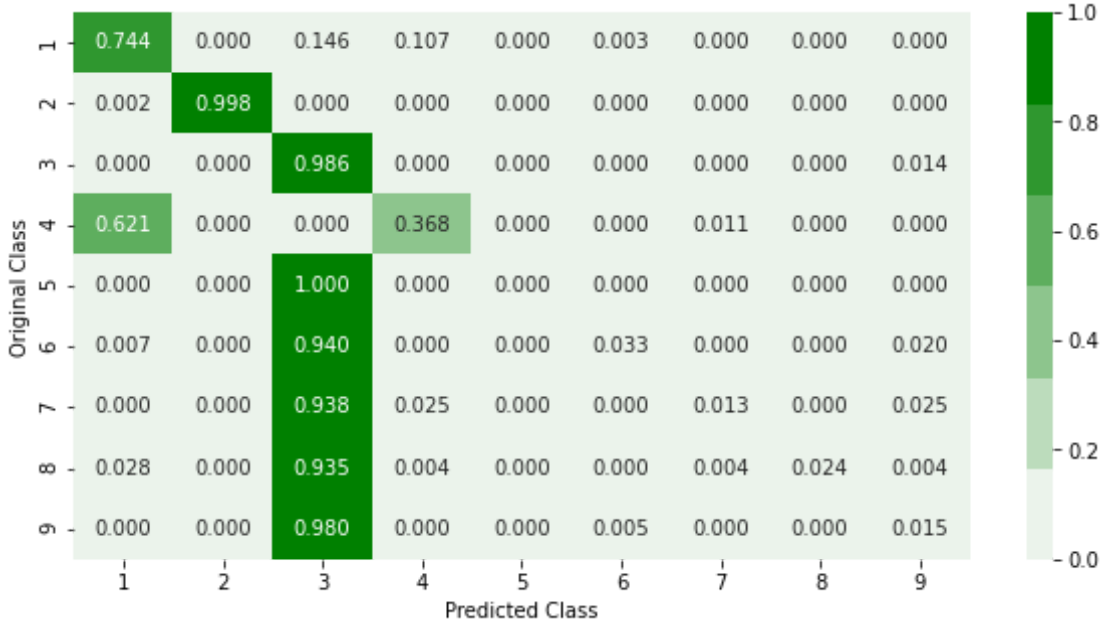



```
------------------------------------------------------ Precision matrix -
------------------------------------------------------
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD


```
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.]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Model	Test Loss	Miss-classification
KNN	1.8894	68.9 %
Logistic Regression	1.8998	72.9 %
Random Forest Classifier	1.284	40.7 %
XGBoost Classsifier	1.0544	37.7 %

Byte-bigrams + Pixcel-intensity + ASM-unigrams

In [16]:
```
data1 = pd.read_csv("top_bigrams_with_id_class.csv")
data2=pd.read_csv("pixel_intensity_features.csv")
data3=pd.read_csv("asmoutputfile.csv")
data = pd.merge(data2,data3,on="ID")
```

In [17]:
```
data = pd.merge(data,data1,on="ID")
data = data.drop("Unnamed: 0_x",axis=1)
data.head(3)
```

Out[17]:

	ID	0	1	2	3	4	5	6
0	itwvmenE3zjuNpJF0lR8	0.006560	0.006560	0.006560	0.013504	0.013504	0.013504	0.012927
1	IABzVohOWJZtRQXrcdC4	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320
2	0Hrfce4X5YGESJPjl9uL	0.010268	0.010268	0.010268	0.008033	0.008033	0.008033	0.008320

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

3 rows × 535 columns

In [4]:
```
data_y = data['Class_y']
# split the data into test and train by maintaining same distribution o
f output varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(data.drop(['ID','Cl
ass_y'], axis=1), data_y,stratify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,strati
fy=y_train,test_size=0.20)
```

In [5]:
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

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

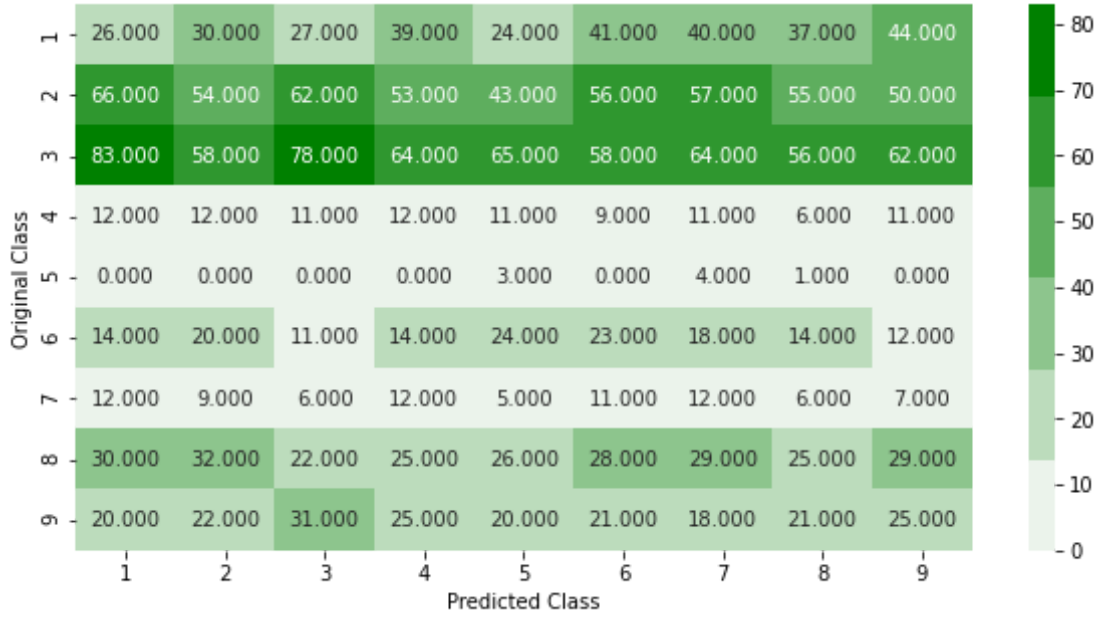
Random Model

In [11]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

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

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

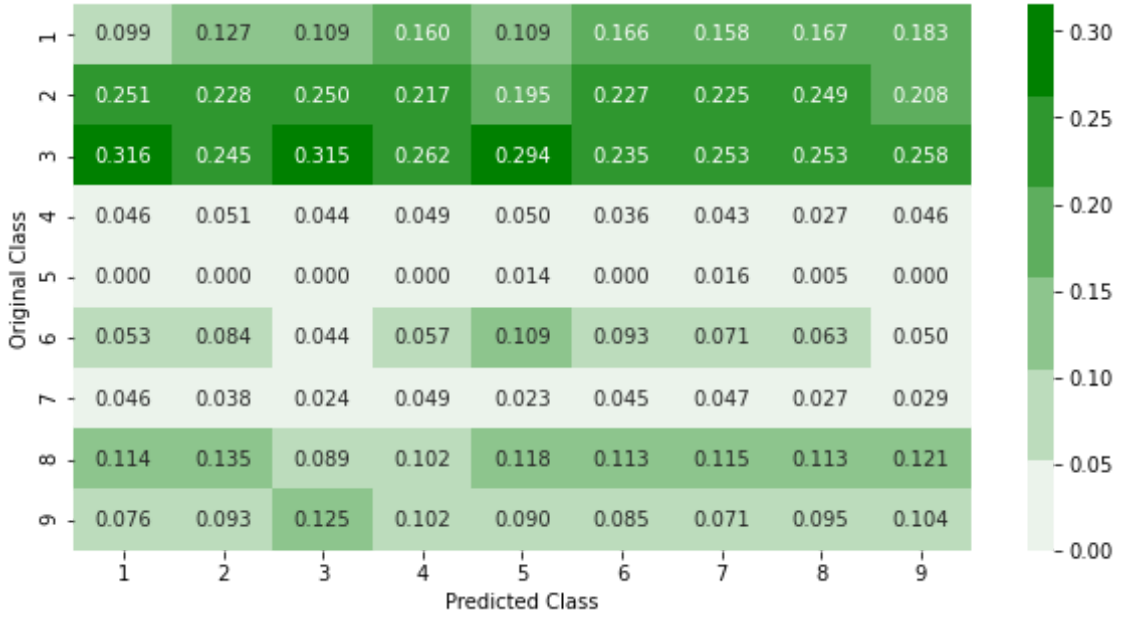
```
_cv,cv_predicted_y, eps=1e-15))


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

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

```
Log loss on Cross Validation Data using Random Model 2.48149395183525
15
Log loss on Test Data using Random Model 2.460083420086678
Number of misclassified points  88.13247470101196
------------------------------------------------------ Confusion matrix -
------------------------------------------------------
```



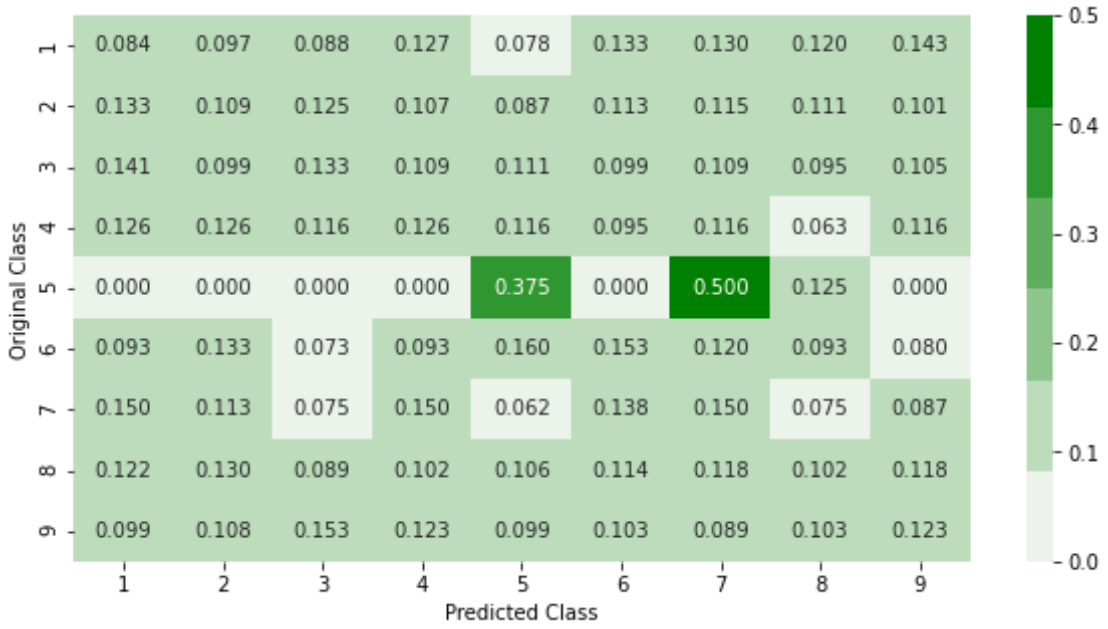
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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



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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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

KNN

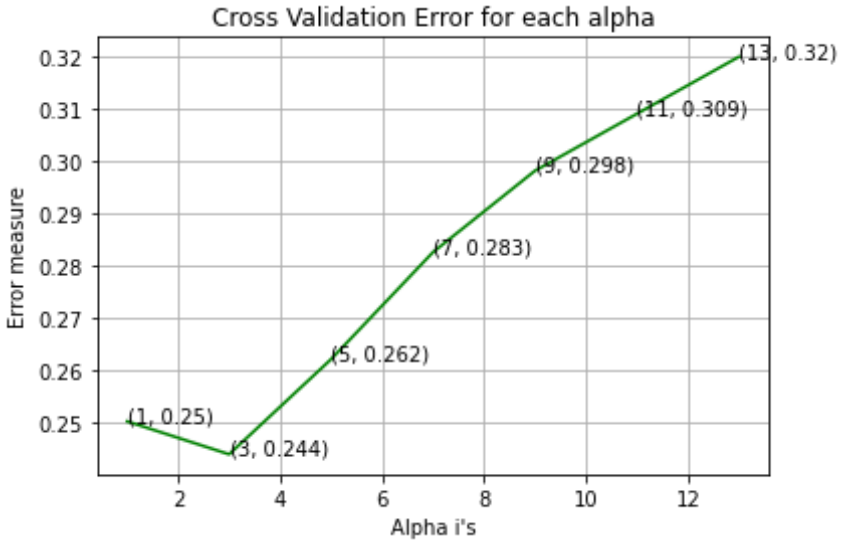
In [12]:
```
alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.cl
asses_, eps=1e-15))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

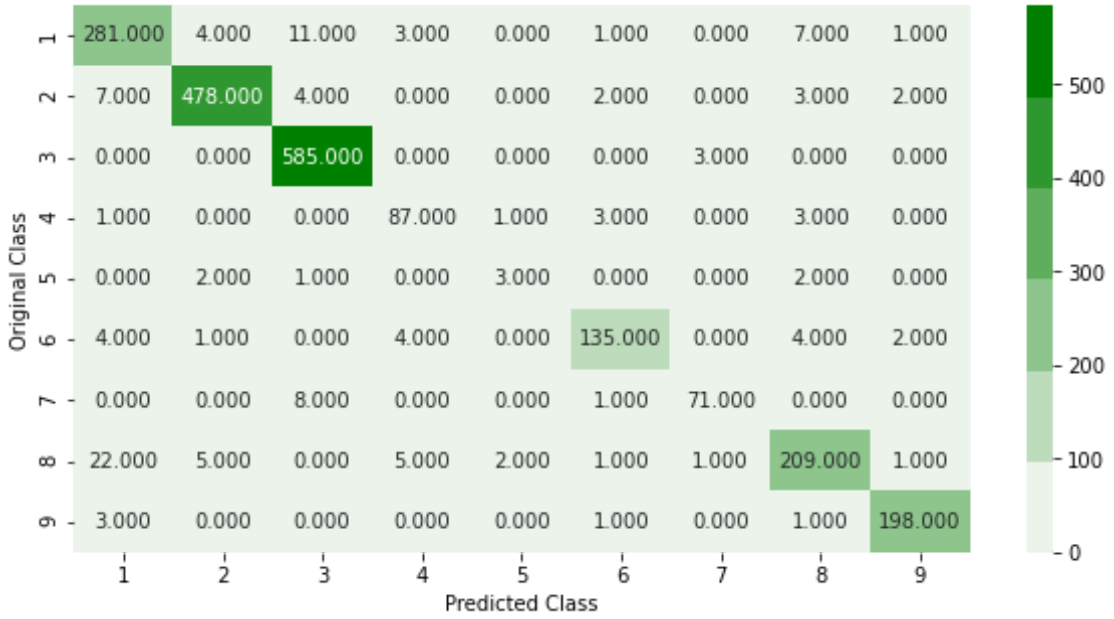
```
log_loss for k =  1 is 0.25030722529129945
log_loss for k =  3 is 0.24402425943122794
log_loss for k =  5 is 0.2621436788417304
log_loss for k =  7 is 0.2825623579821087
log_loss for k =  9 is 0.29808156246588713
log_loss for k =  11 is 0.308950713626348
log_loss for k =  13 is 0.3198189733808239
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



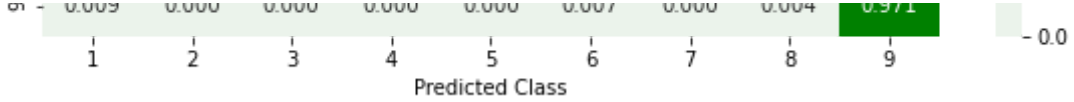

```
For values of best alpha =  3 The train log loss is: 0.11789504046501
932
For values of best alpha =  3 The cross validation log loss is: 0.244
02425943122794
For values of best alpha =  3 The test log loss is: 0.234087890217850
18
Number of misclassified points  5.841766329346826
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

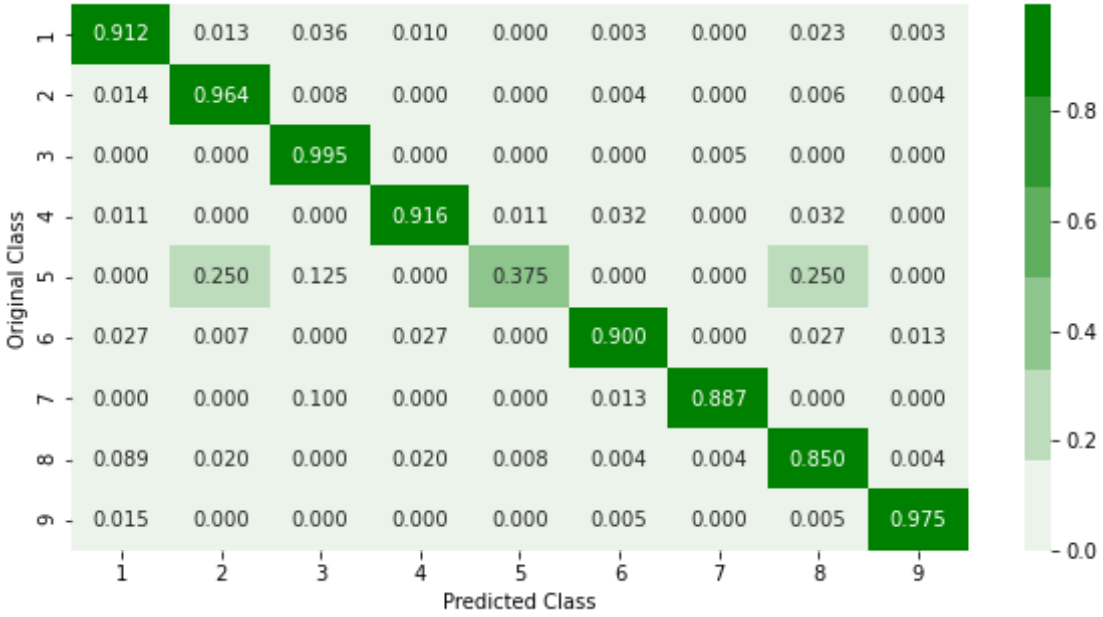


---------- Precision matrix -

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
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 [241]:
```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
```

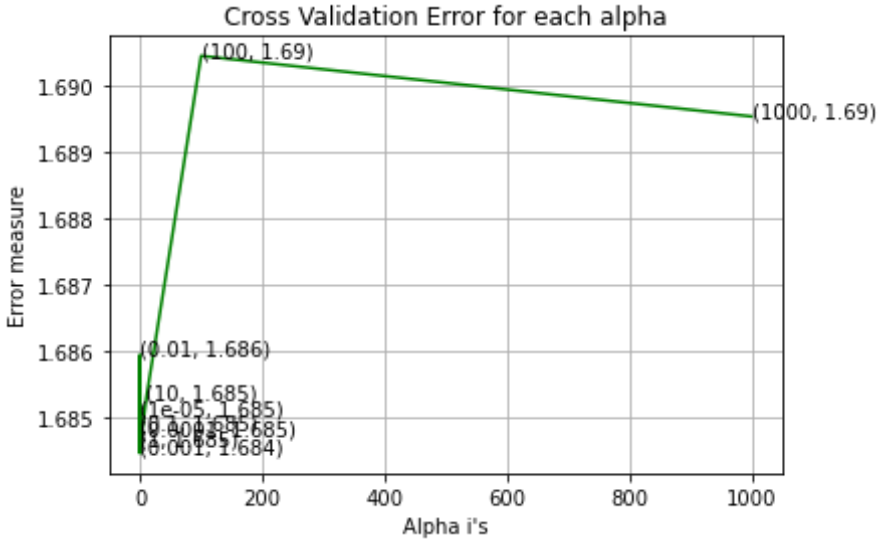
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balance
d')
    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=logistic
R.classes_, eps=1e-15))

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

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_wei
ght='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=lo
gisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logistic
R.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

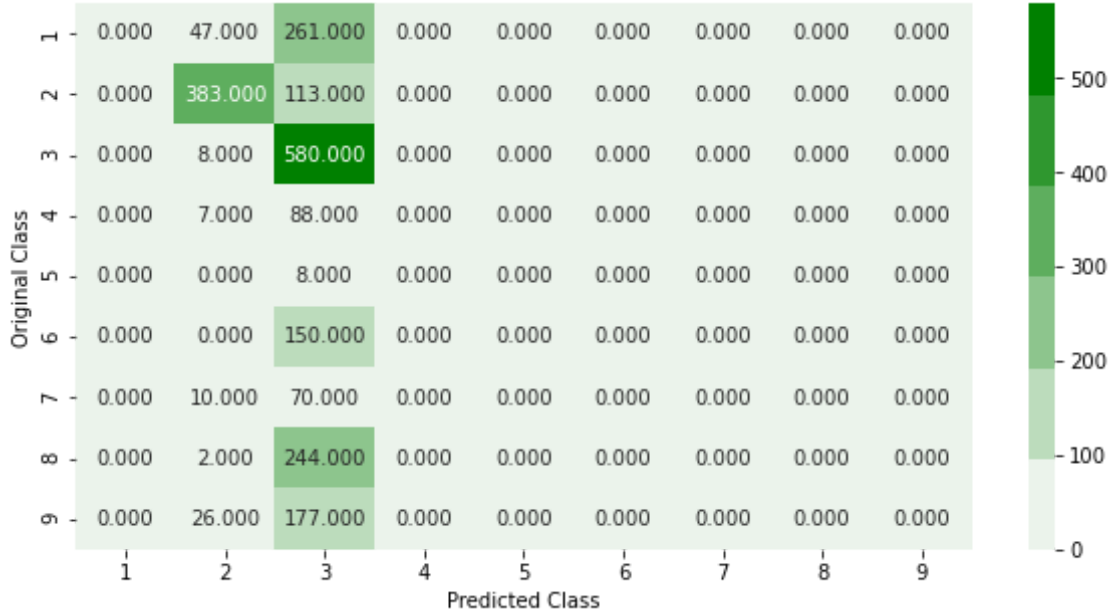
```
print ('log loss for test data',log_loss(y_test, predict_y, labels=logi
sticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.6850273680611285
log_loss for c =  0.0001 is 1.6847393512849764
log_loss for c =  0.001 is 1.6844496644460027
log_loss for c =  0.01 is 1.685936606072852
log_loss for c =  0.1 is 1.6848180272693278
log_loss for c =  1 is 1.6845907836834837
log_loss for c =  10 is 1.6852661619964895
log_loss for c =  100 is 1.690443056465867
log_loss for c =  1000 is 1.6895290880394054
```

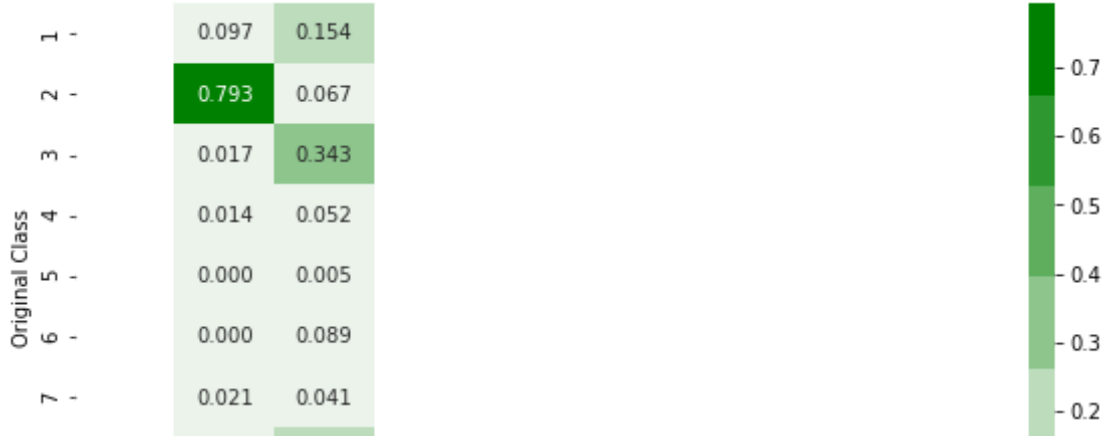



```
log loss for train data 1.6833169219195352
log loss for cv data 1.6844496644460027
log loss for test data 1.6743302512404463
Number of misclassified points  55.703771849126035
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

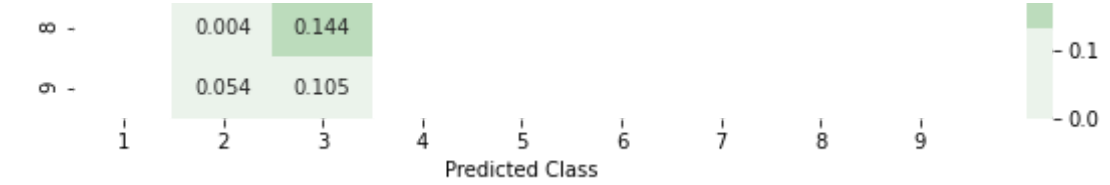
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



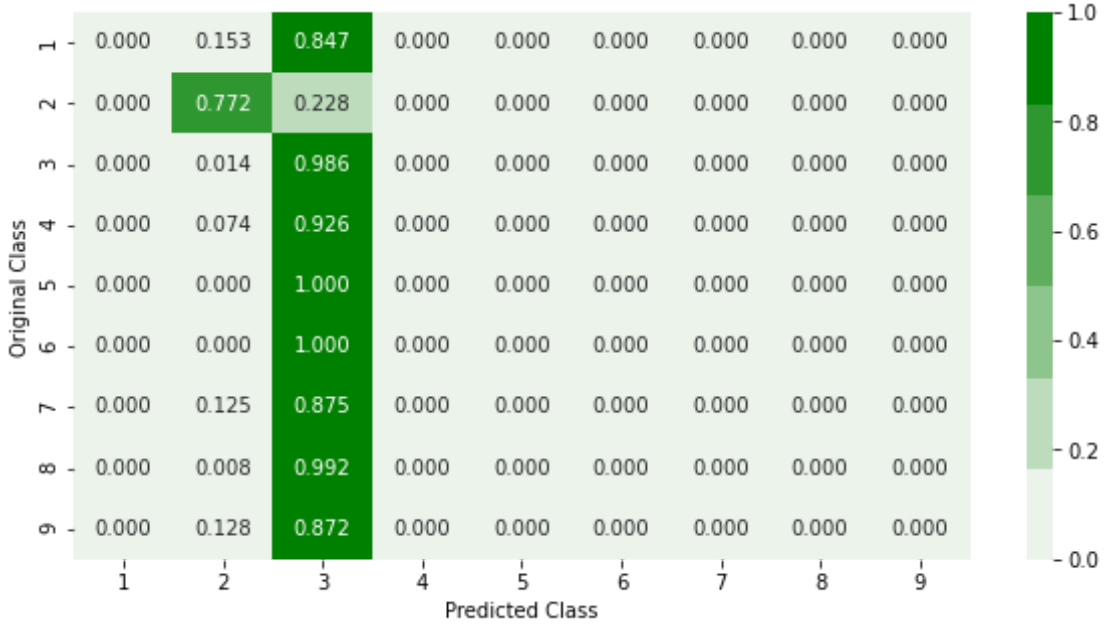
---------- Precision matrix -



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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




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

Random Forest

In [13]:
```
alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

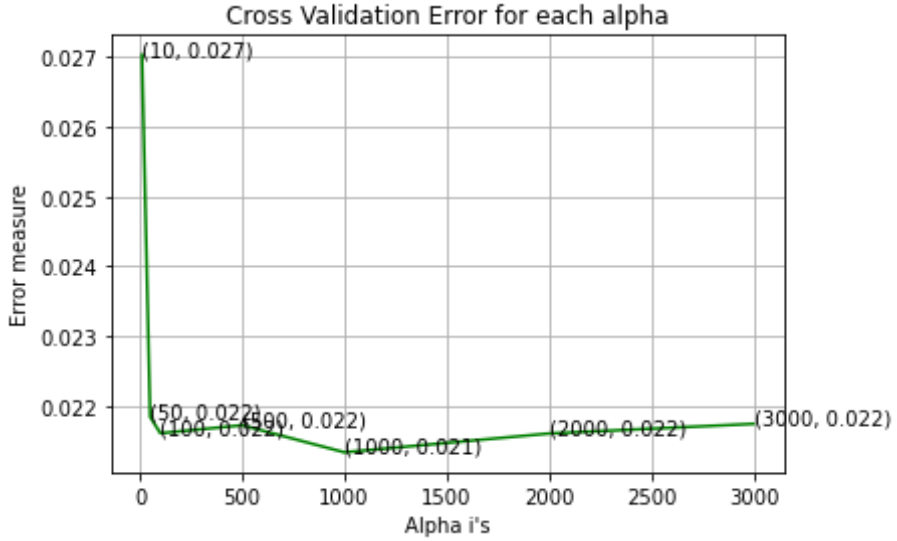
```
from sklearn.ensemble import RandomForestClassifier
for i in tqdm(alpha):
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=
-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.cl
asses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_stat
e=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

```
100%|██████████| 7/7 [03:28<00:00, 29.72s/it]
```

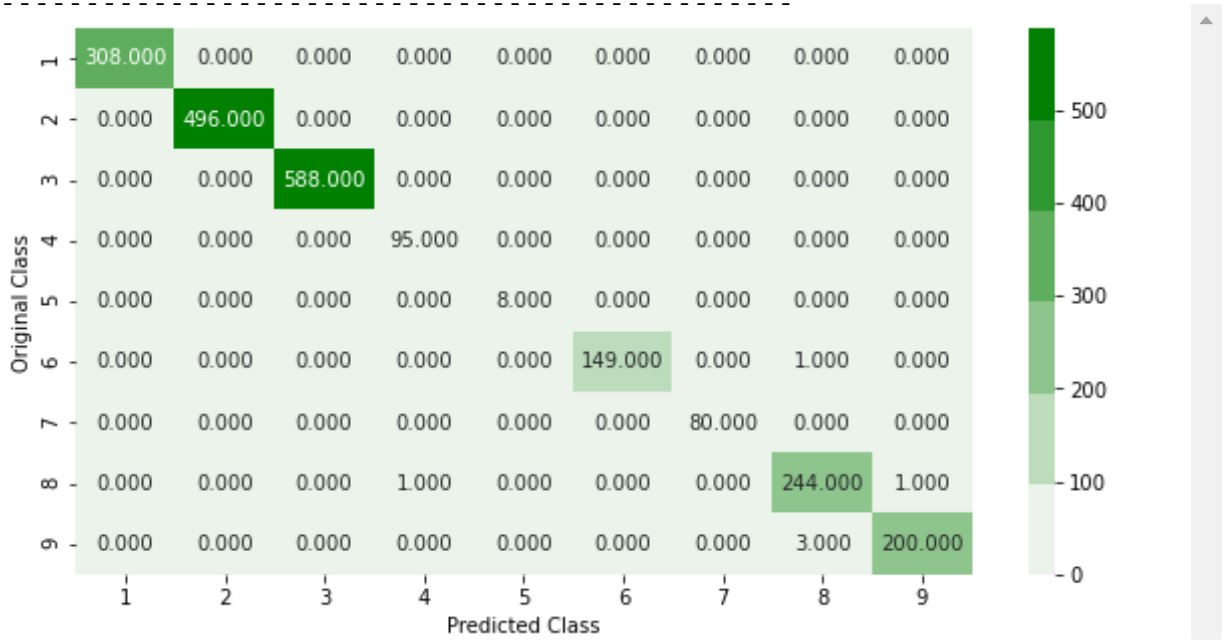
```
log_loss for c =  10 is 0.02702454563415847
log_loss for c =  50 is 0.021849751605969095
log_loss for c =  100 is 0.021616370806786817
log_loss for c =  500 is 0.021730534837724236
log_loss for c =  1000 is 0.021343634239766675
log_loss for c =  2000 is 0.021612408828795955
log_loss for c =  3000 is 0.021750221787880393
```




```
For values of best alpha =  1000 The train log loss is: 0.0123397024952
14225
For values of best alpha =  1000 The cross validation log loss is: 0.02
1343634239766675
For values of best alpha =  1000 The test log loss is: 0.02155481087749
```

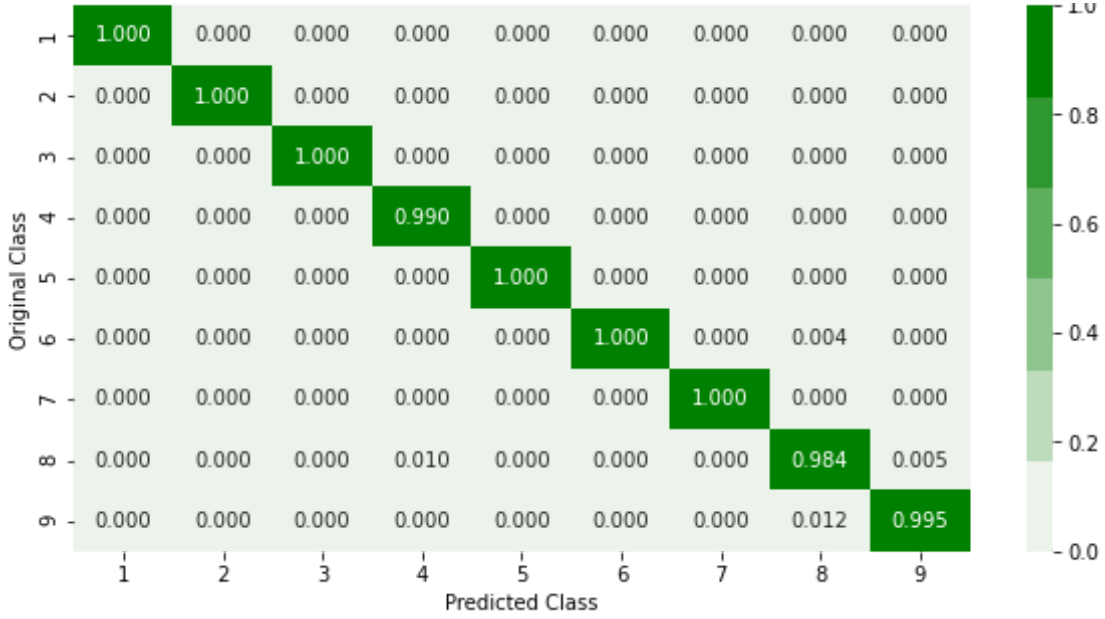
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
747
Number of misclassified points  0.27598896044158233
---------------------------------------------------- Confusion matrix ---
--------------------------------------------------
```

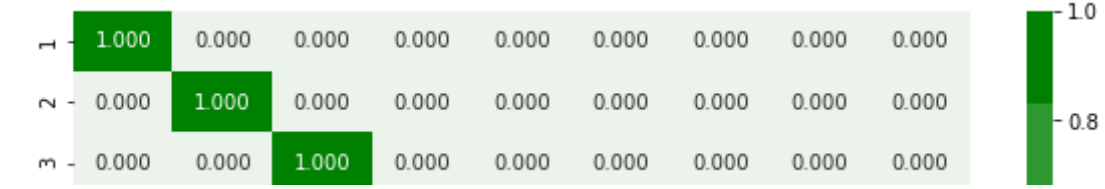



```
---------------------------------------------------- Precision matrix ---
--------------------------------------------------
```

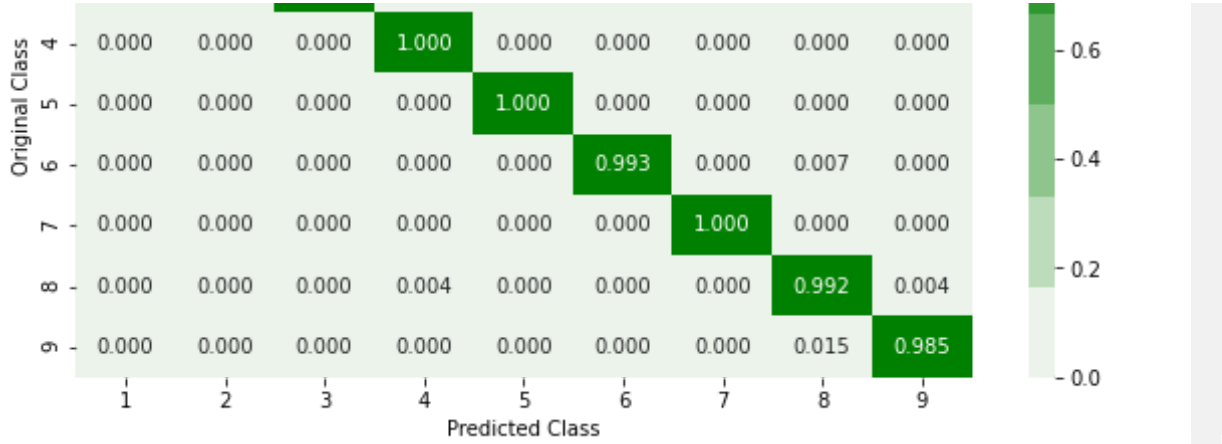
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




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

XGBoost Classification

In [14]:

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

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


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

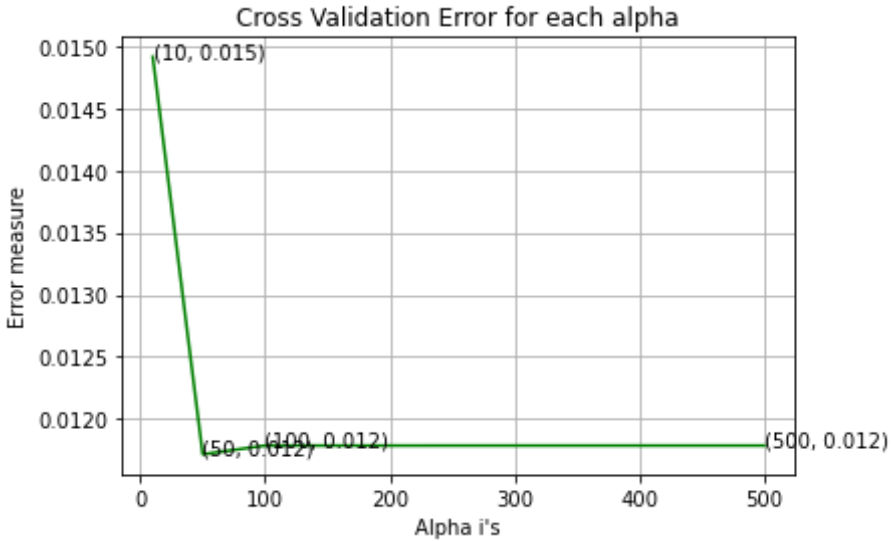
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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

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

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

```
100%|██████████| 4/4 [19:15<00:00, 288.97s/it]
```

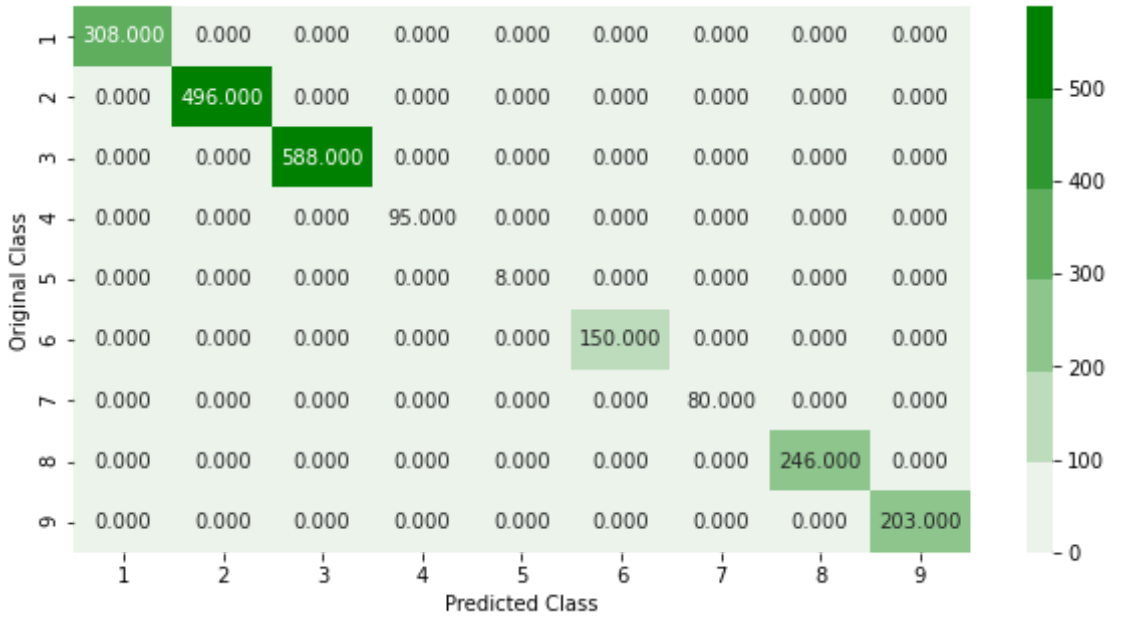
```
log_loss for c =  10 is 0.014913880085089022
log_loss for c =  50 is 0.011715972006006874
log_loss for c =  100 is 0.011786220121411655
log_loss for c =  500 is 0.011786434994453993
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD




```
For values of best alpha =  50 The train log loss is: 0.0074219739257
85645
For values of best alpha =  50 The cross validation log loss is: 0.01
1715972006006874
For values of best alpha =  50 The test log loss is: 0.00752337570560
4348
Number of misclassified points  0.0
---------------------------------------------------- Confusion matrix -
----------------------------------------------------
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD


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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD


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

- - - - - Bytes-Bigrams Features(Top 300) + Pixel Intensity + ASM-Unigram Features

Model	Test Loss	Miss-classification
Random Model	2.46008	88.13 %
KNN	0.2340	5.84 %
Logistic Regression	1.6743	55.70 %
Random Forest Classifier	0.0215	0.27 %
XGBoost Classsifier	0.0075	0.0 %

Summary

----------Only Bytes - Bigram Features(Top 300)----------

Model	Test Loss
Random Model	2.52195
KNN	1.89611
Logistic Regression	1.89969
Random Forest Classifier	0.04673
XGBoost Classsifier	0.04479

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

---------- Bytes-Unigrams + Bytes-Bigrams(Top 300) ----------

Model	Test Loss
Random Model	2.52195
KNN	0.32085
Logistic Regression	1.1275
Random Forest Classifier	0.04717
XGBoost Classsifier	0.048067

---------- Only Pixel Intensity Features ----------

Model	Test Loss
KNN	1.8894
Logistic Regression	1.8998
Random Forest Classifier	1.284
XGBoost Classsifier	1.0544

- - - - - Bytes-Bigrams Features(Top 300) + Pixel Intensity + ASM-Unigram Features

Model	Test Loss	Miss-classification
Random Model	2.46008	88.13 %

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Model	Test Loss	Miss-classification
KNN	0.2340	5.84 %
Logistic Regression	1.6743	55.70 %
Random Forest Classifier	0.0215	0.27 %
XGBoost Classsifier	0.0075	0.0 %

By using Bytes-Bigrams Features(Top 300) + Pixel Intensity + ASM-Unigram Features , got a log loss of 0.0075 (less than 0.01) with XGBoost Classifier

In [ ]:

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD