Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25 (https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25)
2. https://www.youtube.com/watch?v=UwbuW7oK8rk (https://www.youtube.com/watch?v=UwbuW7oK8rk)
3. https://www.youtube.com/watch?v=qxXRKVompI8 (https://www.youtube.com/watch?v=qxXRKVompI8)

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data (https://www.kaggle.com/c/msk-redefining-cancer-treatment/data)
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

ID,Gene,Variation,Class
0,FAM58A,Truncating Mutations,1
1,CBL,W802*,2
2,CBL,Q249E,2
...

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation (https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation)

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
#from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from sklearn.feature_selection import SelectKBest,chi2
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

```
Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

Out[2]:

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

training/training_variants is a comma separated file containing the description of the genetic mutations used for training.
Fields are

- **ID :** the id of the row used to link the mutation to the clinical evidence
- **Gene :** the gene where this genetic mutation is located
- **Variation :** the aminoacid change for this mutations
- **Class :** 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

In [3]:

```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names
=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

```
Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']
```

Out[3]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

In [4]:

```
data_text['TEXT'] = pd.Series(data_text['TEXT']).replace(np.nan,"word")
```

3.1.3. Preprocessing of text

In [5]:

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then retain that word from the data
            if not word in stop_words:
                string += word + " "

        data_text[column][index] = string
```

In [6]:

```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    nlp_preprocessing(row['TEXT'], index, 'TEXT')
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
Time took for preprocessing the text : 205.015603 seconds
```

In [7]:

```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[7]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

In [8]:

```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of output
 varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [9]:

```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

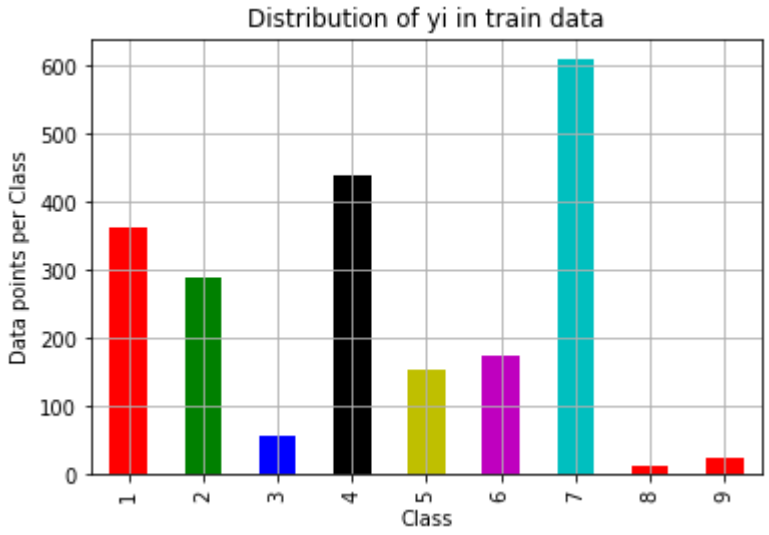
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

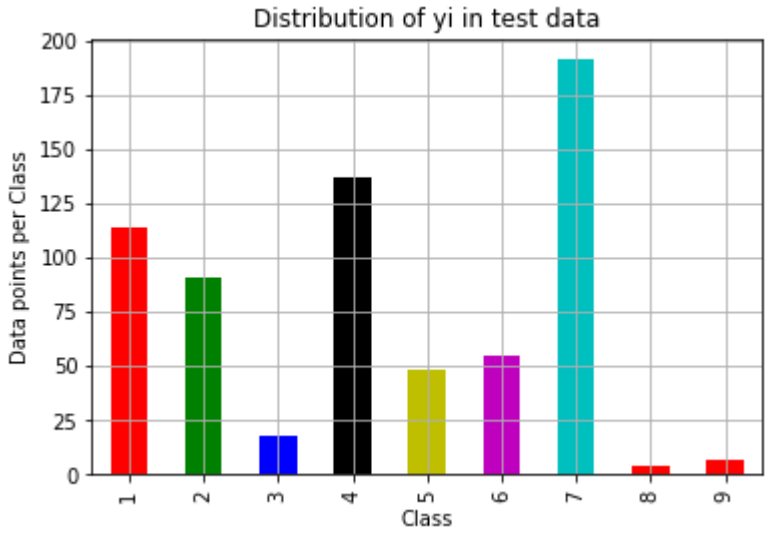
In [10]:

```
# it returns a dict, keys as class labels and values as the number of data point
s in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsor
t.html
# -(train_class_distribution.values): the minus sign will give us in decreasing
 order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.va
lues[i], '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100
), 3), '%)')


print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsor
t.html
# -(train_class_distribution.values): the minus sign will give us in decreasing
 order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.val
ues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100),
3), '%)')

print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsor
t.html
# -(train_class_distribution.values): the minus sign will give us in decreasing
 order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
```

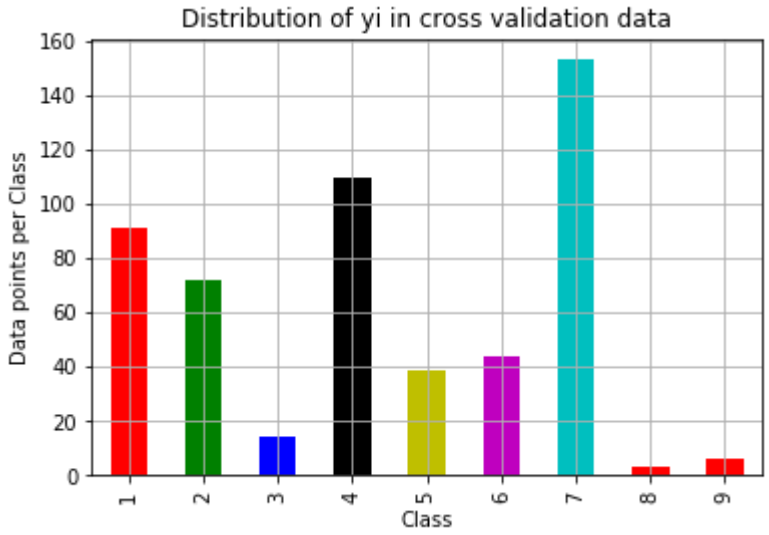
```
    print('Number of data points in class', i+1, ':',cv_class_distribution.value
s[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3),
'%)')
```




```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
--------------------------------------------------------------------------------
------------
```



```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
------------------------------------------------------------------------
------------
```




```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [11]:

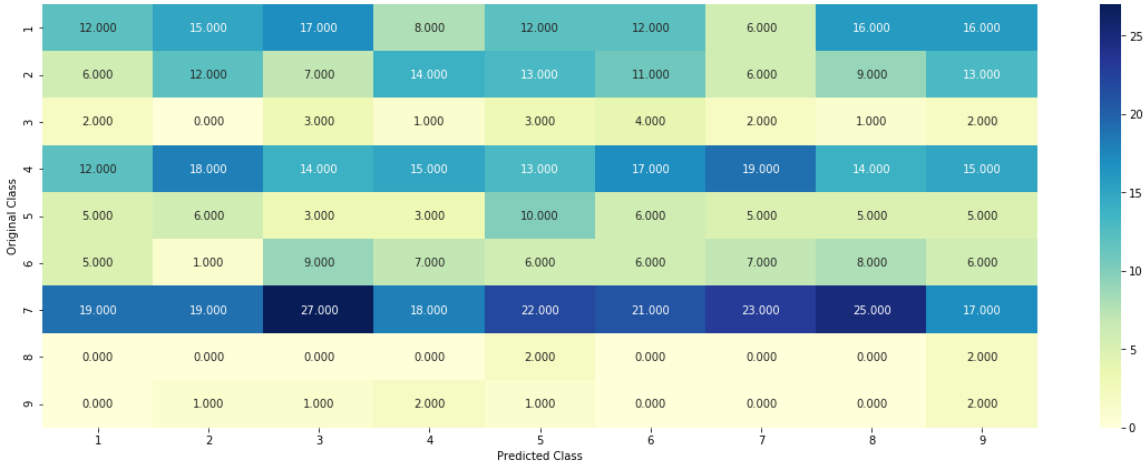
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are
 predicted class j

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

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
cklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
cklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yti
cklabels=labels)
    plt.xlabel('Predicted Class')
```

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

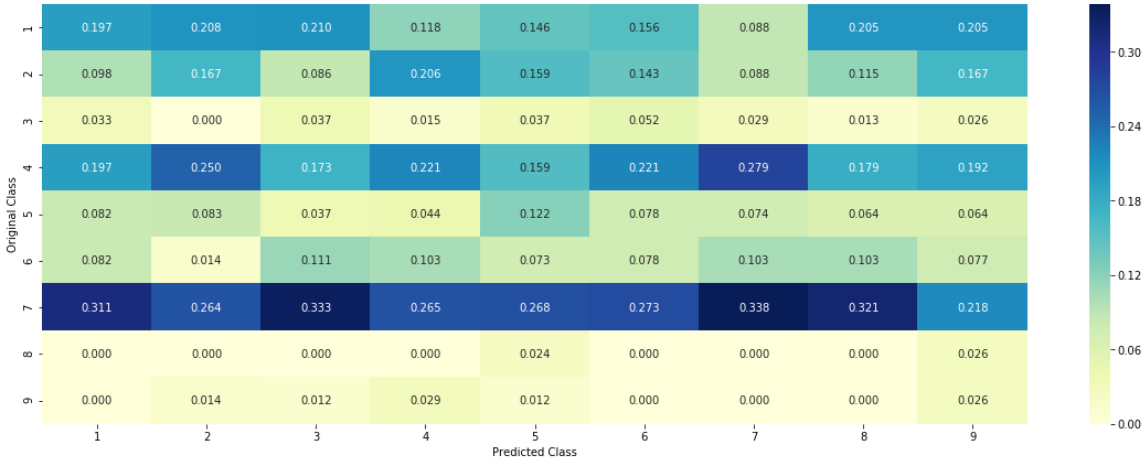
In [12]:

```
# 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 = test_df.shape[0]
cv_data_len = cv_df.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_pr
edicted_y, eps=1e-15))


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

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

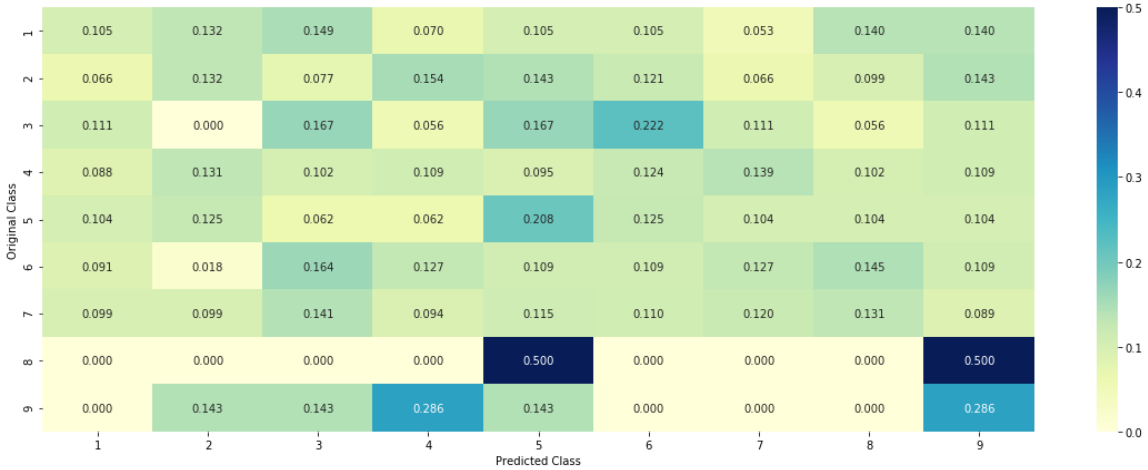
```
Log loss on Cross Validation Data using Random Model 2.5422063256559
52
Log loss on Test Data using Random Model 2.5494240133058828
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ----------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



3.3 Univariate Analysis

In [13]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in tr
ain data dataframe
# build a vector (1*9) , the first element = (number of times it occured in clas
s1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representatio
n of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for ea
ch gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occure
d in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to
```

```
 perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BR
CA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==
i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that p
articular feature occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818
177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.0378
7878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918
366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.05102040
8163265307, 0.05102040816326530 7, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818
18181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818
181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060606060606
0608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060
606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918238993710
6917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.0691823
89937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476821192052
95, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317
880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333
334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.2999999
9999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each featu
re value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is th
```

```
ere in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- (numerator + 10*alpha) / (denominator + 90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

In [14]:

```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 230
BRCA1     175
TP53      101
EGFR       88
BRCA2      80
PTEN       71
BRAF       58
KIT        53
ERBB2      45
ALK        44
PIK3CA     44
Name: Gene, dtype: int64
```

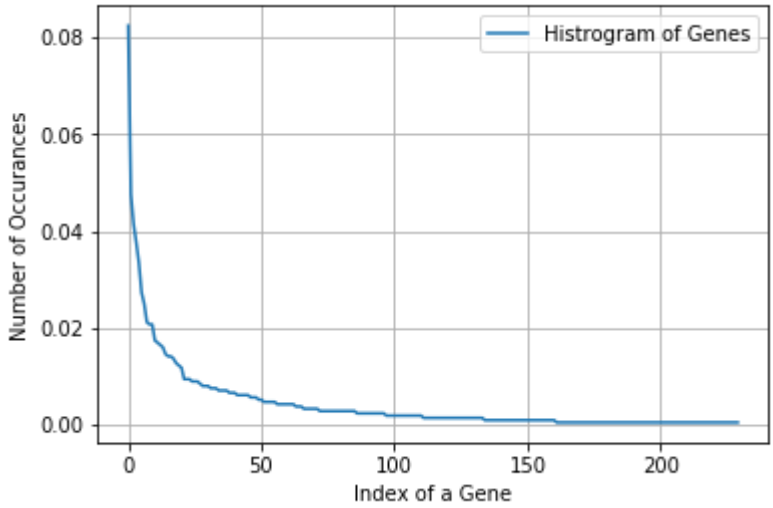
In [15]:

```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in
 the train data, and they are distibuted as follows",)
```

```
Ans: There are 230 different categories of genes in the train data,
and they are distibuted as follows
```

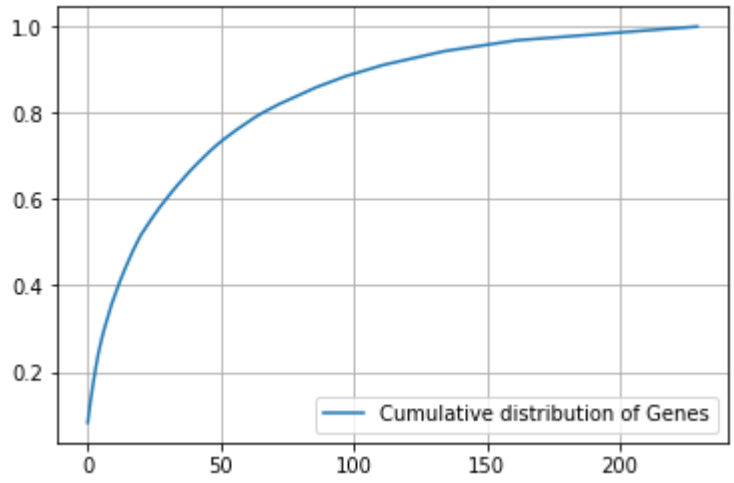
In [16]:

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [17]:

```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

In [18]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train
_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_d
f))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [19]:

```
print("train_gene_feature_responseCoding is converted feature using respone codi
ng method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone
coding method. The shape of gene feature: (2124, 9)
```

In [20]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'
])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [21]:

```
train_df['Gene'].head()
```

Out[21]:

```
2551    BRCA1
1793       AR
503      TP53
1278     HRAS
549     SMAD2
Name: Gene, dtype: object
```

In [22]:

```
gene_vectorizer.get_feature_names()
```

Out[22]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
```

```
    'dnmt3b',
    'dusp4',
    'egfr',
    'eif1ax',
    'elf3',
    'ep300',
    'epas1',
    'epcam',
    'erbb2',
    'erbb3',
    'erbb4',
    'ercc2',
    'ercc3',
    'ercc4',
    'erg',
    'esr1',
    'etv1',
    'etv6',
    'ewsr1',
    'ezh2',
    'fanca',
    'fancc',
    'fat1',
    'fbxw7',
    'fgf19',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'fgfr4',
    'flt1',
    'flt3',
    'foxa1',
    'foxp1',
    'gata3',
    'gli1',
    'gnaq',
    'gnas',
    'h3f3a',
    'hla',
    'hnf1a',
    'hras',
    'idh1',
    'idh2',
    'igf1r',
    'ikzf1',
    'inpp4b',
    'jak1',
    'jak2',
    'jun',
    'kdm5a',
    'kdm5c',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'klf4',
    'kmt2a',
    'kmt2b',
    'kmt2c',
    'kmt2d',
    'knstrn',
```

```
    'kras',
    'lats2',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mapk1',
    'mdm2',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mga',
    'mlh1',
    'mpl',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myd88',
    'myod1',
    'nf1',
    'nf2',
    'nfe2l2',
    'nkx2',
    'notch1',
    'notch2',
    'nras',
    'nsd1',
    'ntrk1',
    'ntrk2',
    'ntrk3',
    'nup93',
    'pax8',
    'pbrm1',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
    'pik3r3',
    'pim1',
    'pms1',
    'pms2',
    'pole',
    'ppp2r1a',
    'ppp6c',
    'prdm1',
    'ptch1',
    'pten',
    'ptpn11',
    'ptprd',
    'ptprt',
    'rab35',
    'rac1',
    'rad21',
    'rad50',
    'rad51c',
```

```
 'rad51d',
 'rad54l',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'rybp',
 'setd2',
 'sf3b1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xrcc2',
 'yap1']
```

In [23]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encodi
ng method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot e
ncoding method. The shape of gene feature: (2124, 229)
```

Q4. How good is this gene feature in predicting y_i?

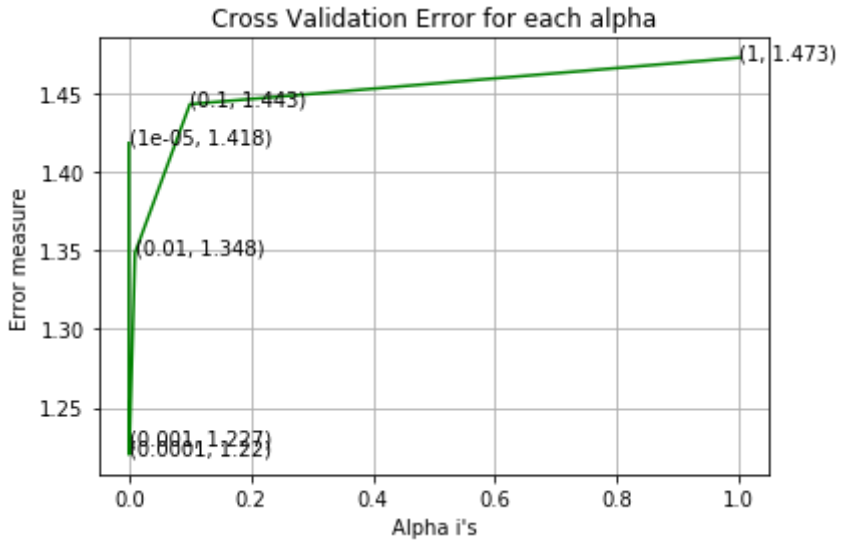
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [24]:

```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

#--------------------------------
# video link:
#--------------------------------


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict
_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.418211954845093
For values of alpha =  0.0001 The log loss is: 1.22013834731225
For values of alpha =  0.001 The log loss is: 1.226676245314495
For values of alpha =  0.01 The log loss is: 1.3483743404923492
For values of alpha =  0.1 The log loss is: 1.4429722009037245
For values of alpha =  1 The log loss is: 1.472533028572825
```




```
For values of best alpha =  0.0001 The train log loss is: 1.03416082
81864554
For values of best alpha =  0.0001 The cross validation log loss is:
1.22013834731225
For values of best alpha =  0.0001 The test log loss is: 1.224664543
5402173
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

In [25]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", un
ique_genes.shape[0], " genes in train dataset?")

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0
]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test
_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(
cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the
230  genes in train dataset?
Ans
1. In test data 640 out of 665 : 96.2406015037594
2. In cross validation data 515 out of  532 : 96.80451127819549
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [26]:

```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1938
Truncating_Mutations    54
Deletion                46
Amplification           45
Fusions                 19
Overexpression           4
T58I                     3
G12V                     3
Q61H                     3
Q61R                     3
C618R                    2
Name: Variation, dtype: int64
```

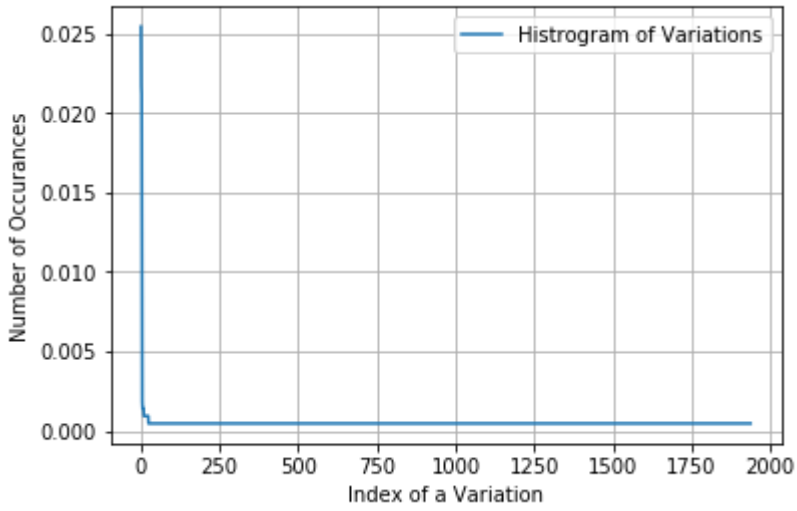
In [27]:

```
print("Ans: There are", unique_variations.shape[0] ,"different categories of var
iations in the train data, and they are distibuted as follows",)
```

```
Ans: There are 1938 different categories of variations in the train
data, and they are distibuted as follows
```

In [28]:

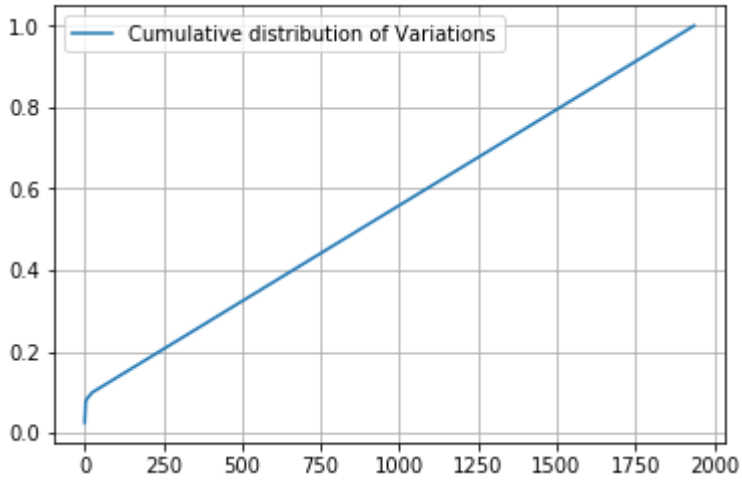
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [29]:

```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02542373 0.04708098 0.06826742 ... 0.99905838 0.99952919 1.
 ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

We will be using both these methods to featurize the Variation Feature

In [30]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variati
on", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
n", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation"
, cv_df))
```

In [31]:

```
print("train_variation_feature_responseCoding is a converted feature using the r
esponse coding method. The shape of Variation feature:", train_variation_feature
_responseCoding.shape)
```

```
train_variation_feature_responseCoding is a converted feature using
the response coding method. The shape of Variation feature: (2124,
9)
```

In [32]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_
df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Va
riation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variat
ion'])
```

In [33]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne
-hot encoding method. The shape of Variation feature:", train_variation_feature_
onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the
onne-hot encoding method. The shape of Variation feature: (2124, 196
5)
```

Q10. How good is this Variation feature in predicting y_i?

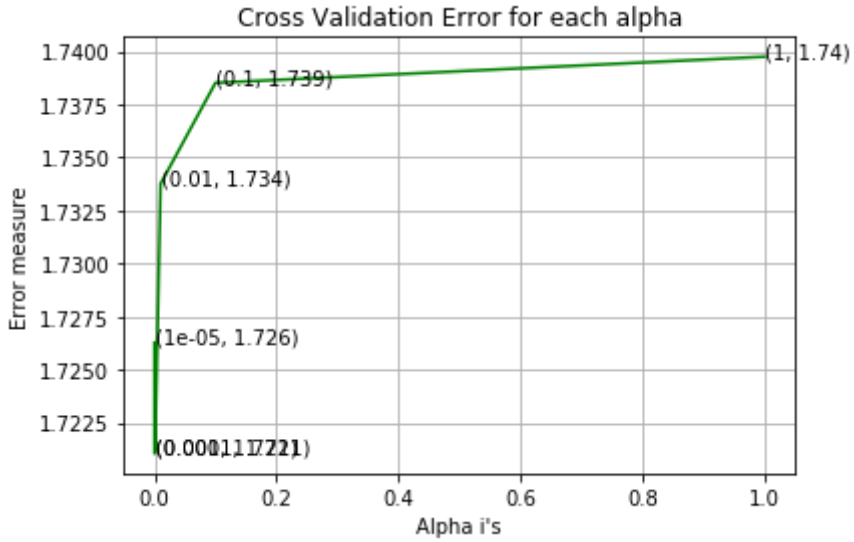
Let's build a model just like the earlier!

In [34]:

```
alpha = [10 ** x for x in range(-5, 1)]

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

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

#-------------------------------
# video link:
#-------------------------------


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict
_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
```

```
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7262719513718132
For values of alpha =  0.0001 The log loss is: 1.7210468323918993
For values of alpha =  0.001 The log loss is: 1.7211136286916773
For values of alpha =  0.01 The log loss is: 1.733756573533442
For values of alpha =  0.1 The log loss is: 1.7385176046784137
For values of alpha =  1 The log loss is: 1.7397491209413942
```




```
For values of best alpha =  0.0001 The train log loss is: 0.74094774
99779145
For values of best alpha =  0.0001 The cross validation log loss is:
1.7210468323918993
For values of best alpha =  0.0001 The test log loss is: 1.693444996
2890878
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Not sure! But lets be very sure using the below analysis.

In [35]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape
[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].sha
pe[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test
_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(
cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1938  genes in test
and cross validation data sets?
Ans
1. In test data 83 out of 665 : 12.481203007518797
2. In cross validation data 50 out of  532 : 9.398496240601503
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

In [36]:

```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
```

In [37]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict
.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(ro
w['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [38]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in
 train data
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'
])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns
 (1*number of features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of ti
mes it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


print("Total number of unique words in train data :", len(train_text_features))
```

```
Total number of unique words in train data : 53722
```

In [39]:

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

In [40]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [41]:

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_t
ext_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text
_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_featu
re_responseCoding.sum(axis=1)).T
```

In [42]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axi
s=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=
0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

In [43]:

```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , r
everse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [44]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 6167, 4: 3413, 5: 2983, 6: 2678, 9: 1872, 7: 1845, 8: 18
18, 12: 1622, 10: 1402, 15: 1061, 11: 993, 14: 925, 13: 922, 16: 88
3, 18: 740, 17: 582, 20: 569, 24: 550, 21: 513, 19: 480, 25: 479, 2
8: 420, 22: 419, 23: 389, 30: 369, 26: 362, 27: 332, 32: 331, 44: 32
5, 36: 311, 56: 287, 29: 273, 33: 257, 40: 252, 35: 249, 31: 248, 3
4: 223, 48: 208, 42: 198, 38: 198, 37: 195, 39: 191, 41: 186, 45: 18
3, 49: 170, 52: 164, 50: 164, 46: 163, 58: 156, 54: 148, 60: 147, 4
3: 146, 57: 135, 51: 131, 64: 128, 53: 124, 47: 118, 59: 117, 55: 11
6, 67: 109, 65: 105, 80: 104, 76: 104, 72: 103, 70: 103, 66: 103, 6
3: 103, 61: 103, 62: 98, 90: 96, 68: 91, 81: 89, 78: 88, 73: 86, 69:
86, 74: 80, 71: 77, 77: 73, 88: 71, 75: 70, 84: 69, 79: 69, 96: 68,
85: 68, 105: 66, 87: 62, 99: 61, 83: 61, 104: 59, 93: 58, 89: 58, 8
6: 58, 120: 57, 97: 57, 91: 57, 114: 56, 100: 56, 112: 55, 82: 55, 1
21: 52, 119: 52, 95: 52, 103: 51, 102: 51, 92: 51, 126: 49, 115: 49,
110: 49, 108: 49, 94: 48, 138: 46, 106: 46, 101: 46, 142: 45, 128: 4
5, 124: 44, 123: 44, 109: 44, 134: 43, 98: 43, 145: 41, 116: 41, 13
3: 39, 150: 38, 136: 38, 111: 38, 156: 37, 130: 37, 117: 37, 107: 3
7, 144: 36, 140: 36, 135: 36, 152: 35, 146: 35, 137: 35, 122: 35, 19
2: 34, 168: 34, 141: 34, 125: 34, 113: 34, 139: 33, 157: 32, 118: 3
2, 164: 31, 131: 31, 129: 31, 162: 29, 151: 29, 195: 28, 172: 28, 15
8: 28, 127: 28, 228: 27, 174: 27, 169: 27, 160: 27, 148: 27, 211: 2
6, 196: 26, 188: 26, 147: 26, 143: 26, 199: 25, 198: 25, 185: 25, 17
1: 25, 201: 24, 180: 24, 167: 24, 132: 24, 280: 23, 217: 23, 193: 2
3, 179: 23, 165: 23, 163: 23, 159: 23, 155: 23, 220: 22, 194: 22, 15
3: 22, 245: 21, 244: 21, 212: 21, 187: 21, 175: 21, 222: 20, 197: 2
0, 177: 20, 173: 20, 308: 19, 263: 19, 261: 19, 252: 19, 246: 19, 23
4: 19, 225: 19, 224: 19, 210: 19, 191: 19, 189: 19, 183: 19, 182: 1
9, 166: 19, 161: 19, 154: 19, 326: 18, 288: 18, 264: 18, 250: 18, 23
1: 18, 229: 18, 216: 18, 206: 18, 204: 18, 184: 18, 178: 18, 310: 1
7, 284: 17, 268: 17, 248: 17, 221: 17, 215: 17, 214: 17, 207: 17, 20
0: 17, 170: 17, 149: 17, 329: 16, 272: 16, 258: 16, 253: 16, 238: 1
6, 232: 16, 226: 16, 208: 16, 205: 16, 202: 16, 335: 15, 304: 15, 29
9: 15, 267: 15, 256: 15, 230: 15, 213: 15, 203: 15, 181: 15, 176: 1
5, 291: 14, 274: 14, 271: 14, 269: 14, 265: 14, 243: 14, 242: 14, 24
1: 14, 236: 14, 209: 14, 190: 14, 355: 13, 322: 13, 321: 13, 307: 1
3, 286: 13, 275: 13, 266: 13, 259: 13, 254: 13, 249: 13, 239: 13, 23
7: 13, 235: 13, 218: 13, 459: 12, 357: 12, 312: 12, 303: 12, 295: 1
2, 278: 12, 276: 12, 251: 12, 227: 12, 223: 12, 219: 12, 423: 11, 38
2: 11, 372: 11, 371: 11, 363: 11, 351: 11, 319: 11, 314: 11, 311: 1
1, 301: 11, 297: 11, 294: 11, 293: 11, 270: 11, 262: 11, 257: 11, 24
7: 11, 240: 11, 233: 11, 492: 10, 460: 10, 441: 10, 418: 10, 395: 1
0, 365: 10, 353: 10, 345: 10, 344: 10, 340: 10, 338: 10, 330: 10, 32
4: 10, 313: 10, 292: 10, 255: 10, 487: 9, 471: 9, 413: 9, 412: 9, 40
5: 9, 394: 9, 390: 9, 386: 9, 385: 9, 381: 9, 376: 9, 373: 9, 359:
9, 354: 9, 352: 9, 349: 9, 347: 9, 336: 9, 315: 9, 300: 9, 283: 9, 2
82: 9, 281: 9, 279: 9, 186: 9, 829: 8, 704: 8, 653: 8, 652: 8, 632:
8, 616: 8, 566: 8, 565: 8, 528: 8, 474: 8, 462: 8, 449: 8, 444: 8, 4
08: 8, 401: 8, 398: 8, 393: 8, 389: 8, 383: 8, 370: 8, 348: 8, 346:
8, 343: 8, 337: 8, 331: 8, 328: 8, 309: 8, 302: 8, 290: 8, 273: 8, 7
25: 7, 593: 7, 569: 7, 550: 7, 524: 7, 504: 7, 478: 7, 472: 7, 467:
7, 456: 7, 453: 7, 452: 7, 451: 7, 438: 7, 435: 7, 432: 7, 426: 7, 4
11: 7, 402: 7, 391: 7, 387: 7, 377: 7, 367: 7, 361: 7, 356: 7, 334:
7, 332: 7, 323: 7, 320: 7, 318: 7, 316: 7, 306: 7, 296: 7, 289: 7, 2
87: 7, 277: 7, 1147: 6, 958: 6, 850: 6, 809: 6, 786: 6, 754: 6, 747:
6, 733: 6, 731: 6, 722: 6, 716: 6, 713: 6, 677: 6, 667: 6, 603: 6, 5
78: 6, 576: 6, 567: 6, 553: 6, 551: 6, 546: 6, 527: 6, 510: 6, 508:
6, 506: 6, 495: 6, 479: 6, 476: 6, 466: 6, 465: 6, 463: 6, 455: 6, 4
54: 6, 448: 6, 447: 6, 437: 6, 431: 6, 430: 6, 429: 6, 425: 6, 422:
6, 419: 6, 396: 6, 392: 6, 380: 6, 378: 6, 362: 6, 360: 6, 358: 6, 3
42: 6, 341: 6, 339: 6, 333: 6, 305: 6, 285: 6, 1466: 5, 1289: 5, 127
6: 5, 942: 5, 921: 5, 875: 5, 768: 5, 743: 5, 738: 5, 734: 5, 680:
```

```
5, 674: 5, 661: 5, 645: 5, 644: 5, 639: 5, 635: 5, 633: 5, 627: 5, 6
23: 5, 622: 5, 614: 5, 605: 5, 601: 5, 598: 5, 592: 5, 588: 5, 579:
5, 570: 5, 568: 5, 564: 5, 560: 5, 559: 5, 558: 5, 547: 5, 541: 5, 5
20: 5, 516: 5, 507: 5, 505: 5, 502: 5, 496: 5, 491: 5, 489: 5, 488:
5, 485: 5, 482: 5, 468: 5, 442: 5, 436: 5, 433: 5, 427: 5, 424: 5, 4
21: 5, 420: 5, 417: 5, 416: 5, 414: 5, 406: 5, 404: 5, 403: 5, 400:
5, 399: 5, 388: 5, 379: 5, 375: 5, 374: 5, 368: 5, 350: 5, 327: 5, 3
25: 5, 317: 5, 298: 5, 260: 5, 4323: 4, 2446: 4, 1822: 4, 1547: 4, 1
499: 4, 1448: 4, 1292: 4, 1232: 4, 1216: 4, 1170: 4, 1089: 4, 1023:
4, 1014: 4, 1003: 4, 1000: 4, 976: 4, 973: 4, 972: 4, 938: 4, 929:
4, 925: 4, 917: 4, 909: 4, 889: 4, 879: 4, 877: 4, 867: 4, 856: 4, 8
55: 4, 847: 4, 846: 4, 838: 4, 835: 4, 833: 4, 817: 4, 789: 4, 787:
4, 784: 4, 779: 4, 767: 4, 763: 4, 758: 4, 757: 4, 755: 4, 753: 4, 7
52: 4, 749: 4, 744: 4, 724: 4, 720: 4, 715: 4, 711: 4, 709: 4, 706:
4, 701: 4, 700: 4, 692: 4, 691: 4, 684: 4, 673: 4, 669: 4, 663: 4, 6
60: 4, 655: 4, 651: 4, 650: 4, 647: 4, 636: 4, 618: 4, 612: 4, 608:
4, 602: 4, 600: 4, 597: 4, 596: 4, 595: 4, 594: 4, 586: 4, 584: 4, 5
83: 4, 577: 4, 574: 4, 572: 4, 563: 4, 545: 4, 542: 4, 540: 4, 539:
4, 538: 4, 534: 4, 533: 4, 519: 4, 515: 4, 513: 4, 503: 4, 499: 4, 4
97: 4, 494: 4, 493: 4, 486: 4, 483: 4, 473: 4, 469: 4, 464: 4, 461:
4, 457: 4, 445: 4, 440: 4, 439: 4, 428: 4, 409: 4, 407: 4, 384: 4, 3
69: 4, 366: 4, 4261: 3, 2174: 3, 2149: 3, 2018: 3, 1991: 3, 1948: 3,
1919: 3, 1909: 3, 1895: 3, 1815: 3, 1783: 3, 1777: 3, 1769: 3, 1763:
3, 1736: 3, 1735: 3, 1688: 3, 1682: 3, 1593: 3, 1590: 3, 1581: 3, 14
96: 3, 1454: 3, 1452: 3, 1444: 3, 1433: 3, 1430: 3, 1390: 3, 1371:
3, 1360: 3, 1329: 3, 1315: 3, 1294: 3, 1272: 3, 1270: 3, 1268: 3, 12
61: 3, 1259: 3, 1248: 3, 1245: 3, 1236: 3, 1234: 3, 1233: 3, 1229:
3, 1226: 3, 1217: 3, 1214: 3, 1211: 3, 1210: 3, 1206: 3, 1201: 3, 11
96: 3, 1194: 3, 1181: 3, 1177: 3, 1176: 3, 1174: 3, 1164: 3, 1162:
3, 1155: 3, 1113: 3, 1109: 3, 1105: 3, 1103: 3, 1098: 3, 1087: 3, 10
83: 3, 1079: 3, 1077: 3, 1073: 3, 1069: 3, 1061: 3, 1043: 3, 1034:
3, 1031: 3, 1029: 3, 1001: 3, 989: 3, 980: 3, 974: 3, 971: 3, 969:
3, 967: 3, 956: 3, 955: 3, 952: 3, 949: 3, 947: 3, 945: 3, 937: 3, 9
32: 3, 930: 3, 923: 3, 920: 3, 914: 3, 910: 3, 900: 3, 894: 3, 893:
3, 892: 3, 888: 3, 884: 3, 878: 3, 876: 3, 874: 3, 872: 3, 854: 3, 8
41: 3, 839: 3, 837: 3, 832: 3, 825: 3, 818: 3, 816: 3, 814: 3, 811:
3, 804: 3, 802: 3, 785: 3, 783: 3, 778: 3, 774: 3, 769: 3, 766: 3, 7
60: 3, 750: 3, 748: 3, 745: 3, 742: 3, 737: 3, 736: 3, 735: 3, 728:
3, 717: 3, 710: 3, 708: 3, 707: 3, 687: 3, 675: 3, 672: 3, 671: 3, 6
68: 3, 666: 3, 665: 3, 664: 3, 662: 3, 659: 3, 658: 3, 657: 3, 654:
3, 641: 3, 637: 3, 619: 3, 617: 3, 615: 3, 613: 3, 610: 3, 609: 3, 6
04: 3, 599: 3, 590: 3, 589: 3, 587: 3, 585: 3, 581: 3, 580: 3, 573:
3, 561: 3, 556: 3, 537: 3, 535: 3, 531: 3, 526: 3, 525: 3, 517: 3, 5
14: 3, 511: 3, 509: 3, 500: 3, 490: 3, 484: 3, 475: 3, 470: 3, 446:
3, 443: 3, 434: 3, 415: 3, 410: 3, 364: 3, 8950: 2, 7114: 2, 7073:
2, 7013: 2, 6993: 2, 6690: 2, 6648: 2, 6556: 2, 5691: 2, 5577: 2, 49
32: 2, 4842: 2, 4840: 2, 4764: 2, 4755: 2, 4699: 2, 4675: 2, 4552:
2, 4120: 2, 4098: 2, 3948: 2, 3942: 2, 3870: 2, 3733: 2, 3679: 2, 36
03: 2, 3592: 2, 3467: 2, 3456: 2, 3436: 2, 3420: 2, 3383: 2, 3351:
2, 3339: 2, 3304: 2, 3276: 2, 3269: 2, 3227: 2, 3129: 2, 3123: 2, 30
65: 2, 3057: 2, 3040: 2, 3004: 2, 2935: 2, 2931: 2, 2799: 2, 2792:
2, 2783: 2, 2774: 2, 2741: 2, 2722: 2, 2693: 2, 2659: 2, 2655: 2, 26
28: 2, 2624: 2, 2595: 2, 2588: 2, 2561: 2, 2559: 2, 2532: 2, 2517:
2, 2490: 2, 2482: 2, 2456: 2, 2433: 2, 2421: 2, 2416: 2, 2384: 2, 23
56: 2, 2315: 2, 2310: 2, 2299: 2, 2298: 2, 2282: 2, 2280: 2, 2273:
2, 2251: 2, 2240: 2, 2233: 2, 2218: 2, 2187: 2, 2160: 2, 2140: 2, 21
36: 2, 2126: 2, 2123: 2, 2093: 2, 2092: 2, 2084: 2, 2078: 2, 2076:
2, 2074: 2, 2059: 2, 2054: 2, 2038: 2, 2016: 2, 2015: 2, 2014: 2, 20
06: 2, 1989: 2, 1968: 2, 1966: 2, 1936: 2, 1935: 2, 1934: 2, 1930:
2, 1928: 2, 1913: 2, 1908: 2, 1898: 2, 1894: 2, 1885: 2, 1874: 2, 18
70: 2, 1862: 2, 1849: 2, 1843: 2, 1841: 2, 1834: 2, 1810: 2, 1804:
```

```
2, 1802: 2, 1797: 2, 1794: 2, 1788: 2, 1785: 2, 1778: 2, 1772: 2, 17
65: 2, 1759: 2, 1741: 2, 1724: 2, 1723: 2, 1721: 2, 1718: 2, 1716:
2, 1714: 2, 1707: 2, 1678: 2, 1654: 2, 1646: 2, 1639: 2, 1638: 2, 16
24: 2, 1615: 2, 1591: 2, 1589: 2, 1586: 2, 1585: 2, 1582: 2, 1578:
2, 1566: 2, 1565: 2, 1563: 2, 1561: 2, 1555: 2, 1554: 2, 1552: 2, 15
50: 2, 1549: 2, 1546: 2, 1540: 2, 1531: 2, 1530: 2, 1525: 2, 1524:
2, 1522: 2, 1502: 2, 1487: 2, 1481: 2, 1479: 2, 1471: 2, 1470: 2, 14
69: 2, 1445: 2, 1440: 2, 1422: 2, 1415: 2, 1412: 2, 1406: 2, 1403:
2, 1400: 2, 1395: 2, 1393: 2, 1385: 2, 1382: 2, 1379: 2, 1378: 2, 13
77: 2, 1372: 2, 1362: 2, 1359: 2, 1358: 2, 1356: 2, 1355: 2, 1354:
2, 1345: 2, 1332: 2, 1331: 2, 1330: 2, 1327: 2, 1320: 2, 1317: 2, 13
01: 2, 1298: 2, 1295: 2, 1283: 2, 1264: 2, 1263: 2, 1255: 2, 1250:
2, 1246: 2, 1244: 2, 1237: 2, 1235: 2, 1223: 2, 1221: 2, 1215: 2, 12
12: 2, 1208: 2, 1207: 2, 1199: 2, 1195: 2, 1189: 2, 1187: 2, 1184:
2, 1175: 2, 1173: 2, 1168: 2, 1165: 2, 1161: 2, 1159: 2, 1157: 2, 11
48: 2, 1146: 2, 1142: 2, 1139: 2, 1134: 2, 1128: 2, 1124: 2, 1123:
2, 1121: 2, 1117: 2, 1116: 2, 1108: 2, 1102: 2, 1100: 2, 1093: 2, 10
82: 2, 1080: 2, 1076: 2, 1070: 2, 1067: 2, 1066: 2, 1065: 2, 1064:
2, 1060: 2, 1054: 2, 1049: 2, 1045: 2, 1040: 2, 1038: 2, 1024: 2, 10
22: 2, 1018: 2, 1015: 2, 1011: 2, 1010: 2, 1009: 2, 1007: 2, 1006:
2, 994: 2, 993: 2, 988: 2, 986: 2, 984: 2, 983: 2, 966: 2, 963: 2, 9
60: 2, 951: 2, 936: 2, 935: 2, 933: 2, 927: 2, 926: 2, 918: 2, 915:
2, 912: 2, 911: 2, 906: 2, 901: 2, 895: 2, 891: 2, 890: 2, 887: 2, 8
82: 2, 881: 2, 871: 2, 870: 2, 869: 2, 862: 2, 860: 2, 858: 2, 852:
2, 849: 2, 844: 2, 840: 2, 836: 2, 834: 2, 826: 2, 822: 2, 820: 2, 8
19: 2, 815: 2, 813: 2, 808: 2, 806: 2, 805: 2, 801: 2, 800: 2, 799:
2, 796: 2, 792: 2, 788: 2, 780: 2, 777: 2, 776: 2, 775: 2, 772: 2, 7
61: 2, 759: 2, 751: 2, 739: 2, 729: 2, 726: 2, 723: 2, 719: 2, 705:
2, 697: 2, 694: 2, 693: 2, 686: 2, 683: 2, 681: 2, 678: 2, 648: 2, 6
43: 2, 640: 2, 638: 2, 634: 2, 629: 2, 626: 2, 625: 2, 621: 2, 611:
2, 607: 2, 591: 2, 557: 2, 555: 2, 554: 2, 552: 2, 549: 2, 548: 2, 5
44: 2, 536: 2, 532: 2, 530: 2, 523: 2, 521: 2, 512: 2, 481: 2, 458:
2, 450: 2, 397: 2, 151425: 1, 118240: 1, 80700: 1, 66966: 1, 66790:
1, 66438: 1, 66264: 1, 63267: 1, 62050: 1, 54921: 1, 53365: 1, 4883
8: 1, 48773: 1, 46890: 1, 46867: 1, 44377: 1, 42761: 1, 42079: 1, 41
820: 1, 41812: 1, 41693: 1, 40993: 1, 40478: 1, 38738: 1, 38524: 1,
37610: 1, 37043: 1, 35929: 1, 34692: 1, 33353: 1, 33141: 1, 33062:
1, 32953: 1, 32902: 1, 32509: 1, 31570: 1, 31475: 1, 29131: 1, 2793
1: 1, 26591: 1, 25935: 1, 25801: 1, 24993: 1, 24694: 1, 24557: 1, 24
240: 1, 24201: 1, 24136: 1, 23889: 1, 23754: 1, 23495: 1, 23136: 1,
21983: 1, 21902: 1, 21863: 1, 21590: 1, 21536: 1, 21482: 1, 21224:
1, 20951: 1, 20511: 1, 20131: 1, 19794: 1, 19769: 1, 19712: 1, 1942
5: 1, 19231: 1, 18994: 1, 18864: 1, 18690: 1, 18602: 1, 18471: 1, 18
423: 1, 18359: 1, 18316: 1, 18160: 1, 18105: 1, 17986: 1, 17958: 1,
17886: 1, 17872: 1, 17756: 1, 17531: 1, 17465: 1, 17431: 1, 17270:
1, 17236: 1, 17192: 1, 17026: 1, 17025: 1, 16987: 1, 16962: 1, 1680
8: 1, 16650: 1, 16597: 1, 16431: 1, 16298: 1, 16264: 1, 15844: 1, 15
841: 1, 15715: 1, 15691: 1, 15620: 1, 15465: 1, 15254: 1, 15253: 1,
15161: 1, 15157: 1, 15130: 1, 15019: 1, 14865: 1, 14760: 1, 14753:
1, 14735: 1, 14679: 1, 14531: 1, 14263: 1, 14241: 1, 14223: 1, 1404
3: 1, 13971: 1, 13964: 1, 13868: 1, 13861: 1, 13655: 1, 13652: 1, 13
628: 1, 13444: 1, 13426: 1, 13314: 1, 13284: 1, 13268: 1, 13161: 1,
13006: 1, 12941: 1, 12892: 1, 12845: 1, 12805: 1, 12801: 1, 12697:
1, 12666: 1, 12637: 1, 12580: 1, 12558: 1, 12541: 1, 12453: 1, 1235
1: 1, 12345: 1, 12283: 1, 12260: 1, 12216: 1, 12202: 1, 12191: 1, 12
142: 1, 12134: 1, 12128: 1, 12126: 1, 12112: 1, 12101: 1, 12082: 1,
11985: 1, 11923: 1, 11876: 1, 11789: 1, 11759: 1, 11754: 1, 11748:
1, 11742: 1, 11737: 1, 11720: 1, 11650: 1, 11622: 1, 11620: 1, 1145
9: 1, 11417: 1, 11404: 1, 11289: 1, 11181: 1, 11159: 1, 11115: 1, 11
063: 1, 11012: 1, 10895: 1, 10873: 1, 10800: 1, 10793: 1, 10756: 1,
10734: 1, 10385: 1, 10267: 1, 10265: 1, 10237: 1, 10186: 1, 10162:
```

```
1, 10160: 1, 10148: 1, 10124: 1, 10051: 1, 10029: 1, 10012: 1, 1000
3: 1, 9942: 1, 9871: 1, 9829: 1, 9822: 1, 9812: 1, 9808: 1, 9790: 1,
9763: 1, 9759: 1, 9744: 1, 9741: 1, 9732: 1, 9723: 1, 9572: 1, 9480:
1, 9440: 1, 9365: 1, 9342: 1, 9322: 1, 9243: 1, 9238: 1, 9223: 1, 91
86: 1, 9173: 1, 9162: 1, 9139: 1, 9074: 1, 9062: 1, 9051: 1, 9046:
1, 9041: 1, 9010: 1, 8987: 1, 8925: 1, 8902: 1, 8887: 1, 8869: 1, 88
68: 1, 8844: 1, 8799: 1, 8776: 1, 8712: 1, 8665: 1, 8644: 1, 8589:
1, 8569: 1, 8480: 1, 8458: 1, 8437: 1, 8410: 1, 8377: 1, 8374: 1, 83
42: 1, 8292: 1, 8281: 1, 8266: 1, 8188: 1, 8182: 1, 8148: 1, 8143:
1, 8123: 1, 8090: 1, 8081: 1, 8079: 1, 8060: 1, 8046: 1, 8012: 1, 80
00: 1, 7979: 1, 7950: 1, 7907: 1, 7903: 1, 7874: 1, 7870: 1, 7865:
1, 7862: 1, 7859: 1, 7803: 1, 7802: 1, 7798: 1, 7766: 1, 7700: 1, 76
96: 1, 7692: 1, 7640: 1, 7625: 1, 7622: 1, 7615: 1, 7600: 1, 7570:
1, 7530: 1, 7504: 1, 7496: 1, 7490: 1, 7471: 1, 7446: 1, 7441: 1, 74
01: 1, 7391: 1, 7357: 1, 7350: 1, 7333: 1, 7311: 1, 7300: 1, 7295:
1, 7291: 1, 7250: 1, 7249: 1, 7244: 1, 7242: 1, 7212: 1, 7183: 1, 71
62: 1, 7142: 1, 7109: 1, 7080: 1, 7064: 1, 7061: 1, 7049: 1, 7045:
1, 7041: 1, 7017: 1, 7007: 1, 7006: 1, 6990: 1, 6946: 1, 6943: 1, 69
25: 1, 6898: 1, 6886: 1, 6867: 1, 6838: 1, 6802: 1, 6800: 1, 6772:
1, 6735: 1, 6726: 1, 6706: 1, 6686: 1, 6631: 1, 6617: 1, 6602: 1, 65
88: 1, 6568: 1, 6565: 1, 6561: 1, 6534: 1, 6523: 1, 6516: 1, 6478:
1, 6423: 1, 6401: 1, 6391: 1, 6377: 1, 6357: 1, 6338: 1, 6331: 1, 62
94: 1, 6285: 1, 6261: 1, 6259: 1, 6247: 1, 6235: 1, 6213: 1, 6212:
1, 6167: 1, 6160: 1, 6157: 1, 6147: 1, 6145: 1, 6134: 1, 6105: 1, 61
03: 1, 6060: 1, 6056: 1, 6035: 1, 6033: 1, 5996: 1, 5982: 1, 5979:
1, 5978: 1, 5960: 1, 5959: 1, 5956: 1, 5949: 1, 5948: 1, 5912: 1, 59
11: 1, 5896: 1, 5886: 1, 5884: 1, 5872: 1, 5871: 1, 5864: 1, 5802:
1, 5800: 1, 5786: 1, 5720: 1, 5718: 1, 5715: 1, 5677: 1, 5675: 1, 56
71: 1, 5668: 1, 5663: 1, 5653: 1, 5652: 1, 5638: 1, 5636: 1, 5618:
1, 5601: 1, 5570: 1, 5569: 1, 5545: 1, 5534: 1, 5531: 1, 5515: 1, 54
96: 1, 5462: 1, 5457: 1, 5446: 1, 5440: 1, 5422: 1, 5415: 1, 5403:
1, 5398: 1, 5388: 1, 5376: 1, 5374: 1, 5368: 1, 5365: 1, 5363: 1, 53
46: 1, 5345: 1, 5310: 1, 5295: 1, 5239: 1, 5220: 1, 5206: 1, 5188:
1, 5176: 1, 5164: 1, 5161: 1, 5158: 1, 5149: 1, 5143: 1, 5122: 1, 51
17: 1, 5115: 1, 5111: 1, 5087: 1, 5080: 1, 5074: 1, 5072: 1, 5070:
1, 5050: 1, 5033: 1, 5028: 1, 5012: 1, 4980: 1, 4975: 1, 4969: 1, 49
54: 1, 4931: 1, 4909: 1, 4908: 1, 4905: 1, 4897: 1, 4889: 1, 4887:
1, 4881: 1, 4866: 1, 4862: 1, 4855: 1, 4850: 1, 4848: 1, 4838: 1, 48
28: 1, 4817: 1, 4795: 1, 4788: 1, 4786: 1, 4777: 1, 4772: 1, 4759:
1, 4751: 1, 4748: 1, 4744: 1, 4743: 1, 4732: 1, 4706: 1, 4700: 1, 46
81: 1, 4679: 1, 4678: 1, 4651: 1, 4636: 1, 4630: 1, 4626: 1, 4620:
1, 4596: 1, 4572: 1, 4568: 1, 4550: 1, 4526: 1, 4515: 1, 4501: 1, 44
99: 1, 4492: 1, 4484: 1, 4469: 1, 4464: 1, 4462: 1, 4456: 1, 4443:
1, 4432: 1, 4425: 1, 4418: 1, 4417: 1, 4415: 1, 4405: 1, 4393: 1, 43
92: 1, 4389: 1, 4387: 1, 4376: 1, 4367: 1, 4362: 1, 4355: 1, 4326:
1, 4324: 1, 4317: 1, 4304: 1, 4294: 1, 4293: 1, 4291: 1, 4290: 1, 42
85: 1, 4281: 1, 4263: 1, 4258: 1, 4254: 1, 4242: 1, 4236: 1, 4232:
1, 4222: 1, 4206: 1, 4203: 1, 4201: 1, 4193: 1, 4186: 1, 4170: 1, 41
61: 1, 4158: 1, 4154: 1, 4151: 1, 4140: 1, 4138: 1, 4133: 1, 4130:
1, 4126: 1, 4117: 1, 4116: 1, 4106: 1, 4105: 1, 4104: 1, 4096: 1, 40
84: 1, 4080: 1, 4078: 1, 4077: 1, 4074: 1, 4069: 1, 4068: 1, 4066:
1, 4064: 1, 4052: 1, 4032: 1, 4005: 1, 3999: 1, 3996: 1, 3994: 1, 39
85: 1, 3978: 1, 3976: 1, 3971: 1, 3965: 1, 3962: 1, 3960: 1, 3959:
1, 3954: 1, 3946: 1, 3922: 1, 3921: 1, 3920: 1, 3908: 1, 3902: 1, 38
92: 1, 3889: 1, 3886: 1, 3885: 1, 3879: 1, 3876: 1, 3863: 1, 3862:
1, 3856: 1, 3854: 1, 3816: 1, 3812: 1, 3809: 1, 3807: 1, 3805: 1, 38
04: 1, 3803: 1, 3802: 1, 3778: 1, 3771: 1, 3769: 1, 3758: 1, 3757:
1, 3754: 1, 3753: 1, 3752: 1, 3741: 1, 3725: 1, 3720: 1, 3715: 1, 37
12: 1, 3711: 1, 3700: 1, 3694: 1, 3692: 1, 3688: 1, 3687: 1, 3680:
1, 3678: 1, 3676: 1, 3658: 1, 3656: 1, 3653: 1, 3645: 1, 3640: 1, 36
27: 1, 3624: 1, 3609: 1, 3608: 1, 3605: 1, 3598: 1, 3597: 1, 3591:
```

```
1, 3588: 1, 3582: 1, 3581: 1, 3574: 1, 3571: 1, 3563: 1, 3562: 1, 35
55: 1, 3551: 1, 3547: 1, 3545: 1, 3538: 1, 3537: 1, 3536: 1, 3527:
1, 3526: 1, 3525: 1, 3521: 1, 3520: 1, 3507: 1, 3504: 1, 3499: 1, 34
95: 1, 3494: 1, 3490: 1, 3483: 1, 3478: 1, 3475: 1, 3471: 1, 3465:
1, 3461: 1, 3460: 1, 3459: 1, 3450: 1, 3442: 1, 3429: 1, 3424: 1, 34
22: 1, 3418: 1, 3416: 1, 3411: 1, 3406: 1, 3401: 1, 3399: 1, 3390:
1, 3387: 1, 3375: 1, 3373: 1, 3359: 1, 3354: 1, 3352: 1, 3350: 1, 33
48: 1, 3347: 1, 3346: 1, 3341: 1, 3337: 1, 3336: 1, 3332: 1, 3329:
1, 3328: 1, 3321: 1, 3317: 1, 3312: 1, 3297: 1, 3290: 1, 3288: 1, 32
84: 1, 3281: 1, 3279: 1, 3278: 1, 3260: 1, 3259: 1, 3258: 1, 3252:
1, 3246: 1, 3237: 1, 3235: 1, 3233: 1, 3231: 1, 3222: 1, 3220: 1, 32
15: 1, 3209: 1, 3202: 1, 3184: 1, 3183: 1, 3182: 1, 3181: 1, 3174:
1, 3172: 1, 3162: 1, 3158: 1, 3157: 1, 3156: 1, 3149: 1, 3142: 1, 31
40: 1, 3139: 1, 3138: 1, 3136: 1, 3130: 1, 3128: 1, 3120: 1, 3117:
1, 3114: 1, 3112: 1, 3107: 1, 3105: 1, 3102: 1, 3098: 1, 3089: 1, 30
88: 1, 3087: 1, 3082: 1, 3081: 1, 3080: 1, 3079: 1, 3078: 1, 3075:
1, 3068: 1, 3060: 1, 3052: 1, 3045: 1, 3042: 1, 3039: 1, 3030: 1, 30
27: 1, 3025: 1, 3023: 1, 3016: 1, 3014: 1, 3009: 1, 3006: 1, 2999:
1, 2992: 1, 2990: 1, 2988: 1, 2986: 1, 2982: 1, 2981: 1, 2972: 1, 29
68: 1, 2962: 1, 2957: 1, 2953: 1, 2949: 1, 2948: 1, 2947: 1, 2943:
1, 2941: 1, 2938: 1, 2924: 1, 2920: 1, 2912: 1, 2907: 1, 2902: 1, 28
98: 1, 2893: 1, 2884: 1, 2877: 1, 2875: 1, 2873: 1, 2861: 1, 2856:
1, 2848: 1, 2841: 1, 2840: 1, 2834: 1, 2833: 1, 2832: 1, 2825: 1, 28
08: 1, 2801: 1, 2794: 1, 2793: 1, 2780: 1, 2776: 1, 2772: 1, 2770:
1, 2768: 1, 2764: 1, 2745: 1, 2740: 1, 2737: 1, 2730: 1, 2720: 1, 27
19: 1, 2713: 1, 2712: 1, 2711: 1, 2710: 1, 2700: 1, 2694: 1, 2690:
1, 2686: 1, 2682: 1, 2673: 1, 2668: 1, 2666: 1, 2658: 1, 2652: 1, 26
46: 1, 2644: 1, 2643: 1, 2639: 1, 2638: 1, 2636: 1, 2635: 1, 2631:
1, 2626: 1, 2625: 1, 2623: 1, 2620: 1, 2612: 1, 2611: 1, 2609: 1, 26
05: 1, 2601: 1, 2599: 1, 2597: 1, 2589: 1, 2586: 1, 2580: 1, 2576:
1, 2572: 1, 2568: 1, 2564: 1, 2560: 1, 2553: 1, 2552: 1, 2546: 1, 25
44: 1, 2540: 1, 2536: 1, 2531: 1, 2530: 1, 2525: 1, 2515: 1, 2512:
1, 2508: 1, 2502: 1, 2500: 1, 2499: 1, 2495: 1, 2489: 1, 2476: 1, 24
72: 1, 2471: 1, 2465: 1, 2464: 1, 2463: 1, 2461: 1, 2458: 1, 2457:
1, 2449: 1, 2448: 1, 2447: 1, 2445: 1, 2441: 1, 2440: 1, 2437: 1, 24
36: 1, 2432: 1, 2428: 1, 2426: 1, 2424: 1, 2423: 1, 2422: 1, 2411:
1, 2408: 1, 2404: 1, 2402: 1, 2401: 1, 2395: 1, 2383: 1, 2381: 1, 23
79: 1, 2376: 1, 2375: 1, 2373: 1, 2371: 1, 2370: 1, 2369: 1, 2363:
1, 2360: 1, 2352: 1, 2348: 1, 2345: 1, 2343: 1, 2339: 1, 2338: 1, 23
32: 1, 2330: 1, 2329: 1, 2328: 1, 2325: 1, 2320: 1, 2316: 1, 2313:
1, 2311: 1, 2309: 1, 2301: 1, 2295: 1, 2293: 1, 2292: 1, 2289: 1, 22
88: 1, 2283: 1, 2281: 1, 2278: 1, 2277: 1, 2270: 1, 2269: 1, 2268:
1, 2266: 1, 2265: 1, 2262: 1, 2258: 1, 2253: 1, 2249: 1, 2245: 1, 22
43: 1, 2241: 1, 2236: 1, 2231: 1, 2228: 1, 2225: 1, 2220: 1, 2216:
1, 2213: 1, 2209: 1, 2208: 1, 2206: 1, 2203: 1, 2199: 1, 2198: 1, 21
95: 1, 2186: 1, 2184: 1, 2180: 1, 2175: 1, 2171: 1, 2169: 1, 2168:
1, 2164: 1, 2163: 1, 2162: 1, 2161: 1, 2159: 1, 2157: 1, 2152: 1, 21
50: 1, 2146: 1, 2145: 1, 2143: 1, 2129: 1, 2114: 1, 2112: 1, 2110:
1, 2105: 1, 2103: 1, 2100: 1, 2099: 1, 2098: 1, 2095: 1, 2087: 1, 20
83: 1, 2077: 1, 2073: 1, 2072: 1, 2070: 1, 2065: 1, 2060: 1, 2055:
1, 2053: 1, 2052: 1, 2050: 1, 2045: 1, 2044: 1, 2043: 1, 2042: 1, 20
39: 1, 2035: 1, 2034: 1, 2033: 1, 2028: 1, 2024: 1, 2021: 1, 2020:
1, 2013: 1, 2012: 1, 2010: 1, 2004: 1, 2003: 1, 2001: 1, 1996: 1, 19
92: 1, 1990: 1, 1986: 1, 1984: 1, 1981: 1, 1976: 1, 1975: 1, 1974:
1, 1972: 1, 1964: 1, 1963: 1, 1962: 1, 1961: 1, 1956: 1, 1951: 1, 19
49: 1, 1946: 1, 1945: 1, 1940: 1, 1939: 1, 1925: 1, 1924: 1, 1921:
1, 1920: 1, 1918: 1, 1917: 1, 1912: 1, 1911: 1, 1910: 1, 1907: 1, 19
05: 1, 1903: 1, 1902: 1, 1900: 1, 1897: 1, 1896: 1, 1891: 1, 1890:
1, 1887: 1, 1884: 1, 1880: 1, 1875: 1, 1869: 1, 1867: 1, 1864: 1, 18
61: 1, 1860: 1, 1858: 1, 1857: 1, 1854: 1, 1853: 1, 1851: 1, 1850:
1, 1847: 1, 1846: 1, 1842: 1, 1838: 1, 1829: 1, 1826: 1, 1825: 1, 18
```

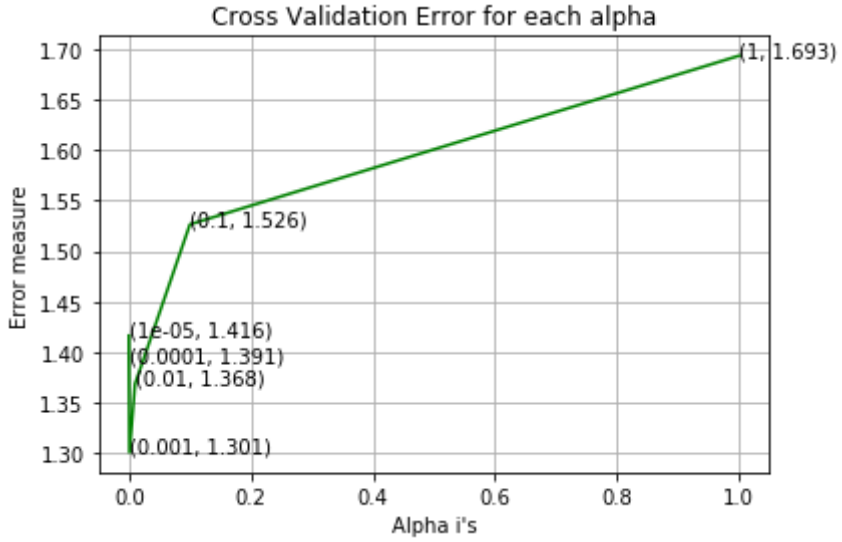
```
20: 1, 1819: 1, 1816: 1, 1811: 1, 1809: 1, 1807: 1, 1806: 1, 1805:
1, 1803: 1, 1800: 1, 1798: 1, 1795: 1, 1792: 1, 1787: 1, 1784: 1, 17
82: 1, 1775: 1, 1774: 1, 1771: 1, 1753: 1, 1751: 1, 1748: 1, 1746:
1, 1745: 1, 1744: 1, 1734: 1, 1732: 1, 1728: 1, 1720: 1, 1719: 1, 17
17: 1, 1713: 1, 1708: 1, 1706: 1, 1705: 1, 1704: 1, 1703: 1, 1702:
1, 1701: 1, 1698: 1, 1697: 1, 1696: 1, 1695: 1, 1693: 1, 1692: 1, 16
91: 1, 1684: 1, 1683: 1, 1680: 1, 1679: 1, 1677: 1, 1675: 1, 1674:
1, 1673: 1, 1672: 1, 1670: 1, 1666: 1, 1665: 1, 1664: 1, 1663: 1, 16
61: 1, 1660: 1, 1658: 1, 1657: 1, 1653: 1, 1652: 1, 1651: 1, 1643:
1, 1637: 1, 1636: 1, 1634: 1, 1633: 1, 1632: 1, 1629: 1, 1626: 1, 16
21: 1, 1620: 1, 1618: 1, 1617: 1, 1614: 1, 1613: 1, 1612: 1, 1610:
1, 1605: 1, 1603: 1, 1600: 1, 1597: 1, 1596: 1, 1594: 1, 1588: 1, 15
87: 1, 1584: 1, 1579: 1, 1576: 1, 1573: 1, 1572: 1, 1570: 1, 1567:
1, 1564: 1, 1562: 1, 1557: 1, 1556: 1, 1553: 1, 1548: 1, 1539: 1, 15
33: 1, 1529: 1, 1528: 1, 1526: 1, 1520: 1, 1519: 1, 1518: 1, 1516:
1, 1514: 1, 1513: 1, 1511: 1, 1509: 1, 1508: 1, 1507: 1, 1503: 1, 14
95: 1, 1494: 1, 1493: 1, 1492: 1, 1491: 1, 1485: 1, 1482: 1, 1480:
1, 1478: 1, 1477: 1, 1475: 1, 1473: 1, 1467: 1, 1463: 1, 1460: 1, 14
59: 1, 1457: 1, 1456: 1, 1455: 1, 1450: 1, 1449: 1, 1446: 1, 1441:
1, 1439: 1, 1437: 1, 1434: 1, 1432: 1, 1431: 1, 1429: 1, 1427: 1, 14
25: 1, 1423: 1, 1416: 1, 1414: 1, 1411: 1, 1409: 1, 1407: 1, 1405:
1, 1404: 1, 1401: 1, 1399: 1, 1397: 1, 1396: 1, 1392: 1, 1391: 1, 13
88: 1, 1384: 1, 1383: 1, 1381: 1, 1376: 1, 1375: 1, 1374: 1, 1373:
1, 1365: 1, 1363: 1, 1357: 1, 1353: 1, 1352: 1, 1349: 1, 1348: 1, 13
46: 1, 1343: 1, 1341: 1, 1339: 1, 1338: 1, 1336: 1, 1335: 1, 1334:
1, 1333: 1, 1326: 1, 1323: 1, 1322: 1, 1321: 1, 1319: 1, 1318: 1, 13
16: 1, 1314: 1, 1313: 1, 1312: 1, 1311: 1, 1310: 1, 1309: 1, 1307:
1, 1306: 1, 1304: 1, 1303: 1, 1299: 1, 1296: 1, 1293: 1, 1291: 1, 12
90: 1, 1288: 1, 1287: 1, 1286: 1, 1285: 1, 1281: 1, 1277: 1, 1274:
1, 1269: 1, 1267: 1, 1266: 1, 1265: 1, 1262: 1, 1260: 1, 1258: 1, 12
57: 1, 1256: 1, 1253: 1, 1252: 1, 1251: 1, 1249: 1, 1243: 1, 1242:
1, 1239: 1, 1230: 1, 1228: 1, 1224: 1, 1222: 1, 1220: 1, 1218: 1, 12
13: 1, 1209: 1, 1205: 1, 1203: 1, 1202: 1, 1198: 1, 1197: 1, 1193:
1, 1191: 1, 1186: 1, 1183: 1, 1182: 1, 1180: 1, 1169: 1, 1167: 1, 11
66: 1, 1163: 1, 1158: 1, 1156: 1, 1152: 1, 1145: 1, 1143: 1, 1140:
1, 1138: 1, 1136: 1, 1133: 1, 1130: 1, 1127: 1, 1126: 1, 1125: 1, 11
22: 1, 1118: 1, 1112: 1, 1101: 1, 1099: 1, 1097: 1, 1095: 1, 1094:
1, 1088: 1, 1084: 1, 1078: 1, 1071: 1, 1068: 1, 1063: 1, 1062: 1, 10
59: 1, 1058: 1, 1057: 1, 1056: 1, 1052: 1, 1051: 1, 1050: 1, 1048:
1, 1047: 1, 1042: 1, 1037: 1, 1036: 1, 1033: 1, 1028: 1, 1026: 1, 10
25: 1, 1020: 1, 1019: 1, 1017: 1, 1016: 1, 1013: 1, 1012: 1, 1005:
1, 1004: 1, 1002: 1, 998: 1, 997: 1, 996: 1, 995: 1, 992: 1, 991: 1,
987: 1, 981: 1, 979: 1, 978: 1, 977: 1, 970: 1, 968: 1, 965: 1, 961:
1, 959: 1, 953: 1, 950: 1, 946: 1, 944: 1, 941: 1, 931: 1, 928: 1, 9
24: 1, 919: 1, 916: 1, 913: 1, 908: 1, 905: 1, 904: 1, 902: 1, 897:
1, 896: 1, 886: 1, 883: 1, 880: 1, 868: 1, 866: 1, 865: 1, 864: 1, 8
63: 1, 857: 1, 853: 1, 851: 1, 848: 1, 843: 1, 831: 1, 828: 1, 824:
1, 821: 1, 812: 1, 810: 1, 807: 1, 803: 1, 797: 1, 794: 1, 793: 1, 7
91: 1, 790: 1, 782: 1, 781: 1, 773: 1, 764: 1, 746: 1, 741: 1, 732:
1, 727: 1, 721: 1, 718: 1, 702: 1, 699: 1, 698: 1, 690: 1, 685: 1, 6
82: 1, 676: 1, 649: 1, 646: 1, 642: 1, 631: 1, 630: 1, 628: 1, 624:
1, 620: 1, 575: 1, 543: 1, 529: 1, 522: 1, 501: 1, 498: 1, 480: 1, 4
77: 1})
```

In [45]:

```
# Train a Logistic regression+Calibration model using text features whicha re on
-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

#--------------------------------
# video link:
#--------------------------------


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict
_y, labels=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.41626964629891
For values of alpha =  0.0001 The log loss is: 1.3909706691900312
For values of alpha =  0.001 The log loss is: 1.300721357013104
For values of alpha =  0.01 The log loss is: 1.368384612225062
For values of alpha =  0.1 The log loss is: 1.526247745431413
For values of alpha =  1 The log loss is: 1.6930593908312692
```




```
For values of best alpha =  0.001 The train log loss is: 0.749092576
3549584
For values of best alpha =  0.001 The cross validation log loss is:
1.300721357013104
For values of best alpha =  0.001 The test log loss is: 1.2210982814
492621
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it seems like!

In [46]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [47]:

```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train da
ta")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in t
rain data")
```

```
96.635 % of word of test data appeared in train data
97.829 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [48]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belon
gs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y
))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [49]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [54]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".for
mat(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}
]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".for
mat(word,yes_no))

    print("Out of the top ",no_features," features ", word_present, "are present
 in query point")
```

Stacking the three types of features

In [55]:

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_vari
ation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
on_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea
ture_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_o
nehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_oneh
otCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCodi
ng)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,tra
in_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_
variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variat
ion_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_fe
ature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_featu
re_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_res
ponseCoding))
```

In [56]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_o
nehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_one
hotCoding.shape)
print("(number of data points * number of features) in cross validation data =",
 cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124,
55269)
(number of data points * number of features) in test data =  (665, 5
5269)
(number of data points * number of features) in cross validation dat
a = (532, 55269)
```

In [57]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_r
esponseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_res
ponseCoding.shape)
print("(number of data points * number of features) in cross validation data =",
 cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124,
27)
(number of data points * number of features) in test data =  (665, 2
7)
(number of data points * number of features) in cross validation dat
a = (532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

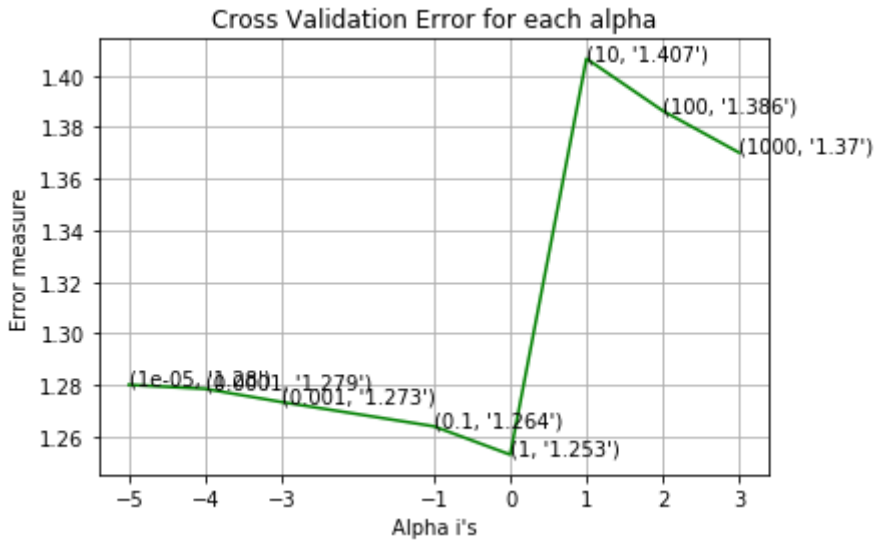
4.1.1.1. Hyper parameter tuning

In [54]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.2800910526546303
for alpha = 0.0001
Log Loss : 1.2785312369525215
for alpha = 0.001
Log Loss : 1.2734293823153178
for alpha = 0.1
Log Loss : 1.2640032684465607
for alpha = 1
Log Loss : 1.2530157080570592
for alpha = 10
Log Loss : 1.406538614416403
for alpha = 100
Log Loss : 1.3863453108174801
for alpha = 1000
Log Loss : 1.3701200767263473
```



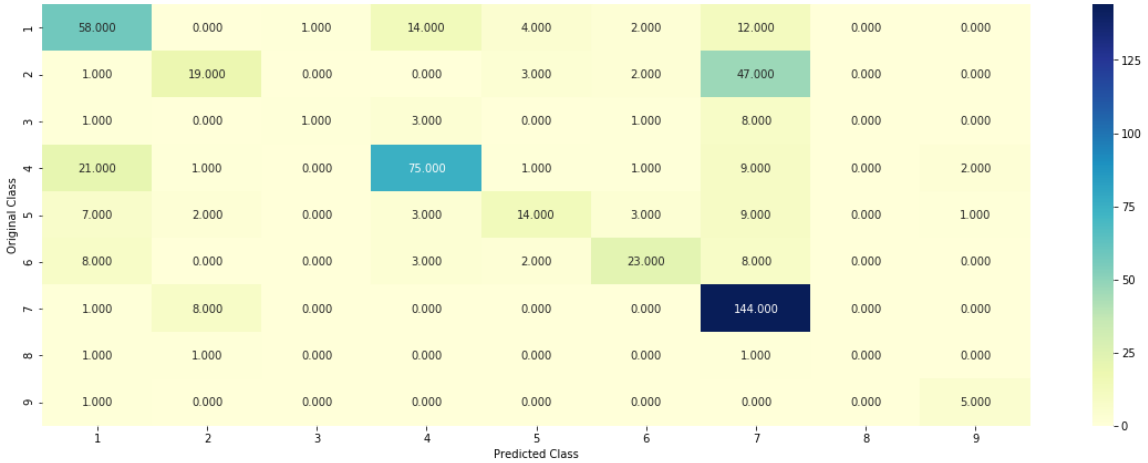

```
For values of best alpha =  1 The train log loss is: 0.9316748192289
975
For values of best alpha =  1 The cross validation log loss is: 1.25
30157080570592
For values of best alpha =  1 The test log loss is: 1.30552422350160
62
```

4.1.1.2. Testing the model with best hyper paramters

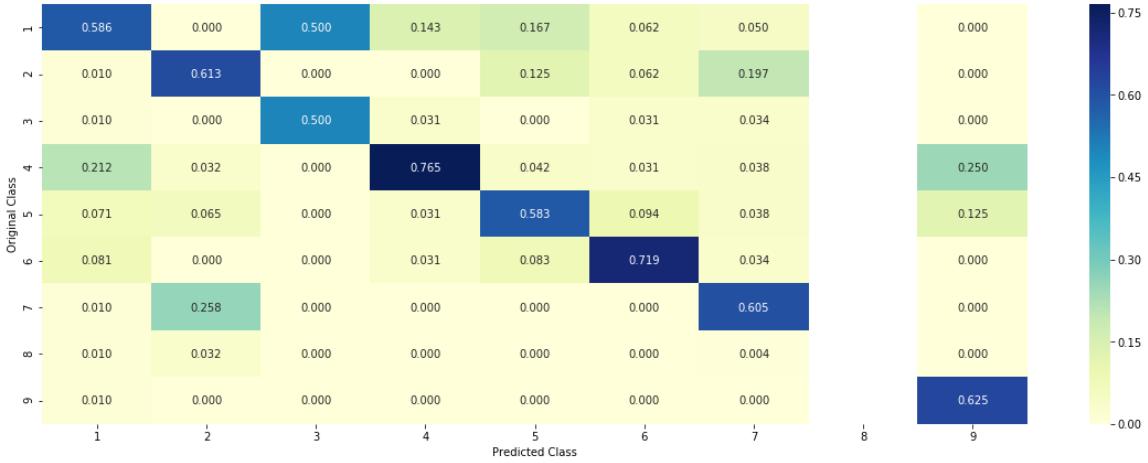
In [55]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


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

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability
 estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

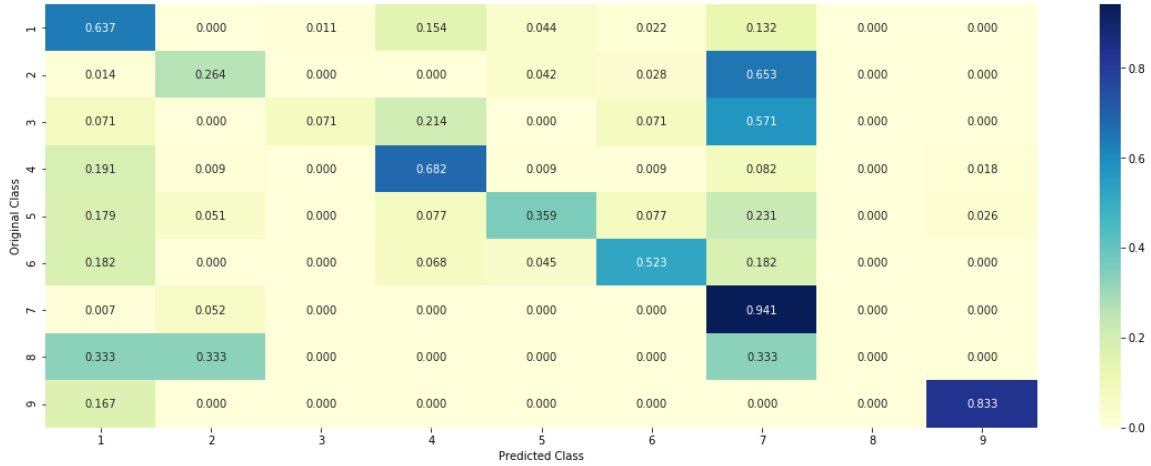
```
Log Loss : 1.2530157080570592
Number of missclassified point : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ----------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.1.1.3. Feature Importance, Correctly classified point

In [57]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6113 0.0982 0.0223 0.0995 0.0522
0.0408 0.0643 0.0063 0.0051]]
Actual Class : 1
--------------------------------------------------
11 Text feature [function] present in test data point [True]
12 Text feature [dna] present in test data point [True]
13 Text feature [type] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [affect] present in test data point [True]
16 Text feature [wild] present in test data point [True]
17 Text feature [one] present in test data point [True]
18 Text feature [two] present in test data point [True]
19 Text feature [binding] present in test data point [True]
22 Text feature [remaining] present in test data point [True]
23 Text feature [sequence] present in test data point [True]
24 Text feature [four] present in test data point [True]
25 Text feature [possible] present in test data point [True]
26 Text feature [amino] present in test data point [True]
27 Text feature [therefore] present in test data point [True]
28 Text feature [containing] present in test data point [True]
29 Text feature [effect] present in test data point [True]
30 Text feature [functions] present in test data point [True]
31 Text feature [region] present in test data point [True]
32 Text feature [large] present in test data point [True]
34 Text feature [specific] present in test data point [True]
35 Text feature [acids] present in test data point [True]
36 Text feature [corresponding] present in test data point [True]
37 Text feature [involved] present in test data point [True]
38 Text feature [present] present in test data point [True]
40 Text feature [results] present in test data point [True]
41 Text feature [three] present in test data point [True]
42 Text feature [structure] present in test data point [True]
43 Text feature [used] present in test data point [True]
45 Text feature [analysis] present in test data point [True]
46 Text feature [indicating] present in test data point [True]
47 Text feature [indicate] present in test data point [True]
48 Text feature [indicated] present in test data point [True]
49 Text feature [transcriptional] present in test data point [True]
51 Text feature [likely] present in test data point [True]
54 Text feature [gene] present in test data point [True]
55 Text feature [essential] present in test data point [True]
56 Text feature [loss] present in test data point [True]
57 Text feature [sequences] present in test data point [True]
58 Text feature [genes] present in test data point [True]
59 Text feature [control] present in test data point [True]
60 Text feature [five] present in test data point [True]
61 Text feature [data] present in test data point [True]
62 Text feature [also] present in test data point [True]
63 Text feature [least] present in test data point [True]
65 Text feature [within] present in test data point [True]
66 Text feature [table] present in test data point [True]
67 Text feature [terminal] present in test data point [True]
68 Text feature [six] present in test data point [True]
69 Text feature [domains] present in test data point [True]
72 Text feature [using] present in test data point [True]
73 Text feature [frameshift] present in test data point [True]
74 Text feature [located] present in test data point [True]
77 Text feature [result] present in test data point [True]
79 Text feature [whereas] present in test data point [True]
80 Text feature [specificity] present in test data point [True]
```

```
83 Text feature [genetic] present in test data point [True]
84 Text feature [shown] present in test data point [True]
86 Text feature [37] present in test data point [True]
87 Text feature [identified] present in test data point [True]
88 Text feature [human] present in test data point [True]
89 Text feature [either] present in test data point [True]
93 Text feature [fraction] present in test data point [True]
94 Text feature [cancer] present in test data point [True]
95 Text feature [changes] present in test data point [True]
96 Text feature [see] present in test data point [True]
97 Text feature [proteins] present in test data point [True]
Out of the top  100  features  67 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [60]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4406 0.097  0.0221 0.2719 0.0517
0.0403 0.0651 0.0063 0.0051]]
Actual Class : 1
--------------------------------------------------
11 Text feature [function] present in test data point [True]
12 Text feature [dna] present in test data point [True]
13 Text feature [type] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [affect] present in test data point [True]
16 Text feature [wild] present in test data point [True]
17 Text feature [one] present in test data point [True]
18 Text feature [two] present in test data point [True]
19 Text feature [binding] present in test data point [True]
23 Text feature [sequence] present in test data point [True]
25 Text feature [possible] present in test data point [True]
26 Text feature [amino] present in test data point [True]
27 Text feature [therefore] present in test data point [True]
28 Text feature [containing] present in test data point [True]
29 Text feature [effect] present in test data point [True]
30 Text feature [functions] present in test data point [True]
31 Text feature [region] present in test data point [True]
34 Text feature [specific] present in test data point [True]
36 Text feature [corresponding] present in test data point [True]
37 Text feature [involved] present in test data point [True]
39 Text feature [form] present in test data point [True]
40 Text feature [results] present in test data point [True]
41 Text feature [three] present in test data point [True]
42 Text feature [structure] present in test data point [True]
43 Text feature [used] present in test data point [True]
44 Text feature [surface] present in test data point [True]
45 Text feature [analysis] present in test data point [True]
46 Text feature [indicating] present in test data point [True]
47 Text feature [indicate] present in test data point [True]
48 Text feature [indicated] present in test data point [True]
49 Text feature [transcriptional] present in test data point [True]
50 Text feature [determined] present in test data point [True]
51 Text feature [likely] present in test data point [True]
52 Text feature [ability] present in test data point [True]
54 Text feature [gene] present in test data point [True]
55 Text feature [essential] present in test data point [True]
56 Text feature [loss] present in test data point [True]
58 Text feature [genes] present in test data point [True]
59 Text feature [control] present in test data point [True]
61 Text feature [data] present in test data point [True]
62 Text feature [also] present in test data point [True]
64 Text feature [conserved] present in test data point [True]
67 Text feature [terminal] present in test data point [True]
69 Text feature [domains] present in test data point [True]
70 Text feature [contains] present in test data point [True]
72 Text feature [using] present in test data point [True]
74 Text feature [located] present in test data point [True]
76 Text feature [directly] present in test data point [True]
77 Text feature [result] present in test data point [True]
78 Text feature [several] present in test data point [True]
79 Text feature [whereas] present in test data point [True]
80 Text feature [specificity] present in test data point [True]
81 Text feature [expected] present in test data point [True]
82 Text feature [addition] present in test data point [True]
84 Text feature [shown] present in test data point [True]
85 Text feature [specifically] present in test data point [True]
```

```
86 Text feature [37] present in test data point [True]
87 Text feature [identified] present in test data point [True]
88 Text feature [human] present in test data point [True]
89 Text feature [either] present in test data point [True]
91 Text feature [important] present in test data point [True]
94 Text feature [cancer] present in test data point [True]
95 Text feature [changes] present in test data point [True]
97 Text feature [proteins] present in test data point [True]
98 Text feature [whether] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

4.2. K Nearest Neighbour Classification

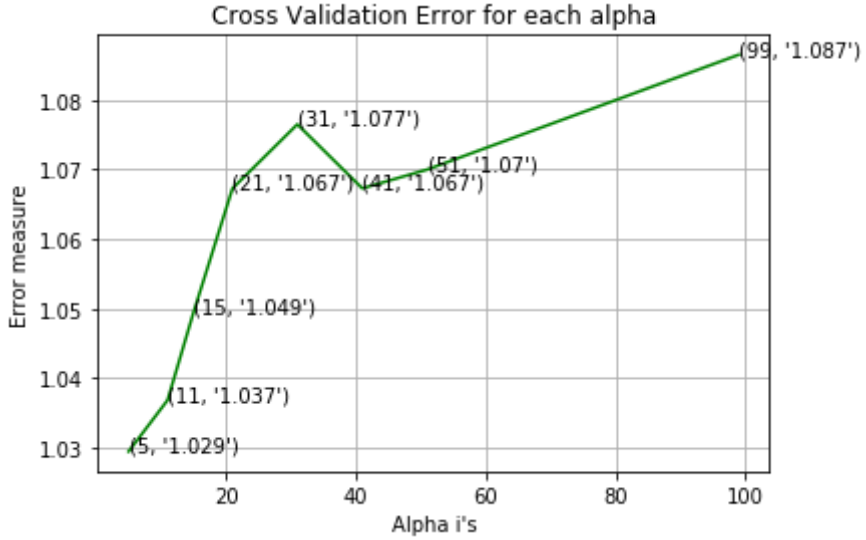
4.2.1. Hyper parameter tuning

In [61]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
d', 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 = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probabil
ity estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.029475678581944
for alpha = 11
Log Loss : 1.0368582543693807
for alpha = 15
Log Loss : 1.0494777165689622
for alpha = 21
Log Loss : 1.067287894836108
for alpha = 31
Log Loss : 1.0765042082855945
for alpha = 41
Log Loss : 1.0672998240699056
for alpha = 51
Log Loss : 1.070002039973082
for alpha = 99
Log Loss : 1.0865130548981816
```



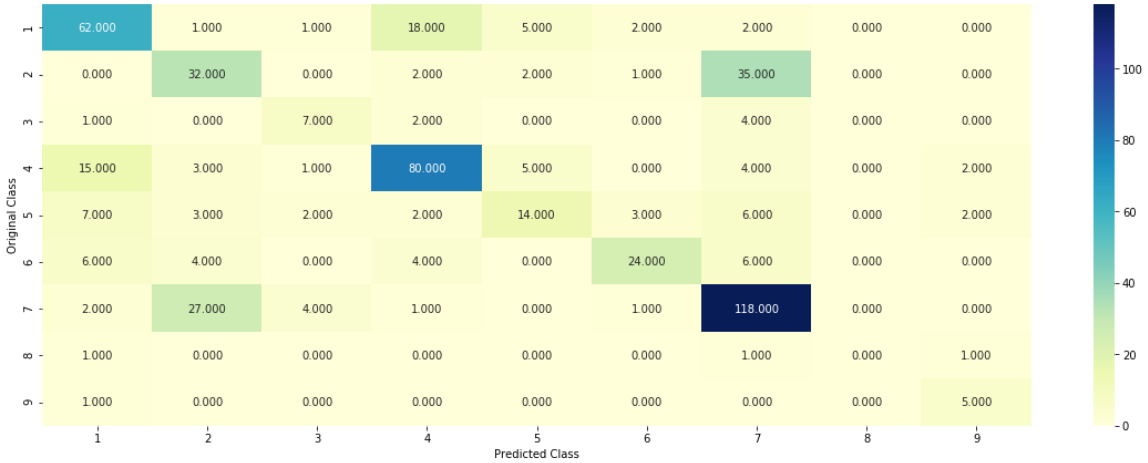

```
For values of best alpha =  5 The train log loss is: 0.5085374993963
868
For values of best alpha =  5 The cross validation log loss is: 1.02
9475678581944
For values of best alpha =  5 The test log loss is: 1.05129053144744
```

4.2.2. Testing the model with best hyper paramters

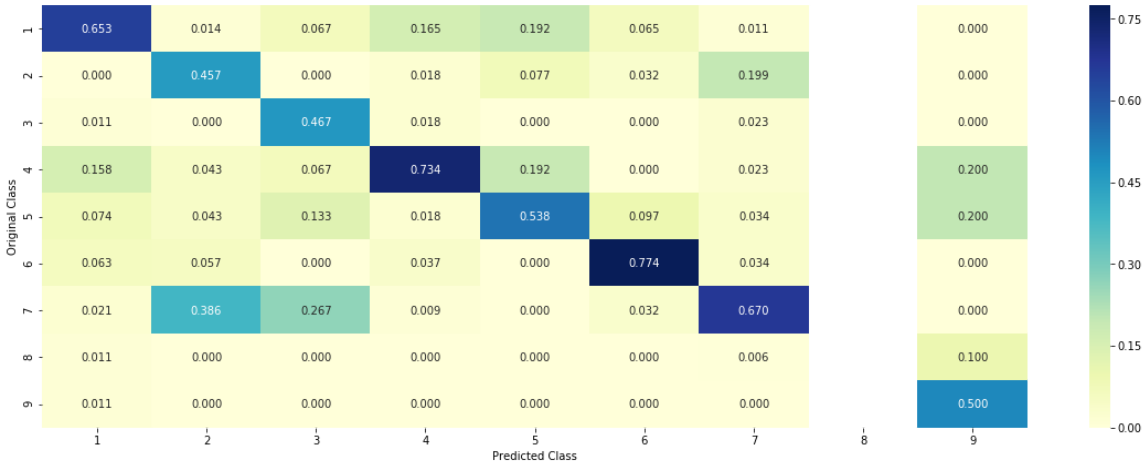
In [62]:

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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

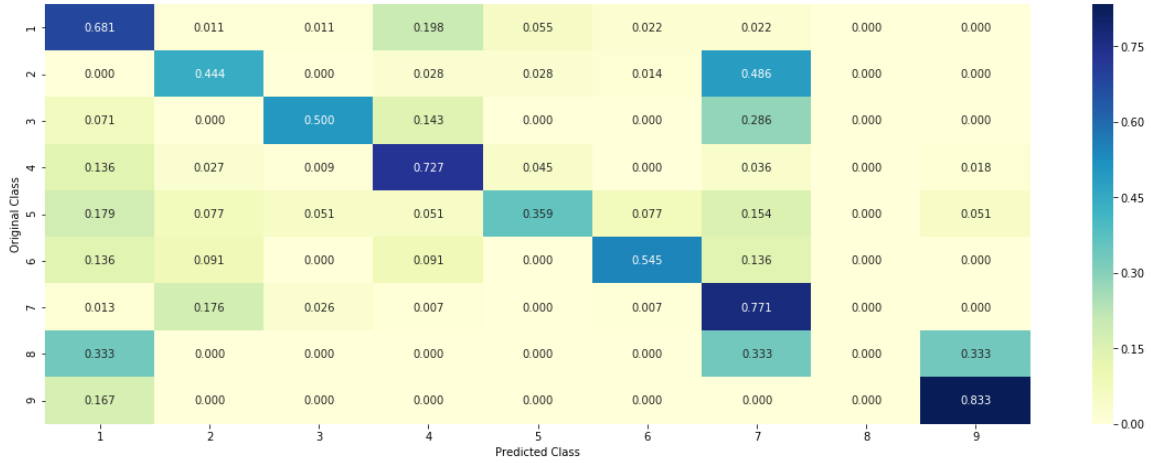
```
Log loss : 1.029475678581944
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.2.3.Sample Query point -1

In [63]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1
), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs t
o classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 1
The  5  nearest neighbours of the test points belongs to classes [1
1 1 1 1]
Fequency of nearest points : Counter({1: 5})
```

4.2.4. Sample Query Point-2

In [64]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(
1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1
), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of
 the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 1
the k value for knn is 5 and the nearest neighbours of the test poin
ts belongs to classes [1 1 4 1 1]
Fequency of nearest points : Counter({1: 4, 4: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [1]:

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

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

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


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
```

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

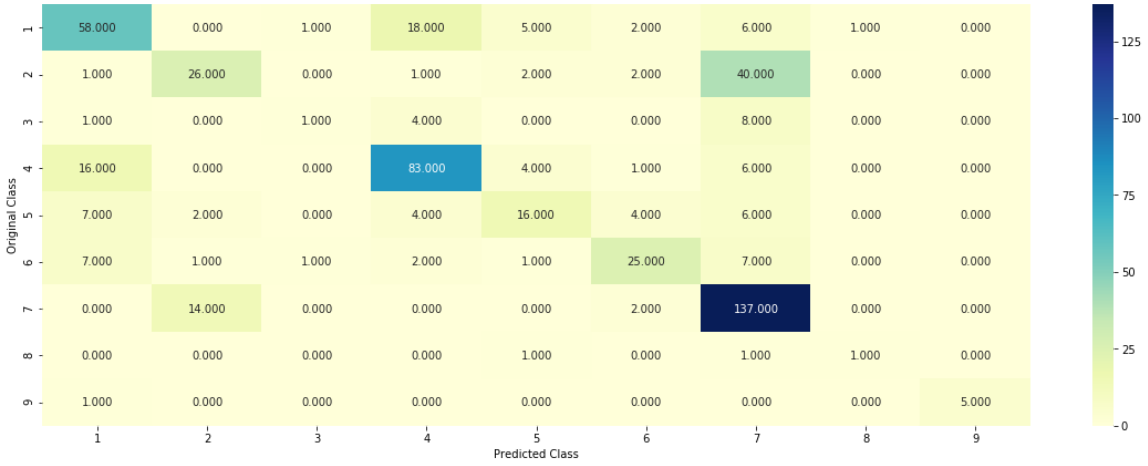
```
for alpha = 1e-06

---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-1-a29de4f76d64> in <module>()
     34 for i in alpha:
     35     print("for alpha =", i)
---> 36     clf = SGDClassifier(class_weight='balanced', alpha=i, pe
nalty='l2', loss='log', random_state=42)
     37     clf.fit(train_x_onehotCoding, train_y)
     38     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

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

4.3.1.2. Testing the model with best hyper paramters

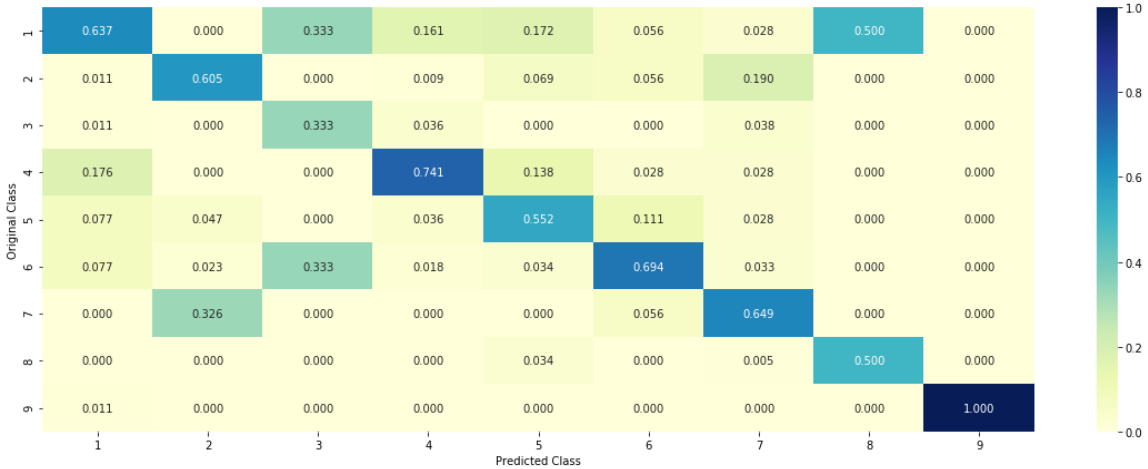
In [66]:

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

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

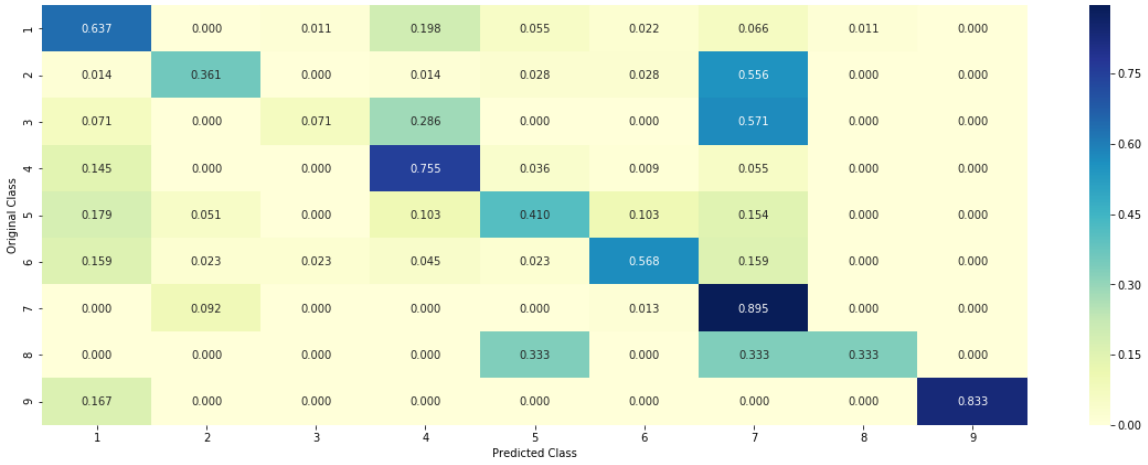
```
Log loss : 1.1550662240888696
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.3.1.3. Feature Importance

In [67]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_
no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point"
)
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," clas
s:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or N
ot']))
```

4.3.1.3.1. Correctly Classified point

In [68]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7653 0.0192 0.0107 0.0076 0.1746
0.0046 0.0072 0.0051 0.0058]]
Actual Class : 1
--------------------------------------------------
298 Text feature [aggregation] present in test data point [True]
304 Text feature [657del5] present in test data point [True]
319 Text feature [frameshift] present in test data point [True]
352 Text feature [truncating] present in test data point [True]
402 Text feature [bethesda] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [69]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5151 0.0855 0.02   0.1683 0.0266
0.0355 0.1291 0.006  0.0139]]
Actual Class : 1
--------------------------------------------------
113 Text feature [mj] present in test data point [True]
187 Text feature [carm1] present in test data point [True]
212 Text feature [ser217] present in test data point [True]
311 Text feature [ptm] present in test data point [True]
313 Text feature [grip1] present in test data point [True]
479 Text feature [immunize] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.3.2. Without Class balancing

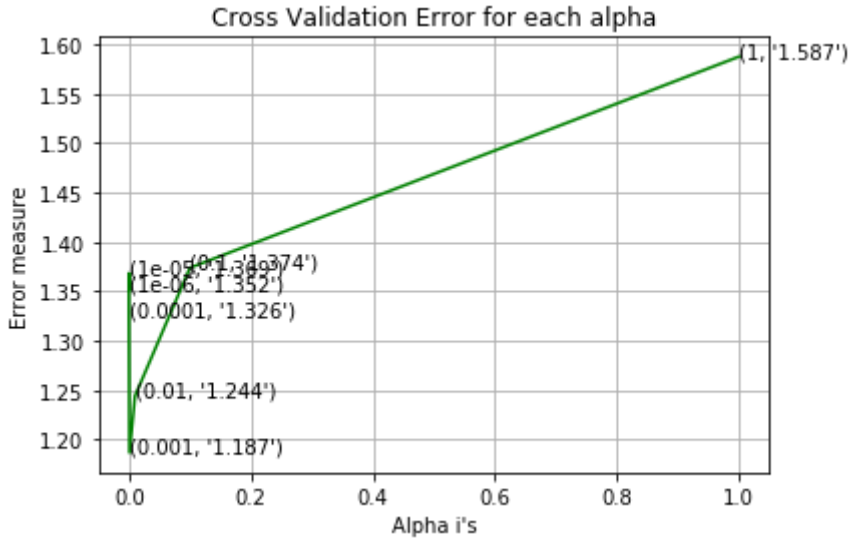
4.3.2.1. Hyper paramter tuning

In [70]:

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

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

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



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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
```

```
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.3517806791575715
for alpha = 1e-05
Log Loss : 1.3685017355883777
for alpha = 0.0001
Log Loss : 1.326045761719215
for alpha = 0.001
Log Loss : 1.1869919090826713
for alpha = 0.01
Log Loss : 1.2443115972293874
for alpha = 0.1
Log Loss : 1.3738595120322834
for alpha = 1
Log Loss : 1.5874128323760492
```



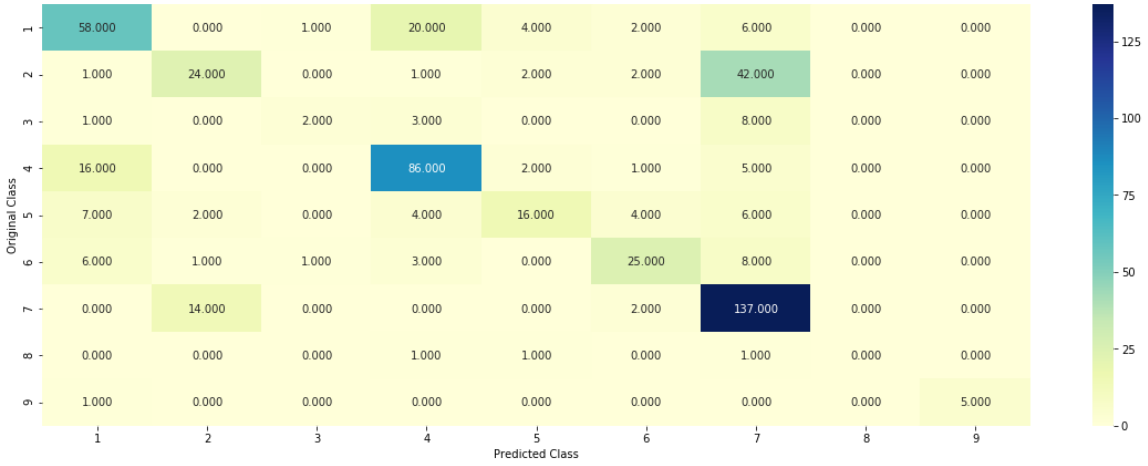

```
For values of best alpha =  0.001 The train log loss is: 0.654358803
8085142
For values of best alpha =  0.001 The cross validation log loss is:
1.1869919090826713
For values of best alpha =  0.001 The test log loss is: 1.1034525883
868787
```

4.3.2.2. Testing model with best hyper parameters

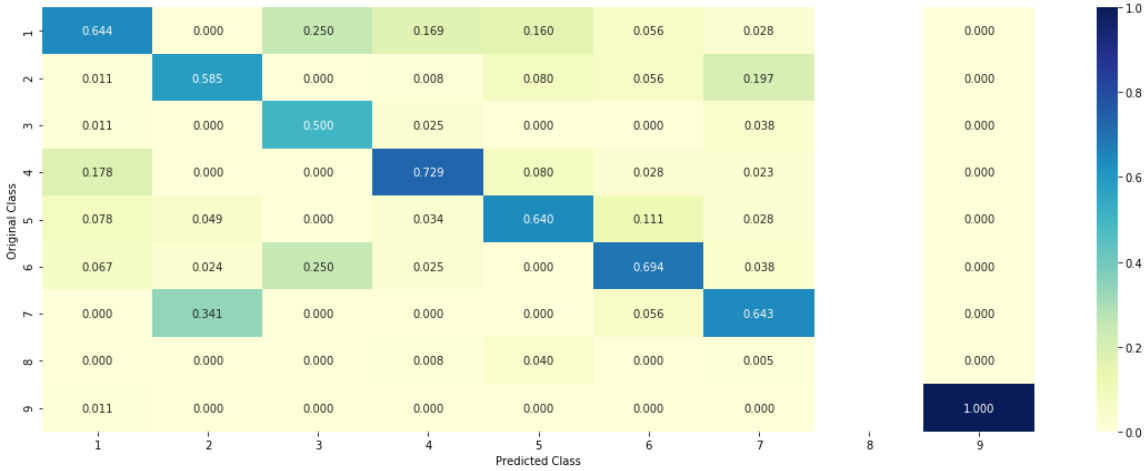
In [71]:

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

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

#-------------------------------
# video link:
#-------------------------------

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

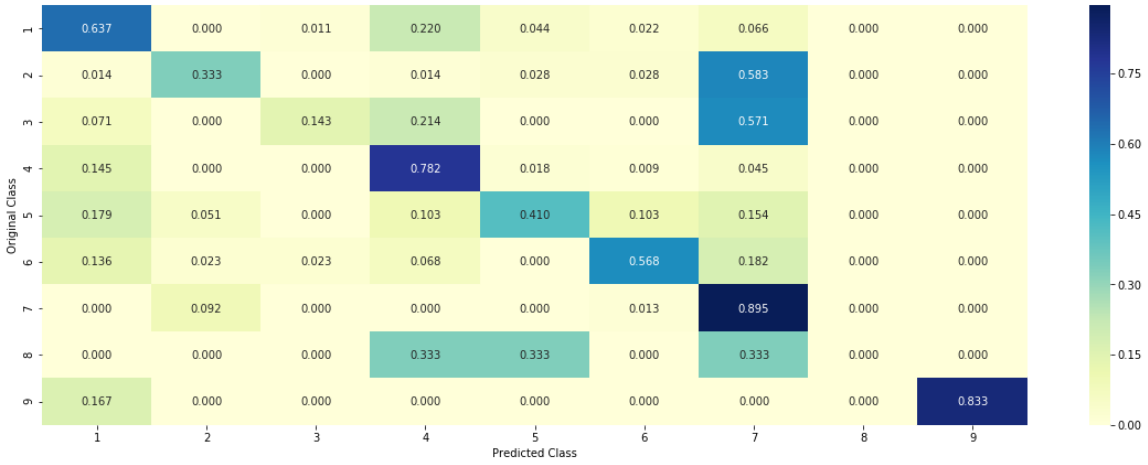
```
Log loss : 1.1869919090826713
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.3.2.3. Feature Importance, Correctly Classified point

In [72]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8081 0.0229 0.003  0.0114 0.1287
0.004  0.0139 0.0057 0.0022]]
Actual Class : 1
--------------------------------------------------
337 Text feature [frameshift] present in test data point [True]
351 Text feature [657del5] present in test data point [True]
353 Text feature [truncating] present in test data point [True]
408 Text feature [aggregation] present in test data point [True]
436 Text feature [bethesda] present in test data point [True]
453 Text feature [project] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [73]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5226 0.0922 0.0124 0.164  0.0277
0.0349 0.1307 0.0062 0.0094]]
Actual Class : 1
--------------------------------------------------
124 Text feature [mj] present in test data point [True]
158 Text feature [carm1] present in test data point [True]
182 Text feature [ser217] present in test data point [True]
266 Text feature [ptm] present in test data point [True]
267 Text feature [grip1] present in test data point [True]
451 Text feature [immunize] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.4. Linear Support Vector Machines

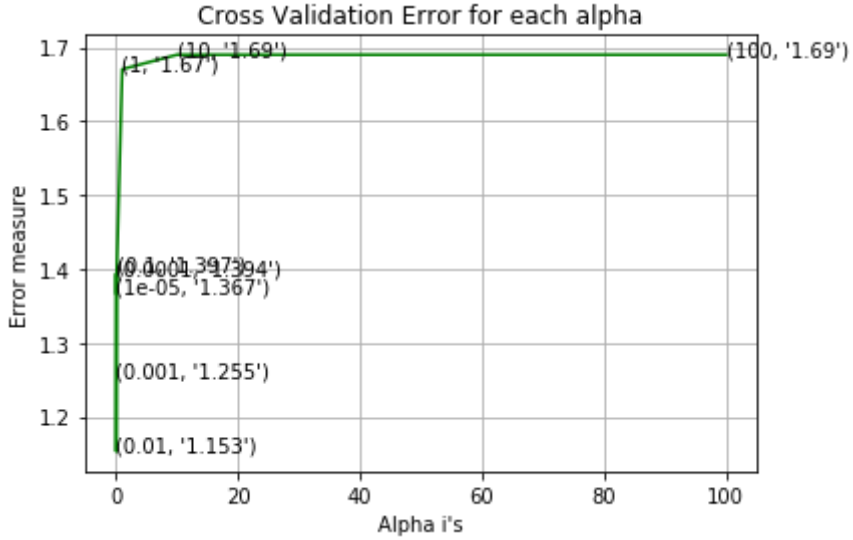
4.4.1. Hyper paramter tuning

In [74]:

```
# read more about support vector machines with linear kernals here http://scikit
-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, pr
obability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_functi
on_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given trainin
g data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/le
ssons/mathematical-derivation-copy-8/
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
d', 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 = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='h
inge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
```

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.3673181464400834
for C = 0.0001
Log Loss : 1.3936754741594692
for C = 0.001
Log Loss : 1.2546259097364527
for C = 0.01
Log Loss : 1.1526875553897424
for C = 0.1
Log Loss : 1.3967099423443512
for C = 1
Log Loss : 1.670352083560736
for C = 10
Log Loss : 1.6900076823378543
for C = 100
Log Loss : 1.6900076884687385
```



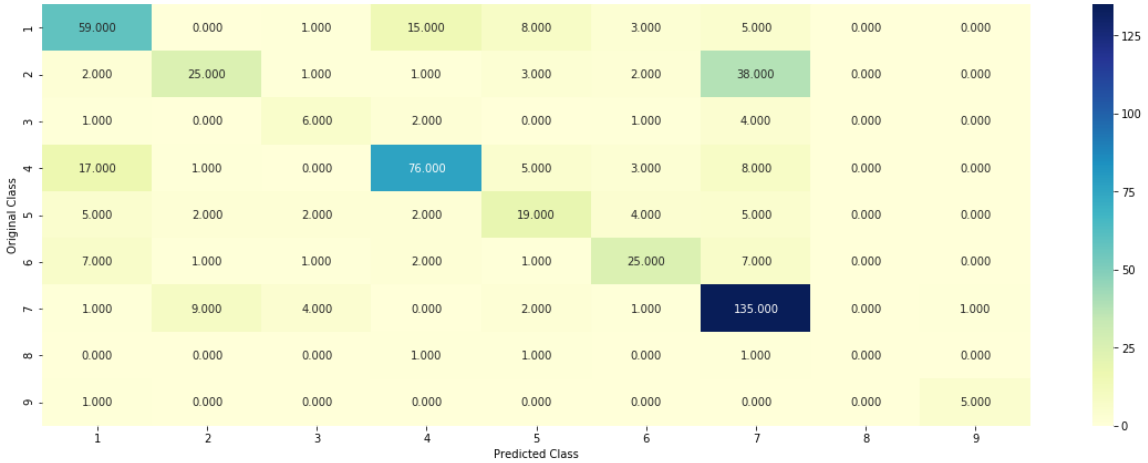

```
For values of best alpha =  0.01 The train log loss is: 0.7657742540
182977
For values of best alpha =  0.01 The cross validation log loss is:
1.1526875553897424
For values of best alpha =  0.01 The test log loss is: 1.15400526791
62704
```

4.4.2. Testing model with best hyper parameters

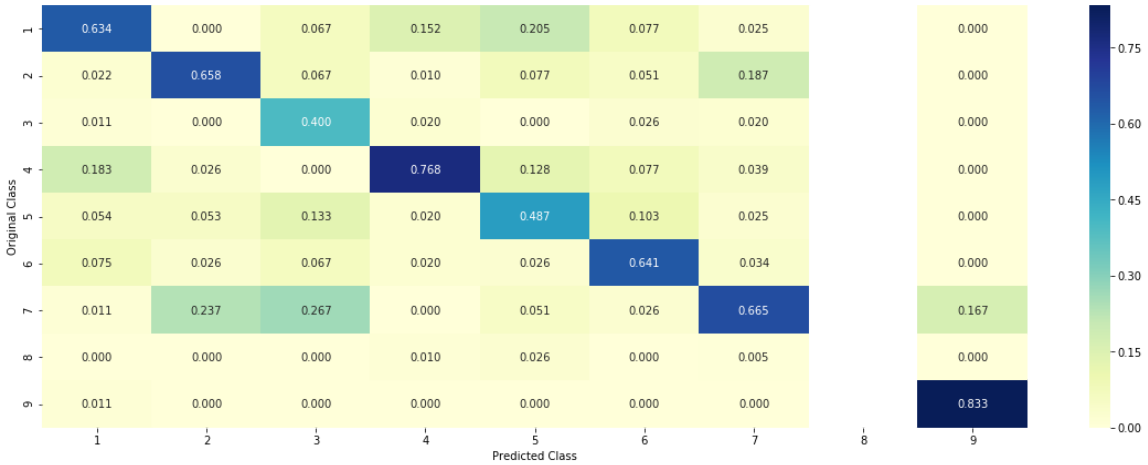
In [75]:

```
# read more about support vector machines with linear kernals here http://scikit
-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, pr
obability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_functi
on_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given trainin
g data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/le
ssons/mathematical-derivation-copy-8/
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight
='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_
state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCodin
g,cv_y, clf)
```

```
Log loss : 1.1526875553897424
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ----------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [76]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8494 0.0406 0.0059 0.0172 0.0326
0.011  0.0348 0.0047 0.0038]]
Actual Class : 1
--------------------------------------------------
50 Text feature [aggregation] present in test data point [True]
55 Text feature [657del5] present in test data point [True]
101 Text feature [bethesda] present in test data point [True]
149 Text feature [project] present in test data point [True]
159 Text feature [frameshift] present in test data point [True]
162 Text feature [ercc2] present in test data point [True]
196 Text feature [genotypic] present in test data point [True]
207 Text feature [nbn] present in test data point [True]
212 Text feature [ner] present in test data point [True]
242 Text feature [alamut] present in test data point [True]
245 Text feature [prone] present in test data point [True]
323 Text feature [truncating] present in test data point [True]
348 Text feature [deficient] present in test data point [True]
369 Text feature [families] present in test data point [True]
453 Text feature [1903] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [77]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df[
'Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_f
eature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4783 0.0865 0.016  0.1638 0.0477
0.042  0.1486 0.007  0.0101]]
Actual Class : 1
--------------------------------------------------
44 Text feature [mj] present in test data point [True]
96 Text feature [baculoviral] present in test data point [True]
166 Text feature [surface] present in test data point [True]
262 Text feature [young] present in test data point [True]
348 Text feature [deficient] present in test data point [True]
373 Text feature [carm1] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [78]:

```
# ------------------------------
# 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_lea
f_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
te=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 trainin
g 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/le
ssons/random-forest-and-their-construction-2/
# ------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
d', 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 = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth
=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_
log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='g
ini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train lo
g loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross va
lidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log
 loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

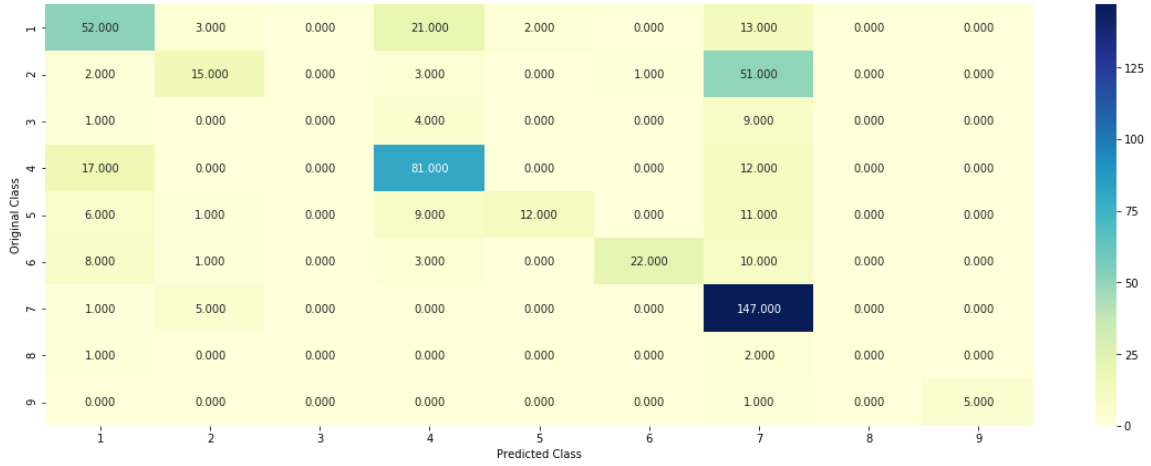
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.245543646597739
for n_estimators = 100 and max depth =  10
Log Loss : 1.1731501324539038
for n_estimators = 200 and max depth =  5
Log Loss : 1.2371426615265606
for n_estimators = 200 and max depth =  10
Log Loss : 1.1608936510696237
for n_estimators = 500 and max depth =  5
Log Loss : 1.2319227176377927
for n_estimators = 500 and max depth =  10
Log Loss : 1.1547828684672465
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2322867316845485
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1550058934211838
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2330183536119812
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1524716181856265
For values of best estimator =  2000 The train log loss is: 0.702060
9670244018
For values of best estimator =  2000 The cross validation log loss i
s: 1.1524716200427836
For values of best estimator =  2000 The test log loss is: 1.1047994
119243285
```

4.5.2. Testing model with best hyper parameters (One Hot Encoding)

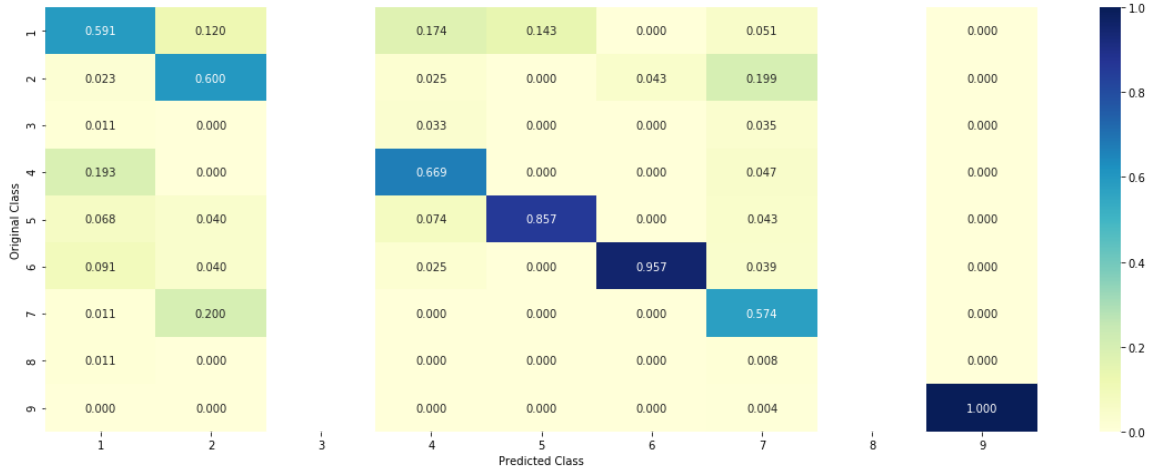
In [79]:

```
# ------------------------------
# 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_lea
f_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
te=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 trainin
g 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/le
ssons/random-forest-and-their-construction-2/
# ------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='g
ini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCodin
g,cv_y, clf)
```

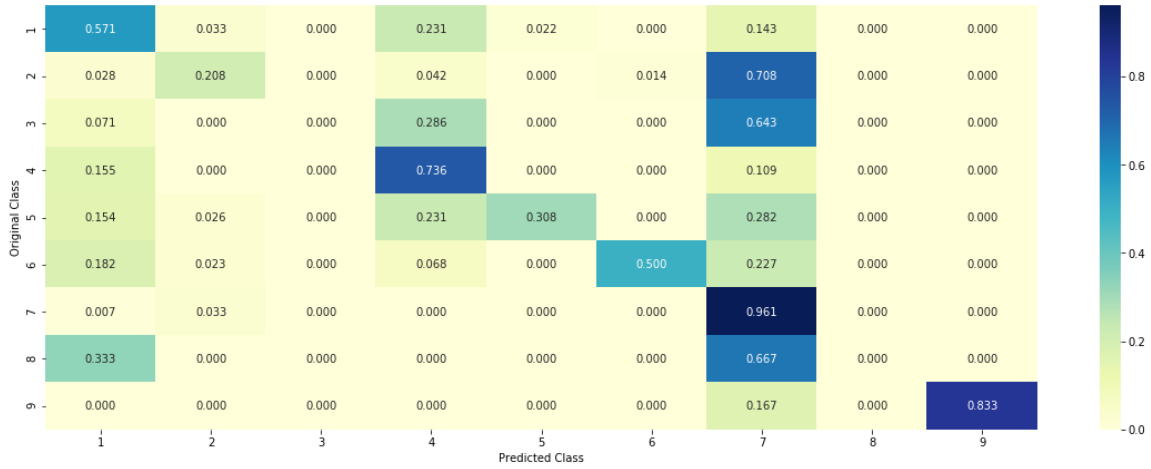
```
Log loss : 1.1524716200427834
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [80]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='g
ini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_in
dex], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6932 0.0345 0.0154 0.1122 0.0597
0.0408 0.0323 0.0048 0.0072]]
Actual Class : 1
--------------------------------------------------
3 Text feature [activation] present in test data point [True]
5 Text feature [function] present in test data point [True]
10 Text feature [missense] present in test data point [True]
19 Text feature [brca1] present in test data point [True]
26 Text feature [patients] present in test data point [True]
31 Text feature [functional] present in test data point [True]
33 Text feature [stability] present in test data point [True]
34 Text feature [cells] present in test data point [True]
36 Text feature [variants] present in test data point [True]
40 Text feature [loss] present in test data point [True]
46 Text feature [cell] present in test data point [True]
50 Text feature [brca2] present in test data point [True]
51 Text feature [deleterious] present in test data point [True]
52 Text feature [pathogenic] present in test data point [True]
54 Text feature [efficacy] present in test data point [True]
60 Text feature [protein] present in test data point [True]
64 Text feature [repair] present in test data point [True]
71 Text feature [pathogenicity] present in test data point [True]
72 Text feature [carriers] present in test data point [True]
75 Text feature [response] present in test data point [True]
77 Text feature [transformation] present in test data point [True]
83 Text feature [amplification] present in test data point [True]
85 Text feature [dna] present in test data point [True]
88 Text feature [treated] present in test data point [True]
89 Text feature [proteins] present in test data point [True]
91 Text feature [potential] present in test data point [True]
99 Text feature [kit] present in test data point [True]
Out of the top  100  features  27 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [81]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_on
ehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_in
dex], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3058 0.0721 0.0218 0.3348 0.0577
0.0522 0.1343 0.007  0.0142]]
Actuall Class : 1
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [signaling] present in test data point [True]
16 Text feature [activated] present in test data point [True]
17 Text feature [growth] present in test data point [True]
18 Text feature [constitutively] present in test data point [True]
21 Text feature [yeast] present in test data point [True]
25 Text feature [therapeutic] present in test data point [True]
28 Text feature [receptor] present in test data point [True]
31 Text feature [functional] present in test data point [True]
33 Text feature [stability] present in test data point [True]
34 Text feature [cells] present in test data point [True]
37 Text feature [lines] present in test data point [True]
39 Text feature [inhibition] present in test data point [True]
40 Text feature [loss] present in test data point [True]
42 Text feature [kinases] present in test data point [True]
45 Text feature [phospho] present in test data point [True]
46 Text feature [cell] present in test data point [True]
60 Text feature [protein] present in test data point [True]
75 Text feature [response] present in test data point [True]
78 Text feature [nuclear] present in test data point [True]
80 Text feature [phosphatase] present in test data point [True]
85 Text feature [dna] present in test data point [True]
88 Text feature [treated] present in test data point [True]
89 Text feature [proteins] present in test data point [True]
91 Text feature [potential] present in test data point [True]
94 Text feature [ligand] present in test data point [True]
98 Text feature [pathway] present in test data point [True]
99 Text feature [kit] present in test data point [True]
Out of the top  100  features  33 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [82]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max
_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_lea
f_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
te=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 trainin
g 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/le
ssons/random-forest-and-their-construction-2/
# ------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
d', 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 = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth
=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
```

```
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_
log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='g
ini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log lo
ss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross valida
tion log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log los
s is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

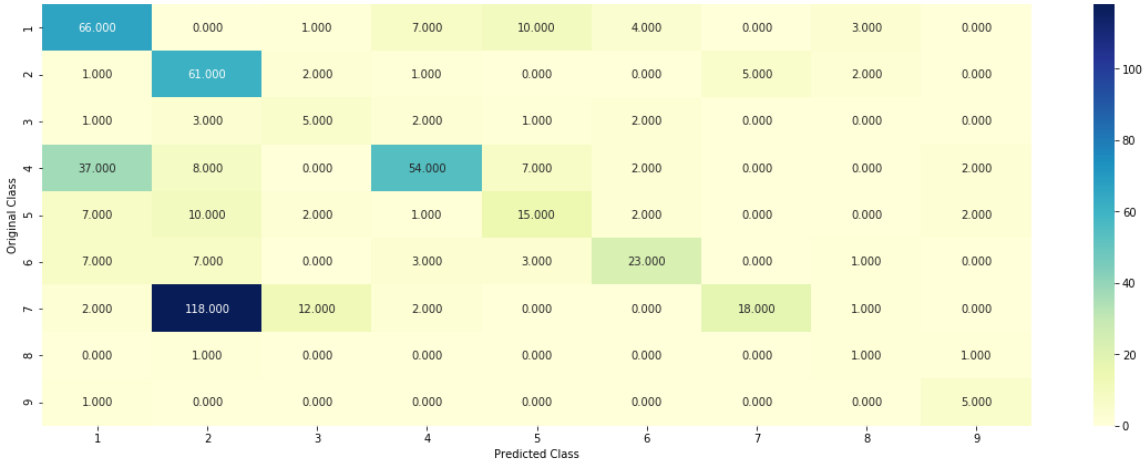
```
for n_estimators = 10 and max depth =  2
Log Loss : 1.970401851694984
for n_estimators = 10 and max depth =  3
Log Loss : 1.5507915671585886
for n_estimators = 10 and max depth =  5
Log Loss : 1.4946674057396008
for n_estimators = 10 and max depth =  10
Log Loss : 1.6679742967986104
for n_estimators = 50 and max depth =  2
Log Loss : 1.637803259340415
for n_estimators = 50 and max depth =  3
Log Loss : 1.2991841469892866
for n_estimators = 50 and max depth =  5
Log Loss : 1.2975182498442042
for n_estimators = 50 and max depth =  10
Log Loss : 1.7076961193784737
for n_estimators = 100 and max depth =  2
Log Loss : 1.5723914201220348
for n_estimators = 100 and max depth =  3
Log Loss : 1.3857901557134258
for n_estimators = 100 and max depth =  5
Log Loss : 1.3400198698567038
for n_estimators = 100 and max depth =  10
Log Loss : 1.759269243618616
for n_estimators = 200 and max depth =  2
Log Loss : 1.6126080890552648
for n_estimators = 200 and max depth =  3
Log Loss : 1.4620604393544918
for n_estimators = 200 and max depth =  5
Log Loss : 1.344650737507763
for n_estimators = 200 and max depth =  10
Log Loss : 1.6788531508830467
for n_estimators = 500 and max depth =  2
Log Loss : 1.6695497948727025
for n_estimators = 500 and max depth =  3
Log Loss : 1.5333817852543772
for n_estimators = 500 and max depth =  5
Log Loss : 1.3588478762332967
for n_estimators = 500 and max depth =  10
Log Loss : 1.7015879092127604
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6385691418471833
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5247825646878157
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3449458976984905
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7221258383811167
For values of best alpha =  50 The train log loss is: 0.062520619493
5509
For values of best alpha =  50 The cross validation log loss is: 1.2
975182498442044
For values of best alpha =  50 The test log loss is: 1.2748028330584
789
```

4.5.4. Testing model with best hyper parameters (Response Coding)

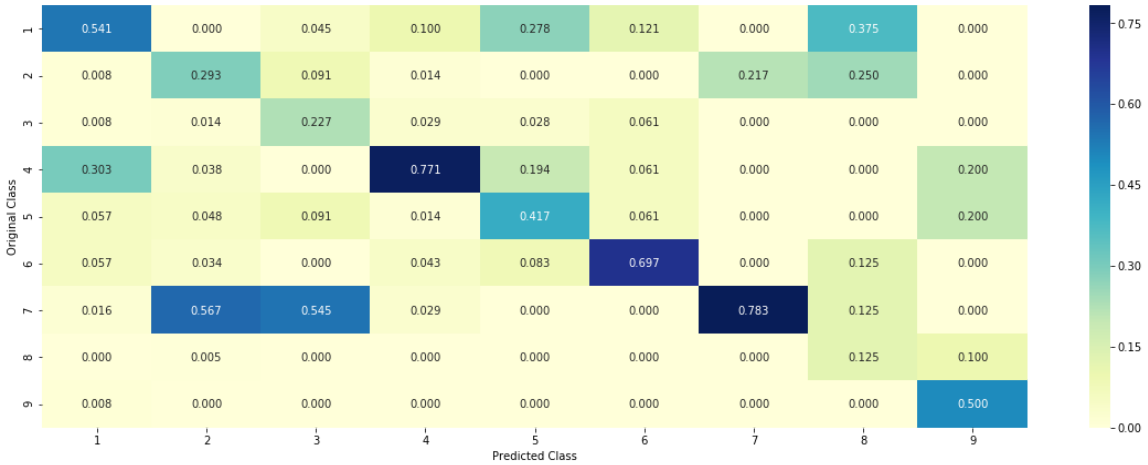
In [83]:

```
# --------------------------------
# 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_lea
f_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
te=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 trainin
g 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/le
ssons/random-forest-and-their-construction-2/
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimator
s=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_state=4
2)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseC
oding,cv_y, clf)
```

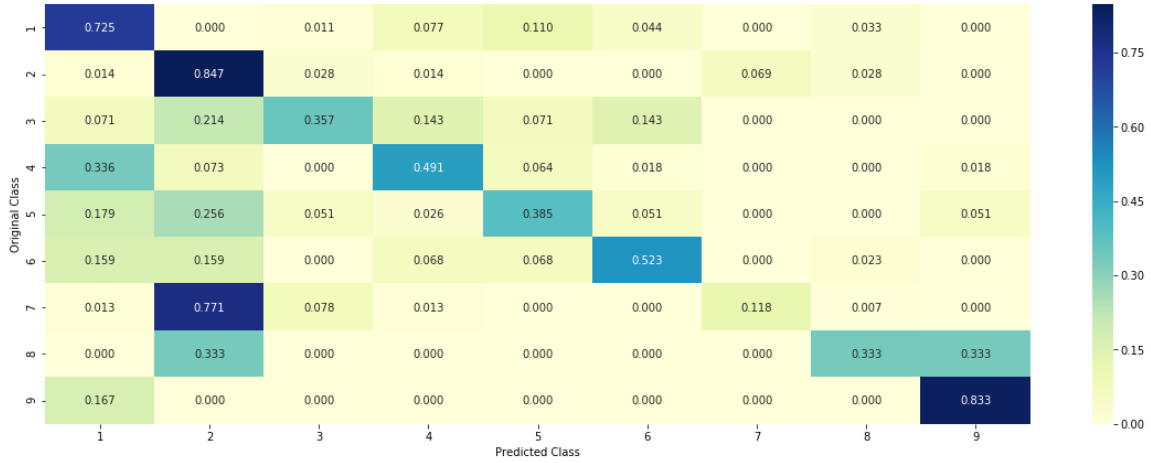
```
Log loss : 1.2975182498442044
Number of mis-classified points : 0.5338345864661654
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [84]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='g
ini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(
1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_re
sponseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5896 0.0332 0.0336 0.0473 0.0543
0.0835 0.013  0.0837 0.0617]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
```

4.5.5.2 Incorrectly Classified point

In [85]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(
1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_re
sponseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6899 0.0351 0.0411 0.0731 0.0189
0.0358 0.0082 0.0569 0.041 ]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [86]:

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

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

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


# read more about support vector machines with linear kernals here http://scikit
-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, pr
obability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_functi
on_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given trainin
g data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/le
ssons/mathematical-derivation-copy-8/
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit
-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# 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_lea
f_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
te=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 trainin
g 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/
# --------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

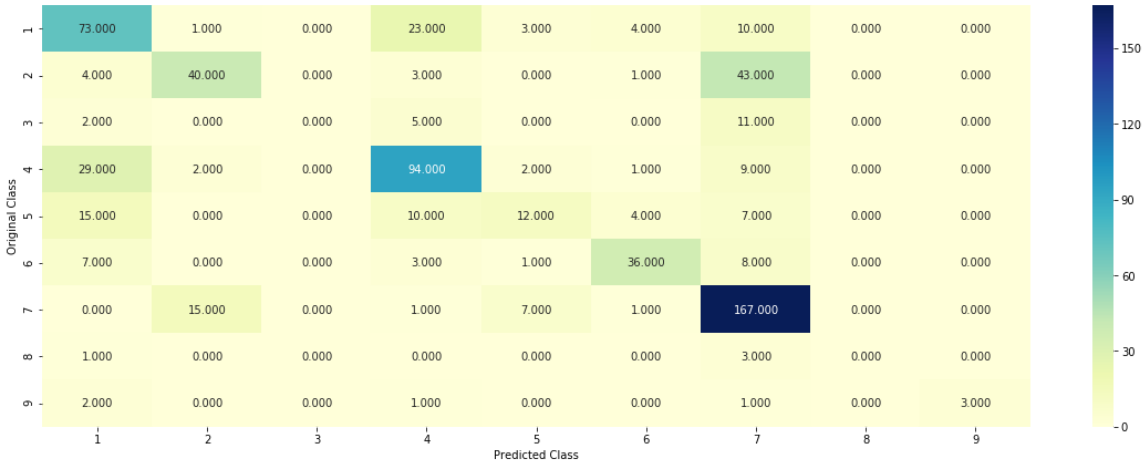
```
Logistic Regression :  Log Loss: 1.16
Support vector machines : Log Loss: 1.67
Naive Bayes : Log Loss: 1.27
-------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.17
9
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.04
3
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.54
1
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.14
6
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.20
7
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.4
42
```

4.7.2 testing the model with the best hyper parameters

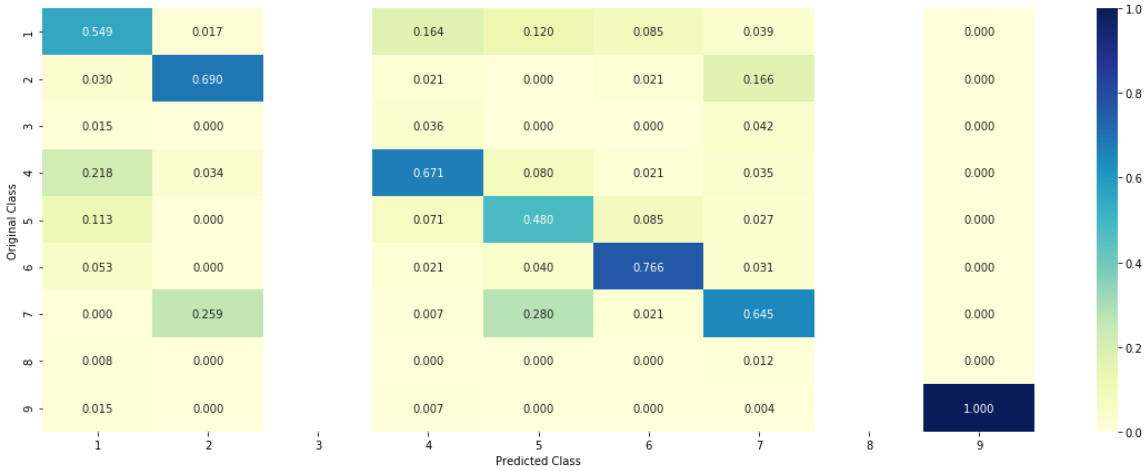
In [87]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_class
ifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_
onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding
))
```

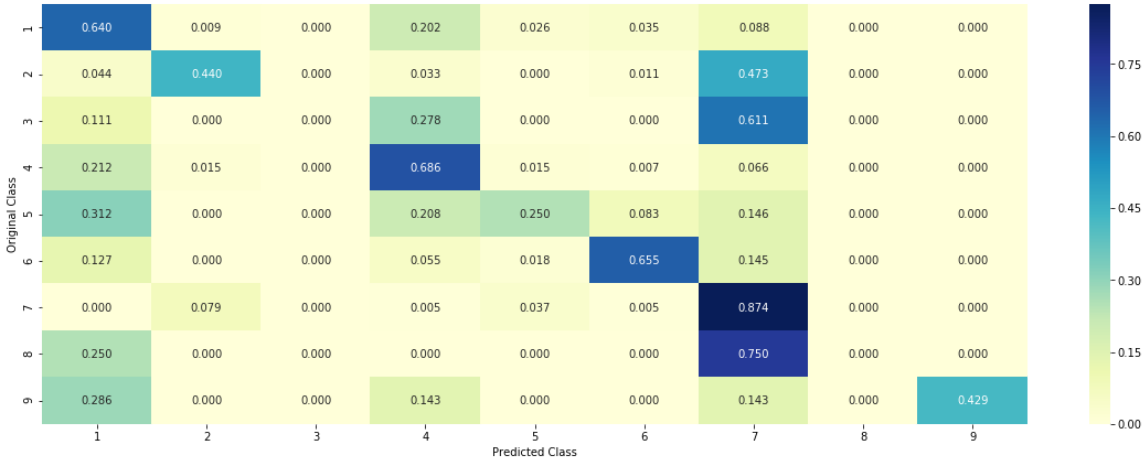
```
Log loss (train) on the stacking classifier : 0.6943286056627154
Log loss (CV) on the stacking classifier : 1.146407315829601
Log loss (test) on the stacking classifier : 1.1486933545771316
Number of missclassified point : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.7.3 Maximum Voting classifier

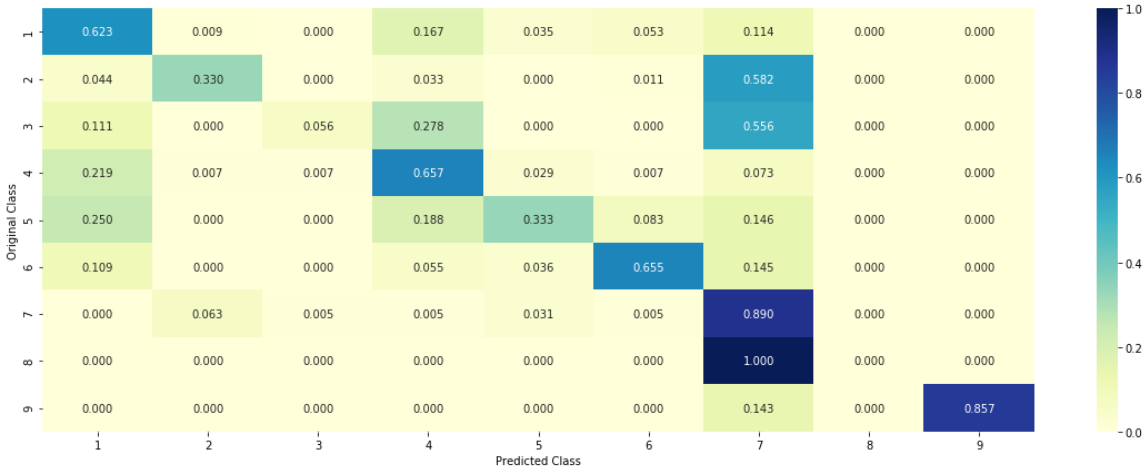
In [88]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.9441732358256435
Log loss (CV) on the VotingClassifier : 1.2224414817516507
Log loss (test) on the VotingClassifier : 1.2215980908103015
Number of missclassified point : 0.3684210526315789
--------------------- Confusion matrix --------------------
```


```
--------------------- Precision matrix (Columm Sum=1) ----------
-----
```


```
--------------------- Recall matrix (Row sum=1) --------------------
```



5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

A1 and A2

In [49]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),max_features=2000)
train_text_feature_onehotCoding  = tf_idf_vect.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = tf_idf_vect.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = tf_idf_vect.transform(cv_df['TEXT'])
feature_names = tf_idf_vect.get_feature_names()
```

In [50]:

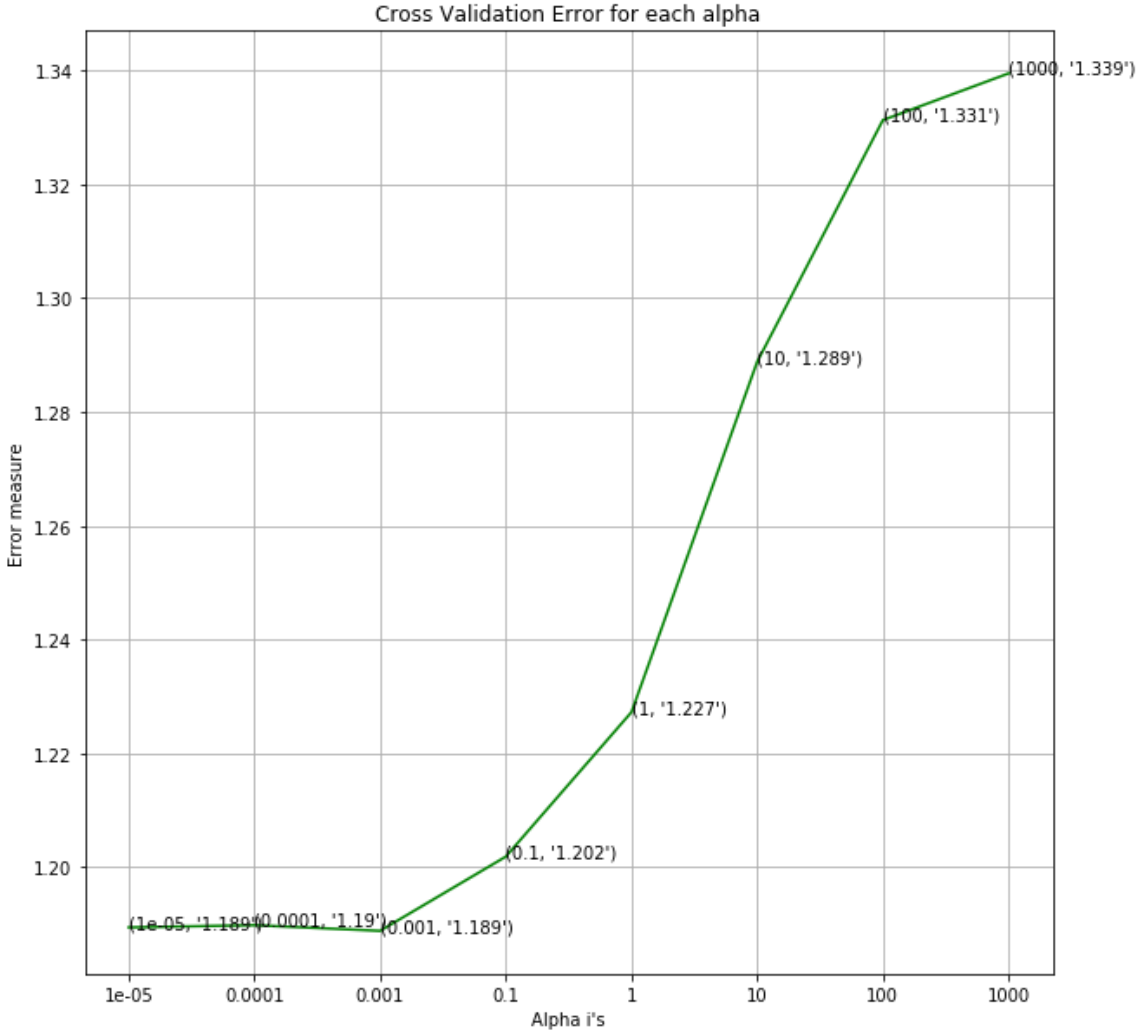
```
#stacking the gene ,variation and text feature together
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_vari
ation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
on_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea
ture_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_o
nehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_oneh
otCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCodi
ng)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

Naive Bayes

In [87]:

```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probabil
ity estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.1893940542371915
for alpha = 0.0001
Log Loss : 1.1897829993815463
for alpha = 0.001
Log Loss : 1.1887896629696313
for alpha = 0.1
Log Loss : 1.2018304165283495
for alpha = 1
Log Loss : 1.2272279967905388
for alpha = 10
Log Loss : 1.2887794413053584
for alpha = 100
Log Loss : 1.3312482156052137
for alpha = 1000
Log Loss : 1.3394564057745828
```

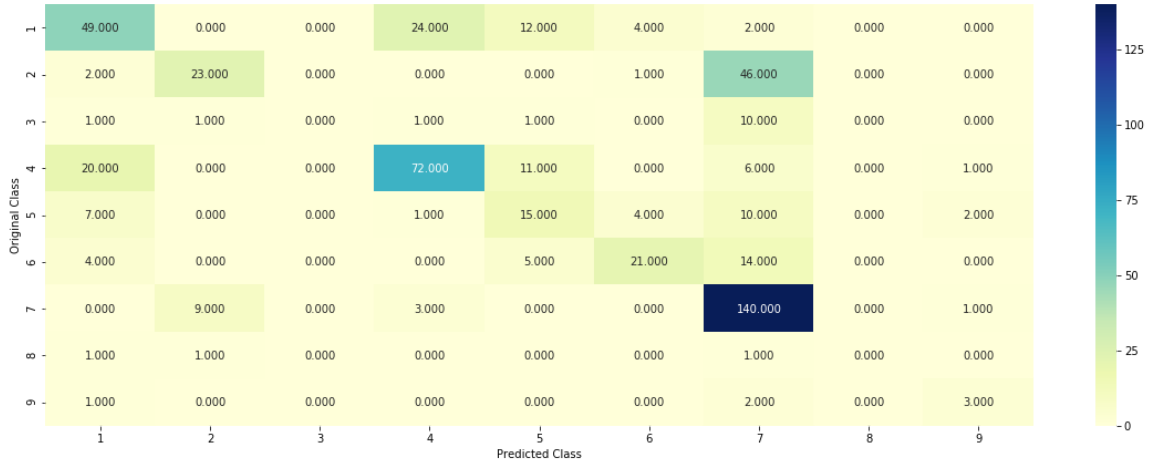


```
For values of best alpha =  0.001 The train log loss is: 0.560782759
478981
For values of best alpha =  0.001 The cross validation log loss is:
1.1887896629696313
For values of best alpha =  0.001 The test log loss is: 1.2364635036
76936
```

In [95]:

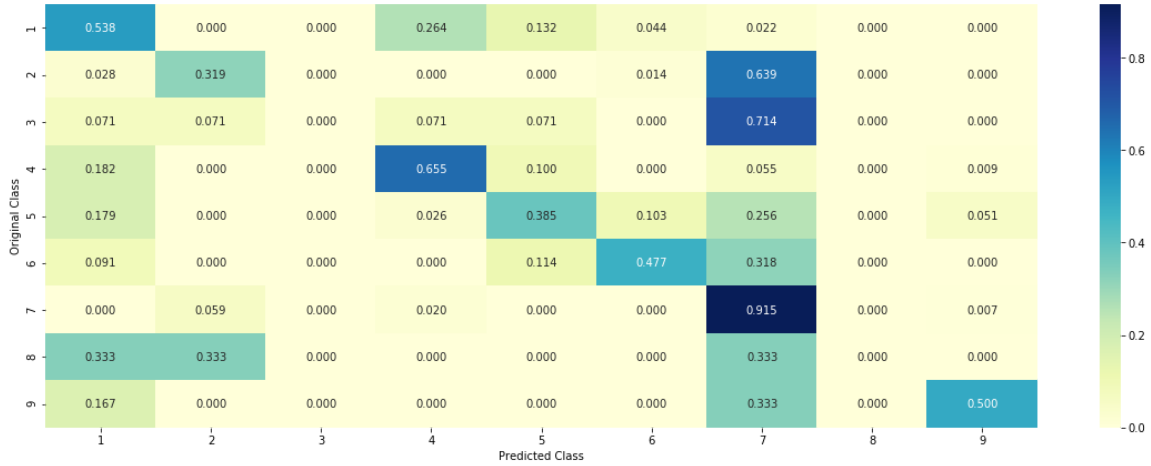
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability
 estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x
_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.1897829993815463
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ----------
-----
```


```
-------------------- Recall matrix (Row sum=1) --------------------
```

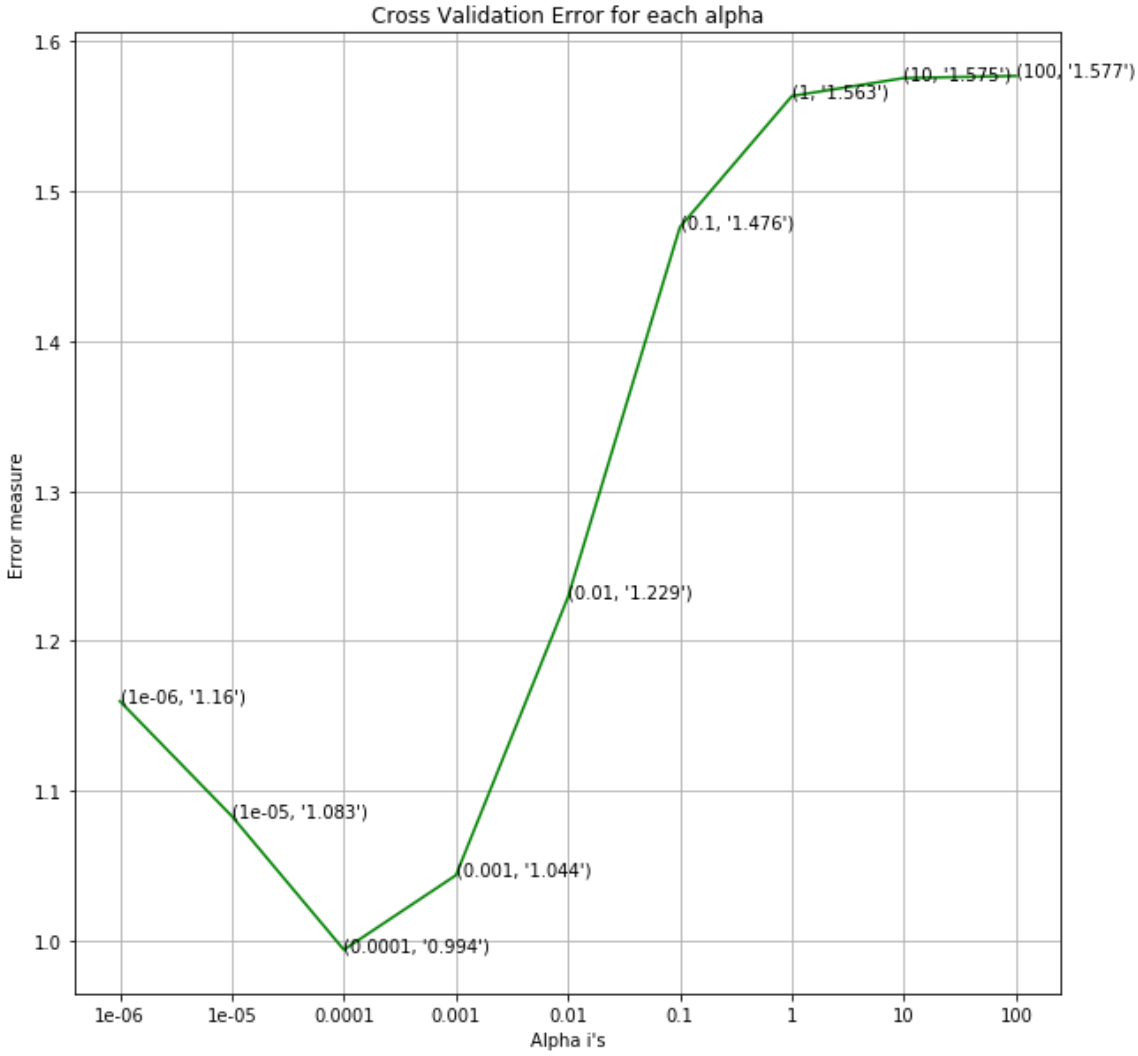


Logistic Regression

In [127]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='lo
g', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probabil
ity estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.159857329318207
for alpha = 1e-05
Log Loss : 1.083152764254492
for alpha = 0.0001
Log Loss : 0.9940981694004949
for alpha = 0.001
Log Loss : 1.0437814693675647
for alpha = 0.01
Log Loss : 1.2291689995367239
for alpha = 0.1
Log Loss : 1.4759490427191932
for alpha = 1
Log Loss : 1.5633490711746605
for alpha = 10
Log Loss : 1.5753157869716783
for alpha = 100
Log Loss : 1.576726303923752
```

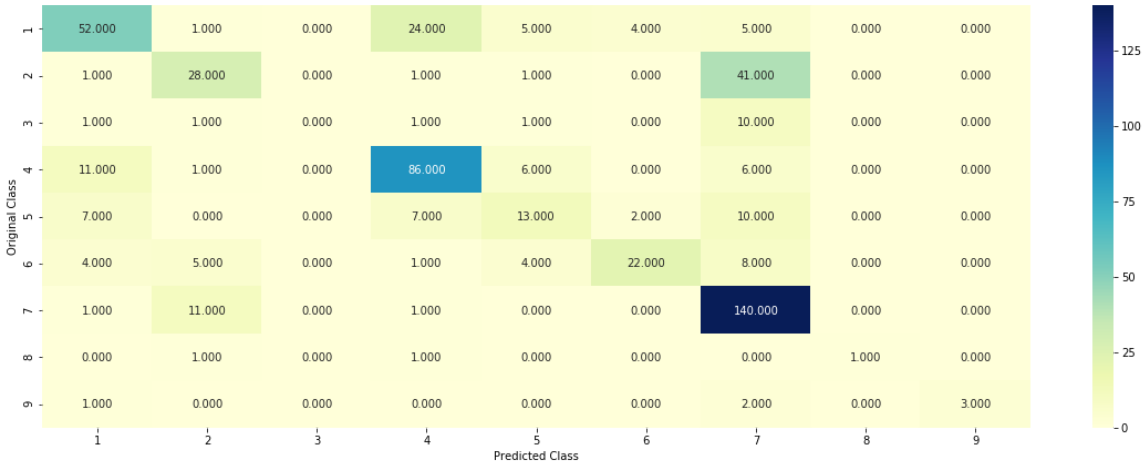



```
For values of best alpha =  0.0001 The train log loss is: 0.45139295
246902905
For values of best alpha =  0.0001 The cross validation log loss is:
0.9940981694004949
For values of best alpha =  0.0001 The test log loss is: 1.011373356
142744
```

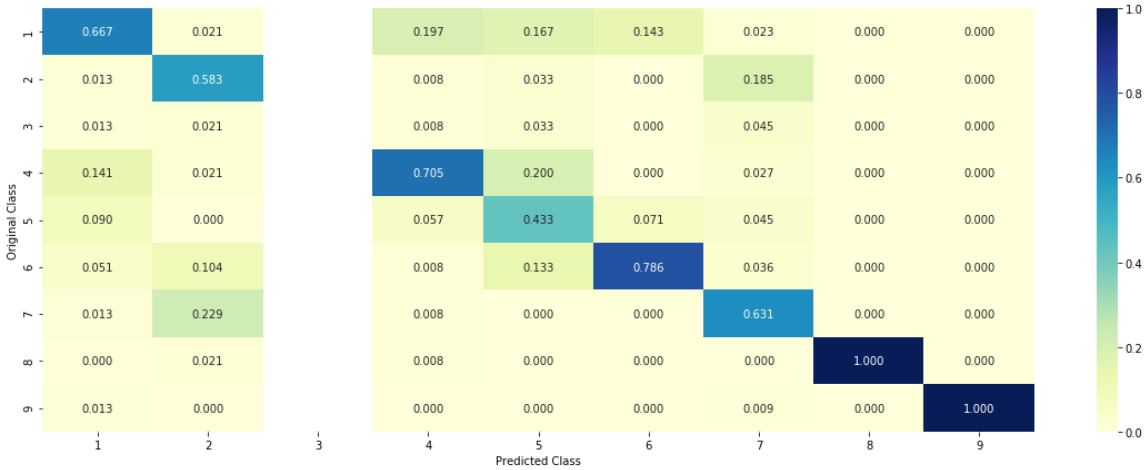
In [128]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
ng, cv_y, clf)
```

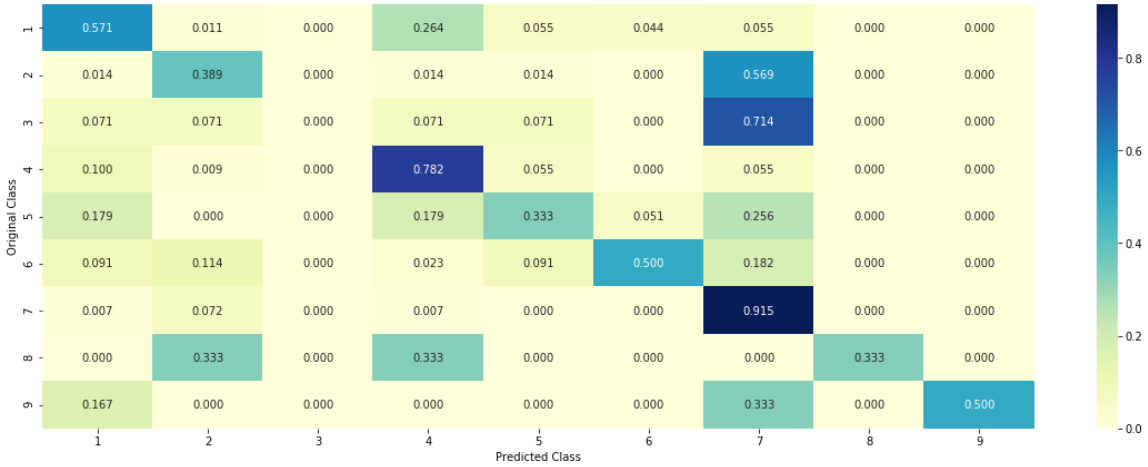
```
Log loss : 0.9940981694004949
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ----------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

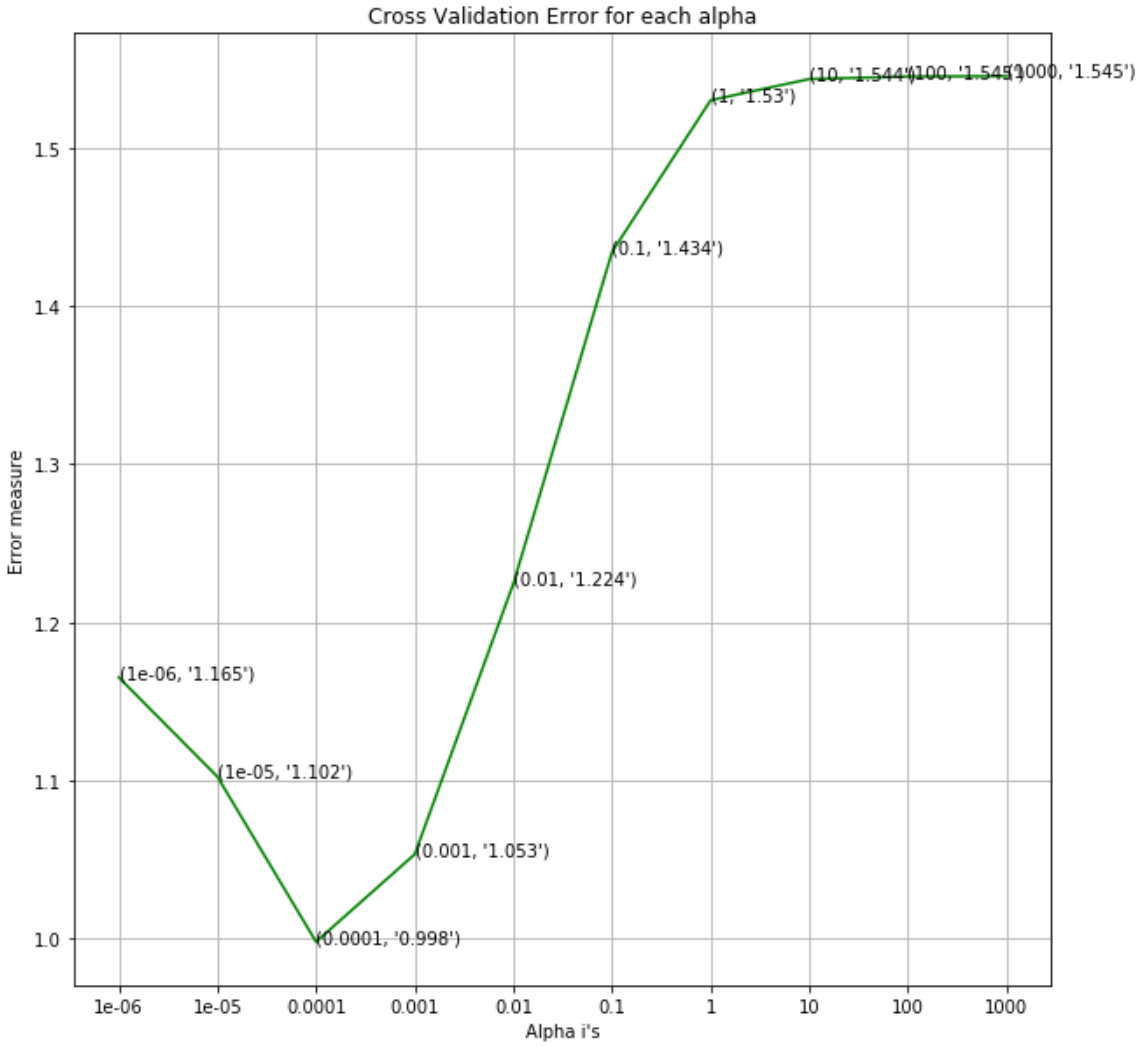


Logistic Regression Without class balancing

In [101]:

```
alpha = [10 ** x for x in range(-6,4)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1650531322923219
for alpha = 1e-05
Log Loss : 1.1021863779702645
for alpha = 0.0001
Log Loss : 0.9977614116682563
for alpha = 0.001
Log Loss : 1.053130814318036
for alpha = 0.01
Log Loss : 1.2243740310201088
for alpha = 0.1
Log Loss : 1.4341992114845128
for alpha = 1
Log Loss : 1.530159622131416
for alpha = 10
Log Loss : 1.5435411187342527
for alpha = 100
Log Loss : 1.5451113025329444
for alpha = 1000
Log Loss : 1.5453568385814278
```



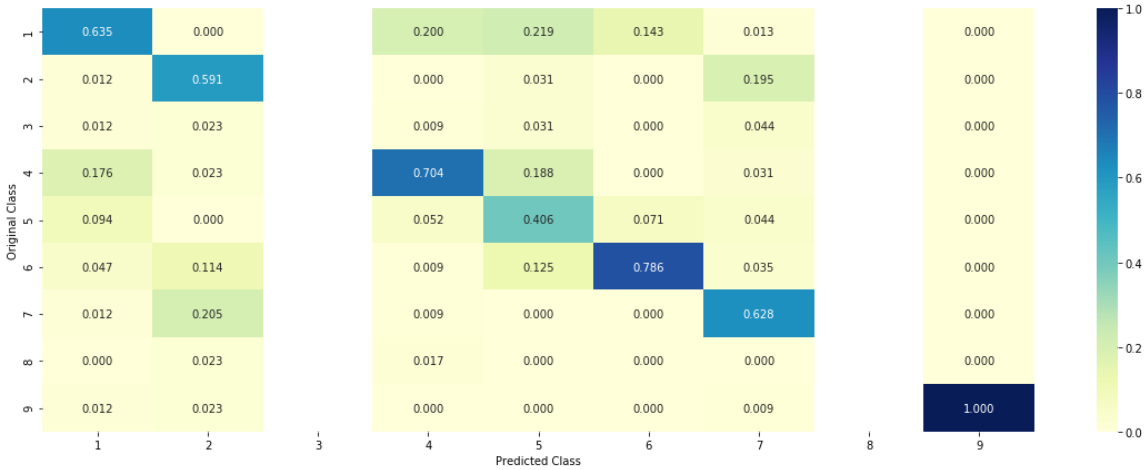

```
For values of best alpha =  0.0001 The train log loss is: 0.45131363
16950974
For values of best alpha =  0.0001 The cross validation log loss is:
0.9977614116682563
For values of best alpha =  0.0001 The test log loss is: 1.013877313
9730666
```

In [102]:

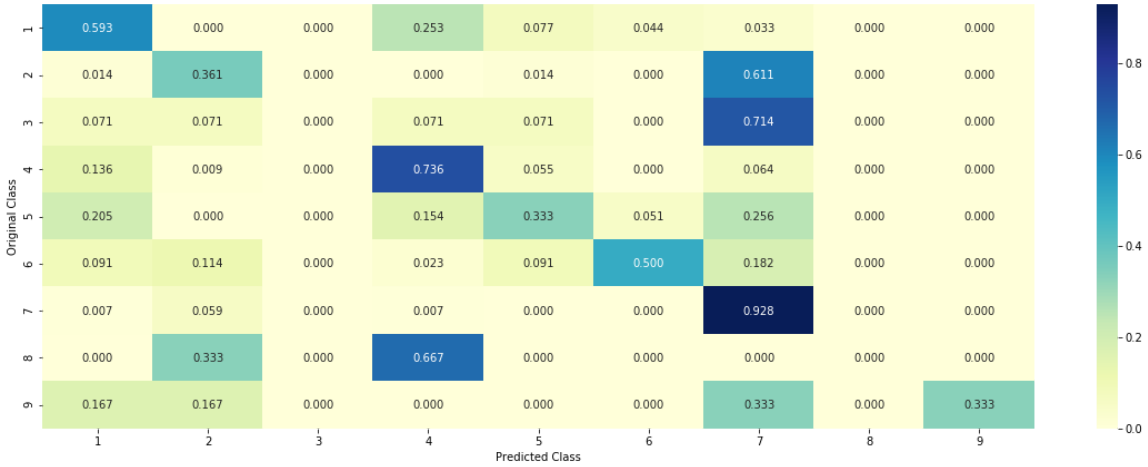
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
ate=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
ng, cv_y, clf)
```

```
Log loss : 0.9977614116682563
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) ----------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

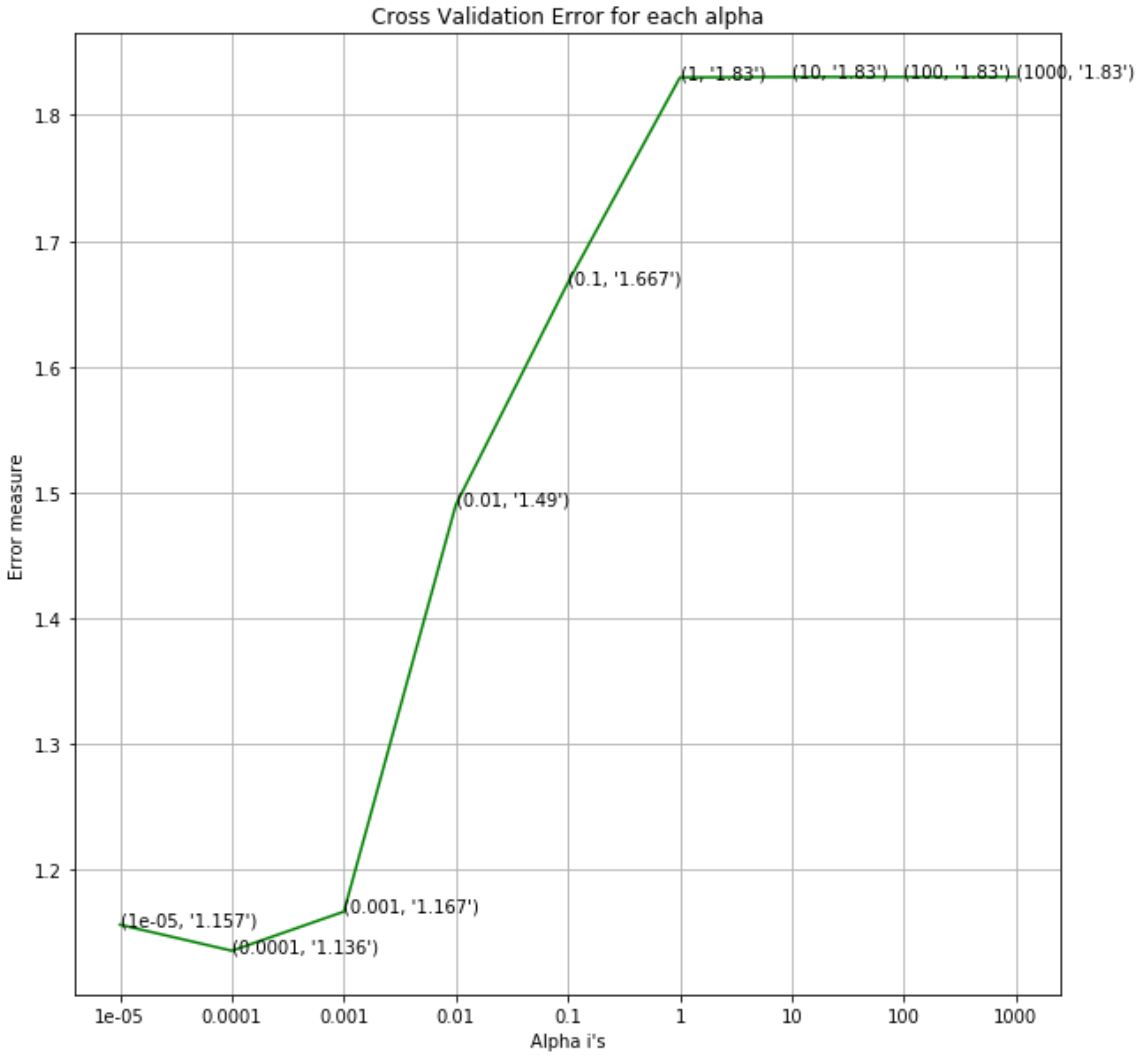


Linear SVM

In [85]:

```
#Linear SVM
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l1', loss='h
inge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.1565396717470717
for C = 0.0001
Log Loss : 1.1358441694065033
for C = 0.001
Log Loss : 1.1669557357935534
for C = 0.01
Log Loss : 1.4904444702695658
for C = 0.1
Log Loss : 1.6668320646864898
for C = 1
Log Loss : 1.8298591041766026
for C = 10
Log Loss : 1.8302599806220925
for C = 100
Log Loss : 1.8302599806220157
for C = 1000
Log Loss : 1.8302599806220092
```

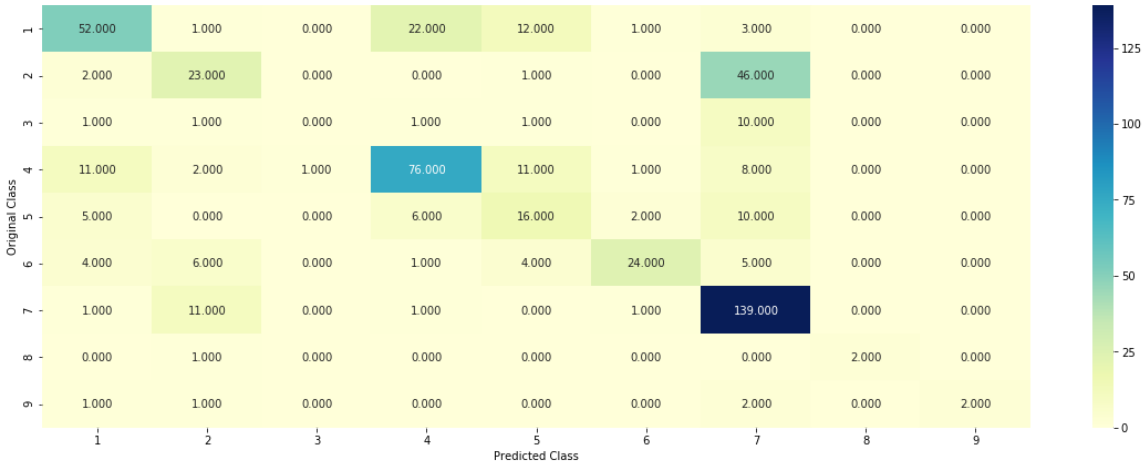



```
For values of best alpha =  0.0001 The train log loss is: 0.93982122
75786568
For values of best alpha =  0.0001 The cross validation log loss is:
1.179765131252822
For values of best alpha =  0.0001 The test log loss is: 1.178663483
103197
```

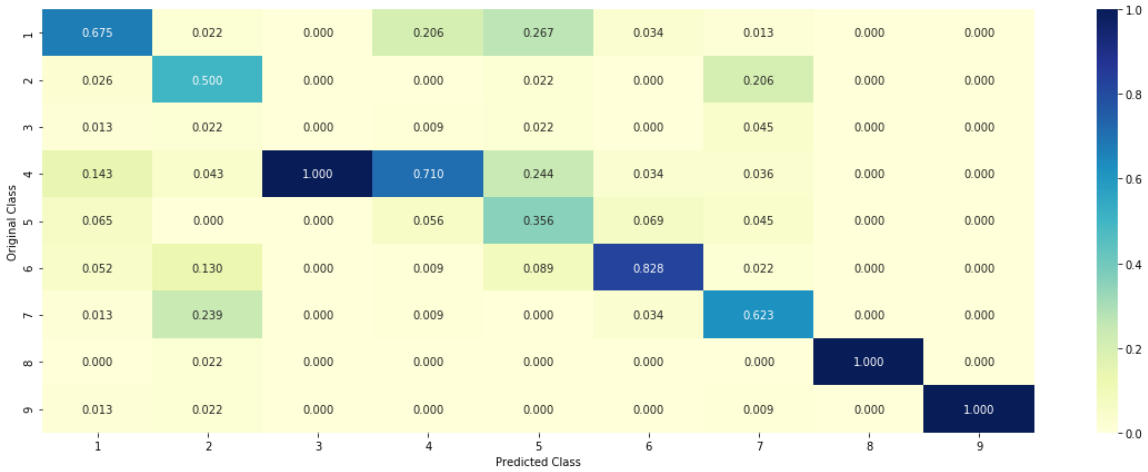
In [105]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_
state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCodin
g,cv_y, clf)
```

```
Log loss : 1.0959906185286656
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

RandomForest

In [51]:

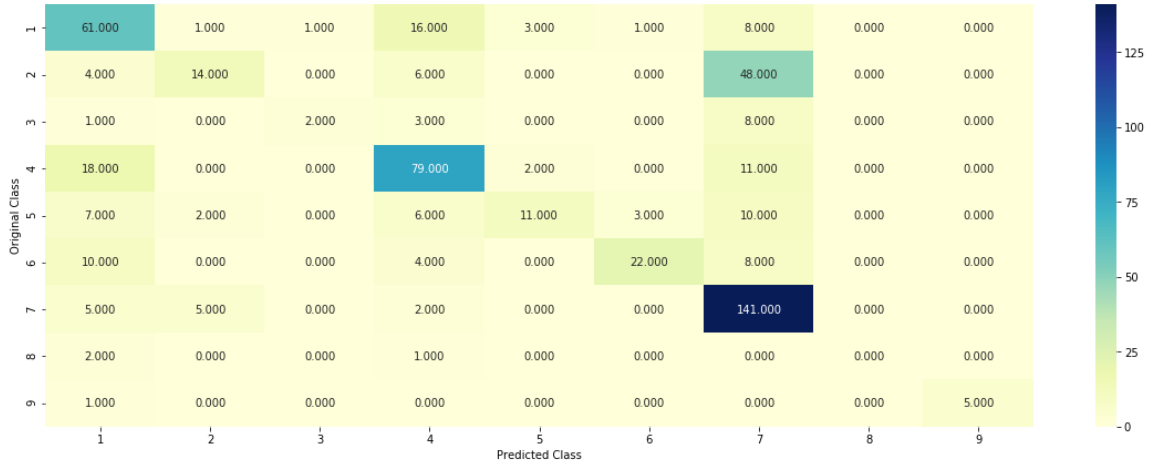
```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10,50,100,500]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth
=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='g
ini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The train lo
g loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The cross va
lidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The test log
 loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1254478894658353
for n_estimators = 100 and max depth =  10
Log Loss : 1.0614793155357773
for n_estimators = 100 and max depth =  50
Log Loss : 1.08163105002893
for n_estimators = 100 and max depth =  100
Log Loss : 1.08163105002893
for n_estimators = 100 and max depth =  500
Log Loss : 1.08163105002893
for n_estimators = 200 and max depth =  5
Log Loss : 1.1155954733788476
for n_estimators = 200 and max depth =  10
Log Loss : 1.0539036096156442
for n_estimators = 200 and max depth =  50
Log Loss : 1.0736584270645235
for n_estimators = 200 and max depth =  100
Log Loss : 1.0736584270645235
for n_estimators = 200 and max depth =  500
Log Loss : 1.0736584270645235
for n_estimators = 500 and max depth =  5
Log Loss : 1.113462260769318
for n_estimators = 500 and max depth =  10
Log Loss : 1.0480423531741883
for n_estimators = 500 and max depth =  50
Log Loss : 1.0722937426723371
for n_estimators = 500 and max depth =  100
Log Loss : 1.0722937426723371
for n_estimators = 500 and max depth =  500
Log Loss : 1.0722937426723371
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1066979604837668
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0495239546975959
for n_estimators = 1000 and max depth =  50
Log Loss : 1.0708427799268172
for n_estimators = 1000 and max depth =  100
Log Loss : 1.0708427799268172
for n_estimators = 1000 and max depth =  500
Log Loss : 1.0708427799268172
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1054715424924306
for n_estimators = 2000 and max depth =  10
Log Loss : 1.0483719996441088
for n_estimators = 2000 and max depth =  50
Log Loss : 1.0708214826763363
for n_estimators = 2000 and max depth =  100
Log Loss : 1.0708214826763363
for n_estimators = 2000 and max depth =  500
Log Loss : 1.0708214826763363
For values of best estimator =  500 The train log loss is: 0.5506304
144916915
For values of best estimator =  500 The cross validation log loss i
s: 1.0480423531741883
For values of best estimator =  500 The test log loss is: 1.08703300
5459812
```

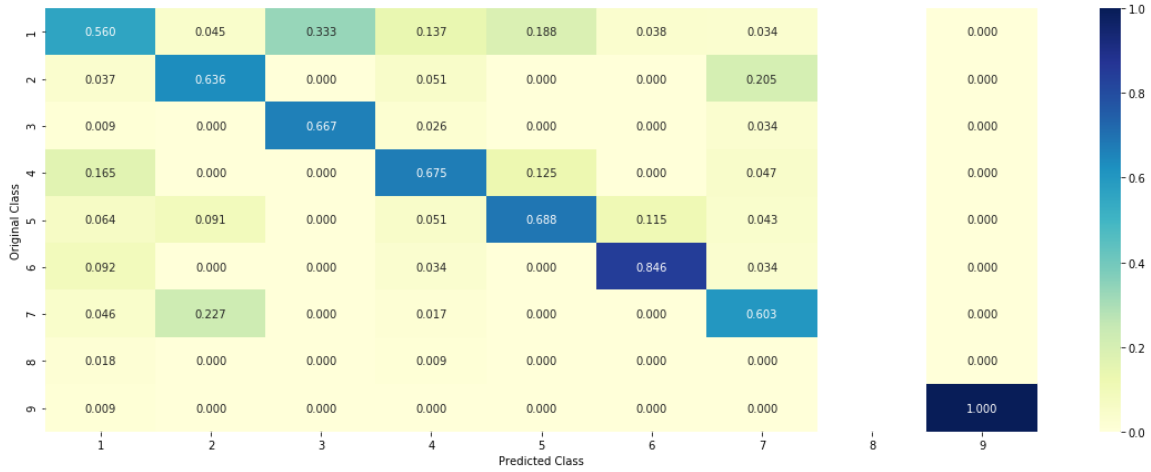
In [105]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='gini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1076598371075794
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

A3

Logistic Regression with unigram and bigram

In [49]:

```
count_vect = CountVectorizer(ngram_range=(1,2),max_features=2000)
train_text_feature_onehotCoding  = count_vect.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = count_vect.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = count_vect.transform(cv_df['TEXT'])
feature_names = count_vect.get_feature_names()
```

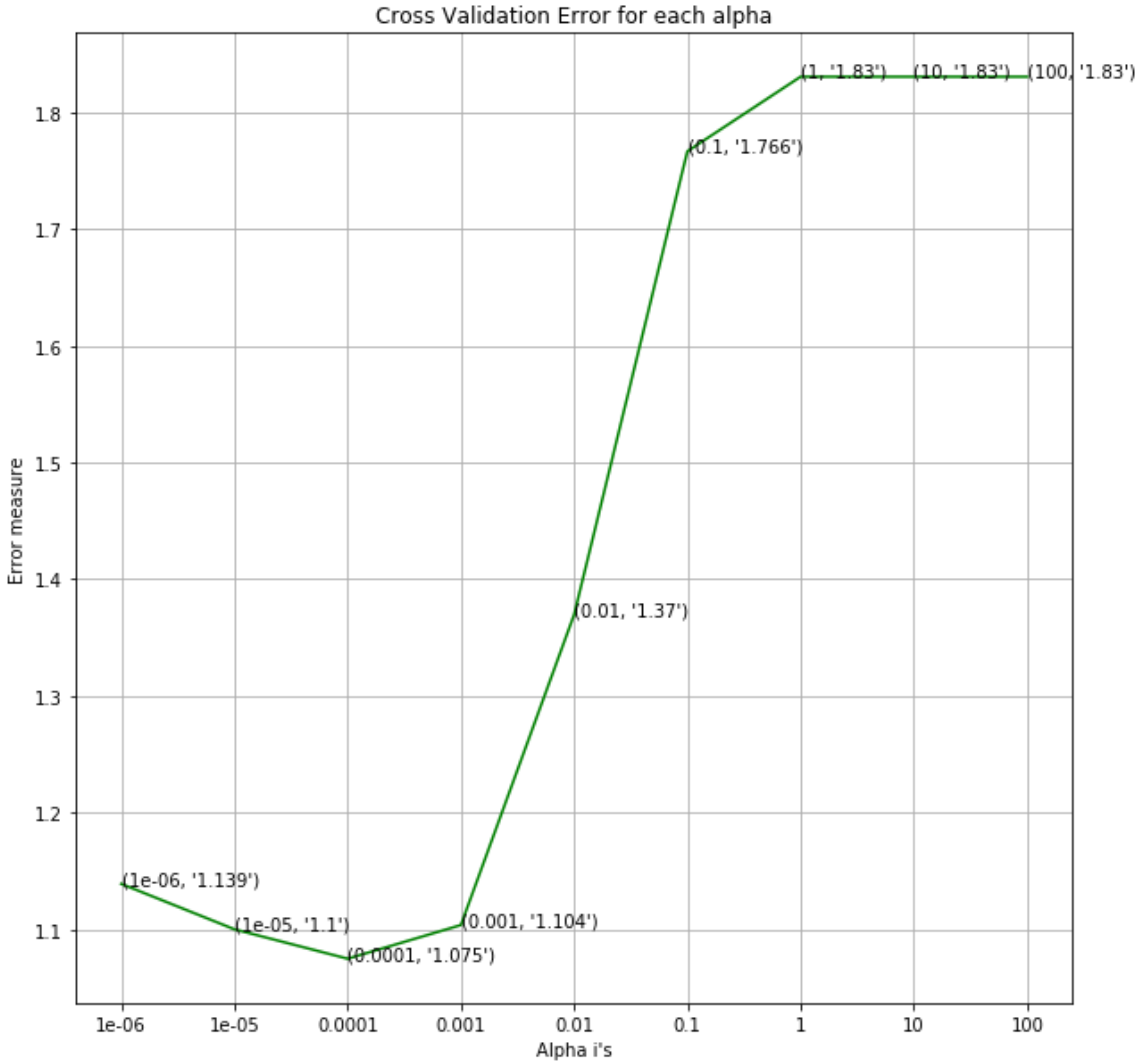
In [50]:

```
#stacking the gene ,variation and text feature together
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_vari
ation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
on_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea
ture_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_o
nehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_oneh
otCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCodi
ng)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [75]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l1', loss='lo
g', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probabil
ity estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
1', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1391935349785767
for alpha = 1e-05
Log Loss : 1.100279561916946
for alpha = 0.0001
Log Loss : 1.075086624696326
for alpha = 0.001
Log Loss : 1.103912783604467
for alpha = 0.01
Log Loss : 1.3697507308869439
for alpha = 0.1
Log Loss : 1.766393055239626
for alpha = 1
Log Loss : 1.8302599806228934
for alpha = 10
Log Loss : 1.8302599806220305
for alpha = 100
Log Loss : 1.8302599806220097
```

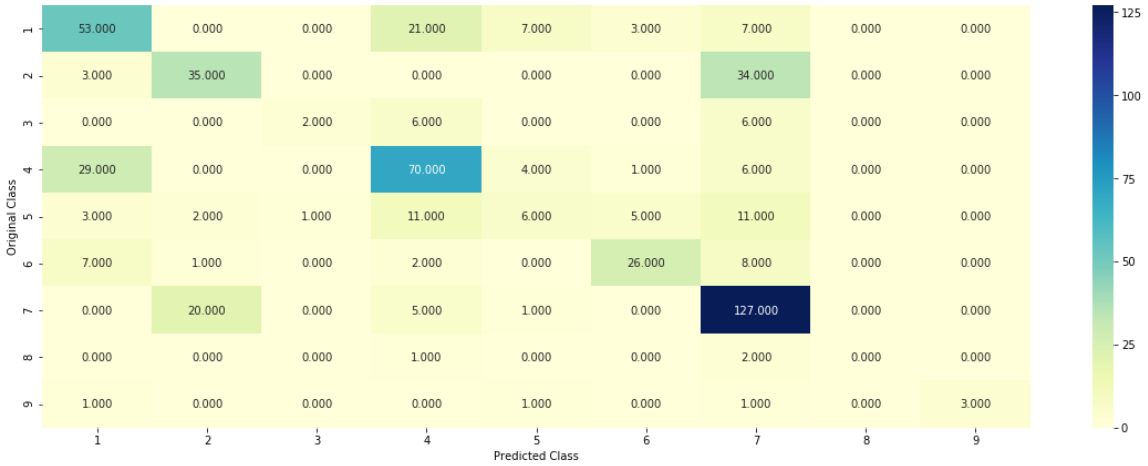



```
For values of best alpha =  0.0001 The train log loss is: 0.81583299
58716024
For values of best alpha =  0.0001 The cross validation log loss is:
1.075086624696326
For values of best alpha =  0.0001 The test log loss is: 1.117455060
9898297
```

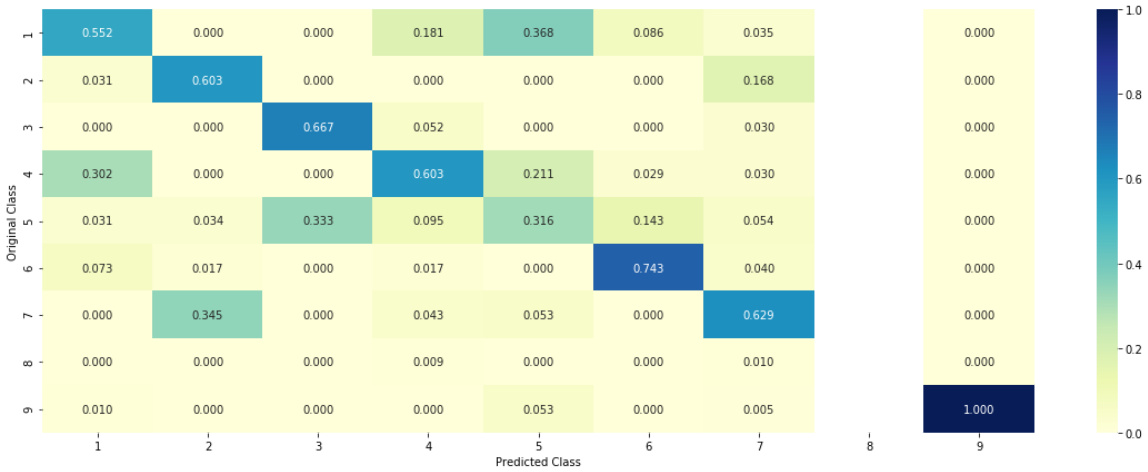
In [77]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
1', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
ng, cv_y, clf)
```

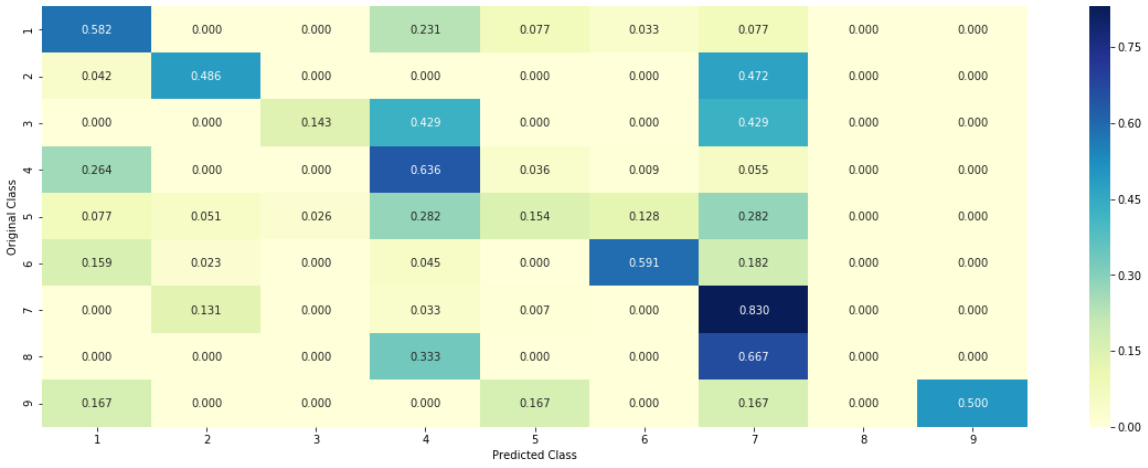
```
Log loss : 1.075086624696326
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



A4

In [53]:

```
def findlen(data):
    length = []
    for x in data:
        text_len = len(x.split())
        length.append(text_len)
    return length
```

In [54]:

```
train_length = findlen(train_df['TEXT'])
test_length = findlen(test_df['TEXT'])
cv_length = findlen(cv_df['TEXT'])
```

In [55]:

```
train_length = [[x] for x in train_length]
test_length =[[x]for x in test_length]
cv_length = [[x]for x in cv_length]
```

In [57]:

```
train_length = np.array(train_length)
test_length = np.array(test_length)
cv_length = np.array(cv_length)
```

In [58]:

```
# Using Google News Word2Vectors
from gensim.models import KeyedVectors
import pickle
from gensim.models import Word2Vec

#model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin',
 binary=True)
```

In [59]:

```
def getListOfSentences(values):
    list_of_sent=[]
    for sent in values:
        list_of_sent.append(sent.split())
    return list_of_sent
```

In [60]:

```
list_of_sent = getListOfSentences(train_df['TEXT'].values)
w2v_model=Word2Vec(list_of_sent,min_count=10,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

In [61]:

```
w2v_model.wv['intragenic'].shape[0]
```

Out[61]:

```
50
```

In [62]:

```
w2v_model.wv.most_similar('intragenic')
```

Out[62]:

```
[('gene', 0.604403555393219),
 ('10q', 0.6022024750709534),
 ('unbalanced', 0.5982071161270142),
 ('pseudogene', 0.5972951054573059),
 ('chromosome', 0.5820643901824951),
 ('3p21', 0.5781393647193909),
 ('chromosomal', 0.573022723197937),
 ('nonsense', 0.5631164908409119),
 ('covering', 0.5492689609527588),
 ('cdkn2a', 0.5478154420852661)]
```

In [63]:

```
def build_avg_vec(sentence, num_features, doc_id, m_name,tf_idf):
    featureVec = np.zeros((num_features,), dtype="float32")
    # we will intialize a vector of size 300 with all zeros
    # we add each word2vec(wordi) to this fetureVec
    nwords = 0

    for word in sentence.split():
        nwords += 1
        if word in w2v_model.wv.vocab:
            if m_name == 'weighted' and word in  tf_idf_vect.vocabulary_:
                featureVec = np.add(featureVec, tf_idf[doc_id, tf_idf_vect.vocab
ulary_[word]] * w2v_model.wv[word])
            elif m_name == 'avg':
                featureVec = np.add(featureVec, w2v_model.wv[word])
    if(nwords>0):
        featureVec = np.divide(featureVec, nwords)
    return featureVec
```

In [64]:

```
def build_avg_vec_for_data(data,tf_idf):
    doc_id = 0
    w2v_title = []
    # for every title we build a avg vector representation
    for i in data['TEXT']:
        w2v_title.append(build_avg_vec(i,50, doc_id,'avg',tf_idf))
        doc_id += 1
    data_frame = pd.DataFrame(w2v_title)
    return data_frame
```

In [65]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),max_features=2000)
train_text_feature_onehotCoding  = tf_idf_vect.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = tf_idf_vect.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = tf_idf_vect.transform(cv_df['TEXT'])
feature_names = tf_idf_vect.get_feature_names()
```

In [66]:

```
w2v_train = build_avg_vec_for_data(train_df,train_text_feature_onehotCoding)
w2v_test = build_avg_vec_for_data(test_df,test_text_feature_onehotCoding)
w2v_cv = build_avg_vec_for_data(cv_df,cv_text_feature_onehotCoding)
```

In [67]:

```
w2v_train.shape[1]
```

Out[67]:

```
50
```

In [68]:

```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_vari
ation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
on_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea
ture_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, w2v_train)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, w2v_test)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, w2v_cv)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [69]:

```
train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,tra
in_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_
variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variat
ion_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, w2v_train))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, w2v_test))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, w2v_cv))
```

Logistic Regression

In [70]:

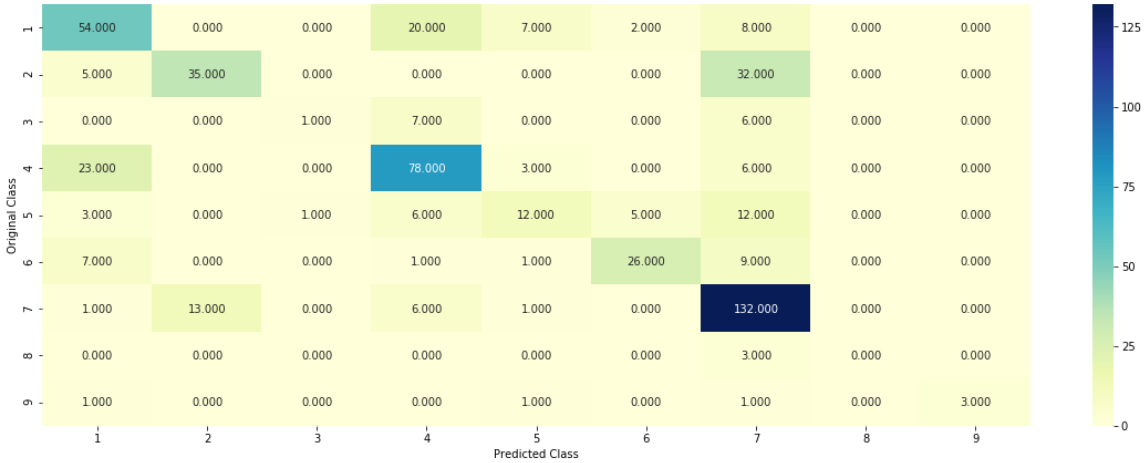
```
alpha = [10 ** x for x in range(-6,4)]
reg =['l1','l2']
cv_log_error_array = []
for i in alpha:
    for r in reg:
        print("for alpha = {} and penalty = {}".format(i,r))
        clf = SGDClassifier(alpha=i, penalty=r, loss='log', random_state=42)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        # to avoid rounding error while multiplying probabilites we use log-prob
ability estimates
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))


best_value = np.argmin(cv_log_error_array)
best_alpha = alpha[int(best_value/2)]
best_reg = reg[int(best_value%2)]
clf = SGDClassifier(alpha=best_alpha, penalty=best_reg, loss='log', random_state
=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ',best_alpha,"penalty =" ,best_reg,"The train
 log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ',best_alpha,"penalty =" ,best_reg,"The cross
 validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-
15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ',best_alpha,"penalty =" ,best_reg,"The test l
og loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06 and penalty = l1
Log Loss : 1.1444637263983286
for alpha = 1e-06 and penalty = l2
Log Loss : 1.1283872216750057
for alpha = 1e-05 and penalty = l1
Log Loss : 1.1093891788226289
for alpha = 1e-05 and penalty = l2
Log Loss : 1.1236236044054784
for alpha = 0.0001 and penalty = l1
Log Loss : 1.0459883713762101
for alpha = 0.0001 and penalty = l2
Log Loss : 1.1127112731763091
for alpha = 0.001 and penalty = l1
Log Loss : 1.0899004306865698
for alpha = 0.001 and penalty = l2
Log Loss : 1.0878108603316294
for alpha = 0.01 and penalty = l1
Log Loss : 1.3631459088062068
for alpha = 0.01 and penalty = l2
Log Loss : 1.2184496995701404
for alpha = 0.1 and penalty = l1
Log Loss : 1.7661455110772168
for alpha = 0.1 and penalty = l2
Log Loss : 1.3773375508226346
for alpha = 1 and penalty = l1
Log Loss : 1.8302599806230972
for alpha = 1 and penalty = l2
Log Loss : 1.5538903825443031
for alpha = 10 and penalty = l1
Log Loss : 1.8302599806220348
for alpha = 10 and penalty = l2
Log Loss : 1.6861548579428323
for alpha = 100 and penalty = l1
Log Loss : 1.8302599806220097
for alpha = 100 and penalty = l2
Log Loss : 1.7094395354246013
for alpha = 1000 and penalty = l1
Log Loss : 1.8302599806220081
for alpha = 1000 and penalty = l2
Log Loss : 1.7119769797256523
For values of best alpha =  0.0001 penalty = l1 The train log loss i
s: 0.788262814337408
For values of best alpha =  0.0001 penalty = l1 The cross validation
log loss is: 1.0459883713762101
For values of best alpha =  0.0001 penalty = l1 The test log loss i
s: 1.1146983092459466
```

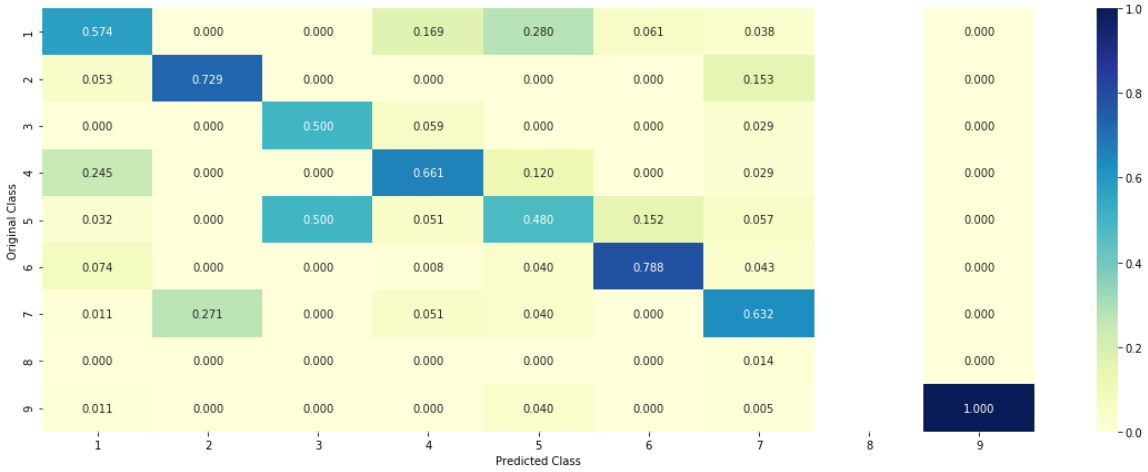
In [73]:

```
clf = SGDClassifier(alpha=best_alpha, penalty='l1', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
ng, cv_y, clf)
```

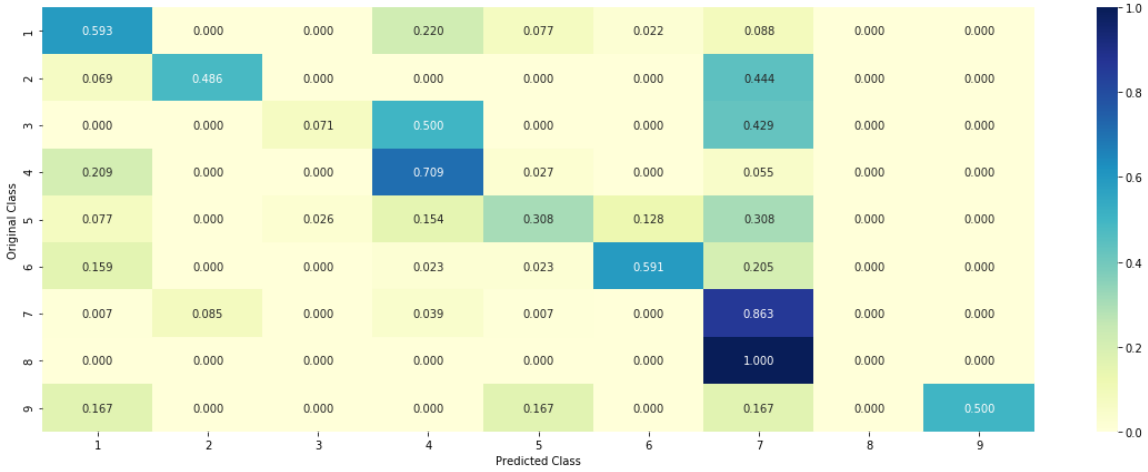
```
Log loss : 1.0459883713762101
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
-----
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

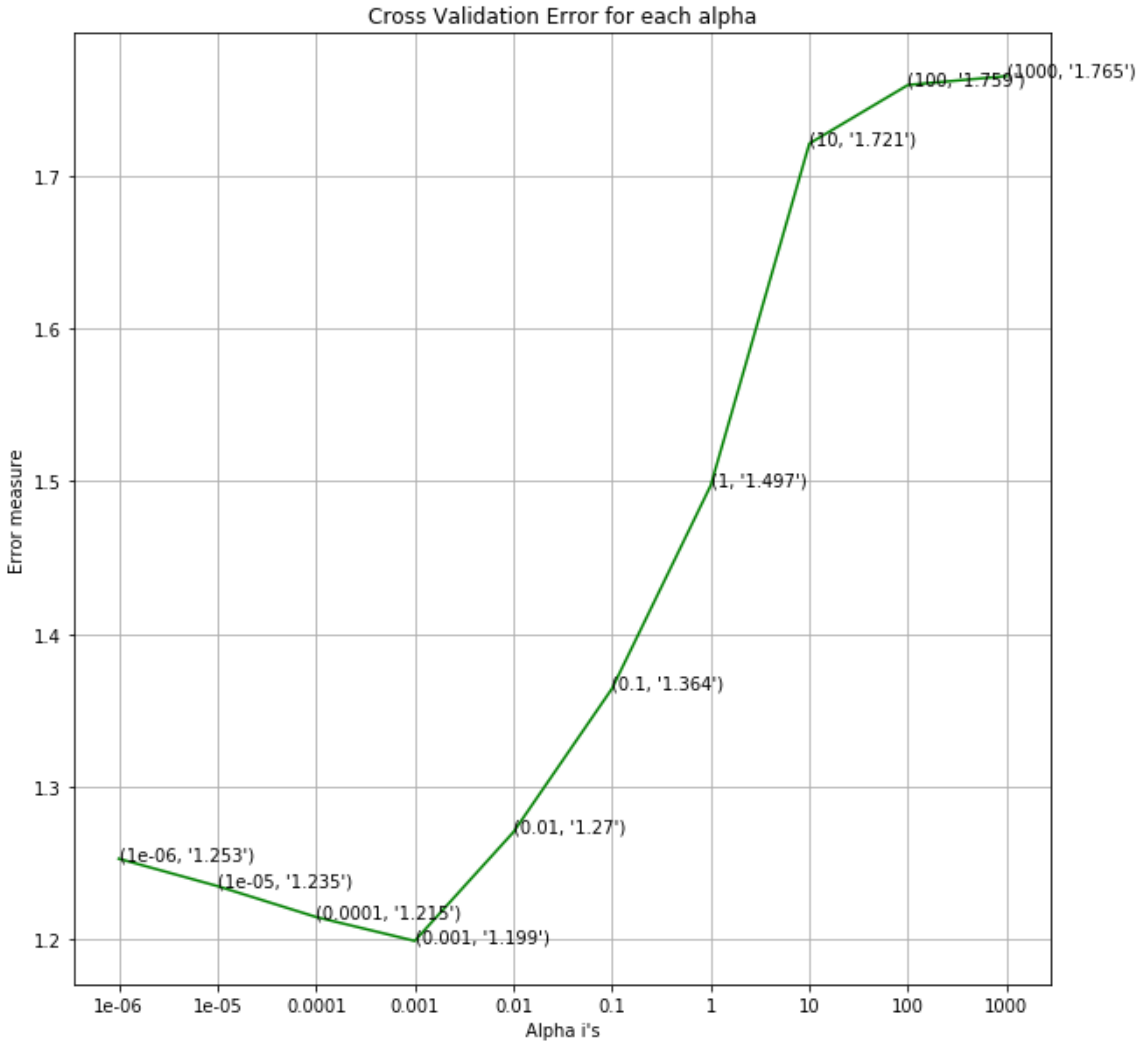


Logistic Regression (response coding and tfidfw2v)

In [87]:

```
alpha = [10 ** x for x in range(-6,4)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='lo
g', random_state=42)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probabil
ity estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.2528006655459203
for alpha = 1e-05
Log Loss : 1.2348925559457973
for alpha = 0.0001
Log Loss : 1.2146603253032617
for alpha = 0.001
Log Loss : 1.1990054783007666
for alpha = 0.01
Log Loss : 1.27019414973551
for alpha = 0.1
Log Loss : 1.3641923032003744
for alpha = 1
Log Loss : 1.4971013351117992
for alpha = 10
Log Loss : 1.7209345141771166
for alpha = 100
Log Loss : 1.7592743112449256
for alpha = 1000
Log Loss : 1.7650763274917618
```

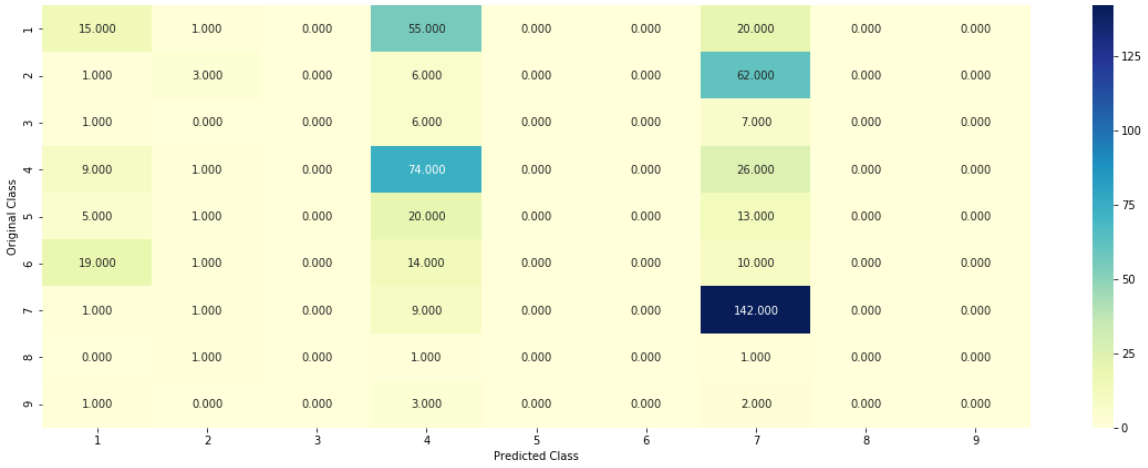



```
For values of best alpha =  0.001 The train log loss is: 1.132498990
3956286
For values of best alpha =  0.001 The cross validation log loss is:
1.1990054783007666
For values of best alpha =  0.001 The test log loss is: 1.1927556021
418486
```

In [88]:

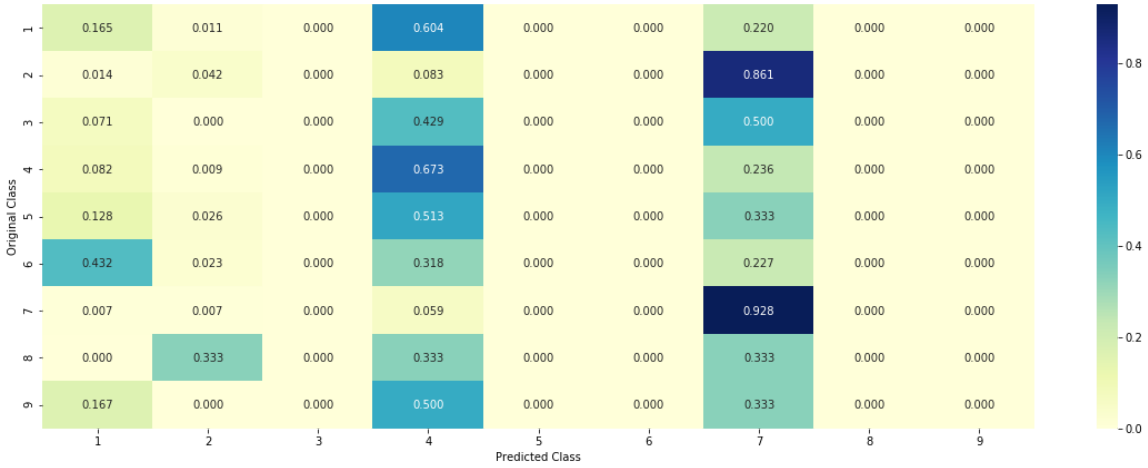
```
clf = SGDClassifier(class_weight='balanced',alpha=best_alpha, penalty='l2', loss
='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_response
Coding, cv_y, clf)
```

```
Log loss : 1.562213693286439
Number of mis-classified points : 0.5601503759398496
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) ----------------
-----
```


```
-------------------- Recall matrix (Row sum=1) --------------------
```



Random Forest

In [90]:

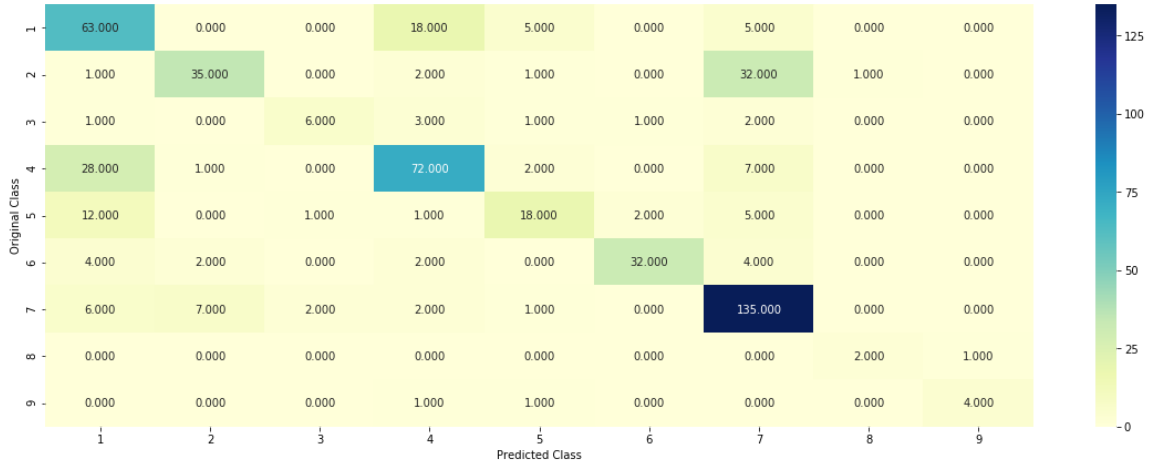
```
alpha = [100,200,500,1000,2000]
max_depth = [5,10,50,100,500]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth
=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='g
ini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The train lo
g loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The cross va
lidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The test log
 loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1791183785298858
for n_estimators = 100 and max depth =  10
Log Loss : 1.0409737790813538
for n_estimators = 100 and max depth =  50
Log Loss : 0.994395997711133
for n_estimators = 100 and max depth =  100
Log Loss : 0.994395997711133
for n_estimators = 100 and max depth =  500
Log Loss : 0.994395997711133
for n_estimators = 200 and max depth =  5
Log Loss : 1.155336604938716
for n_estimators = 200 and max depth =  10
Log Loss : 1.029367624722879
for n_estimators = 200 and max depth =  50
Log Loss : 0.9914313190174783
for n_estimators = 200 and max depth =  100
Log Loss : 0.9914313190174783
for n_estimators = 200 and max depth =  500
Log Loss : 0.9914313190174783
for n_estimators = 500 and max depth =  5
Log Loss : 1.1485170553579194
for n_estimators = 500 and max depth =  10
Log Loss : 1.0211471913282415
for n_estimators = 500 and max depth =  50
Log Loss : 0.9869855744190587
for n_estimators = 500 and max depth =  100
Log Loss : 0.9869855744190587
for n_estimators = 500 and max depth =  500
Log Loss : 0.9869855744190587
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1461455081220315
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0195909626660993
for n_estimators = 1000 and max depth =  50
Log Loss : 0.9877968733287168
for n_estimators = 1000 and max depth =  100
Log Loss : 0.9877810862693077
for n_estimators = 1000 and max depth =  500
Log Loss : 0.9877810862693077
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1463849010291398
for n_estimators = 2000 and max depth =  10
Log Loss : 1.020631892003768
for n_estimators = 2000 and max depth =  50
Log Loss : 0.987629305026247
for n_estimators = 2000 and max depth =  100
Log Loss : 0.9876110311194314
for n_estimators = 2000 and max depth =  500
Log Loss : 0.9876110311194314
For values of best estimator =  500 The train log loss is: 0.4169179
3018393714
For values of best estimator =  500 The cross validation log loss i
s: 0.9869855744190587
For values of best estimator =  500 The test log loss is: 1.04974056
10965704
```

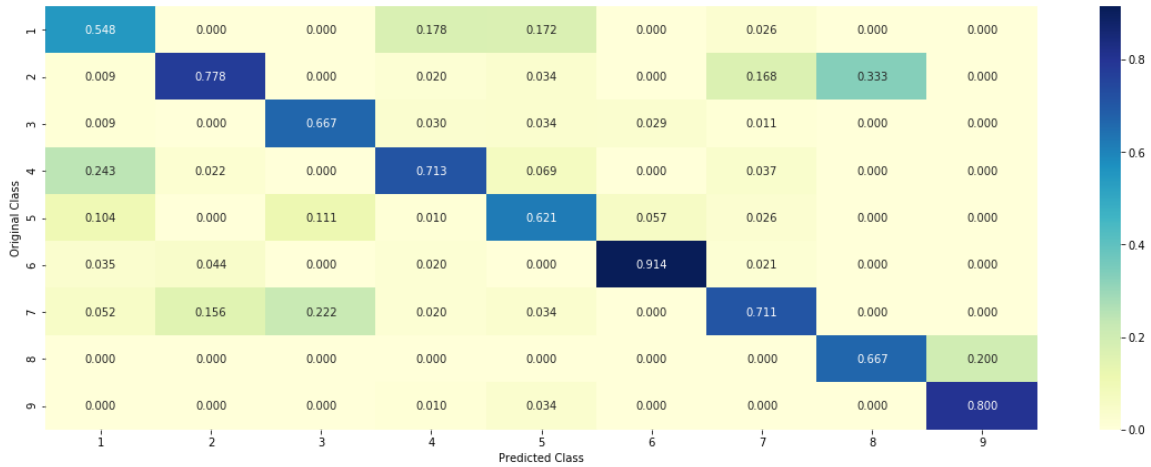
In [91]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='gini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

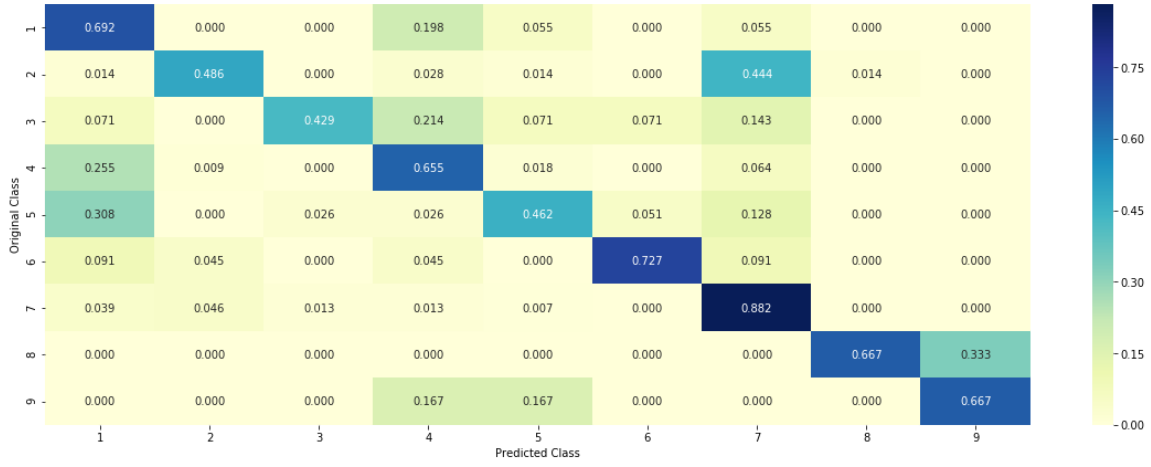
```
Log loss : 0.9869855744190587
Number of mis-classified points : 0.3101503759398496
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Random Forest (response coding)

In [83]:

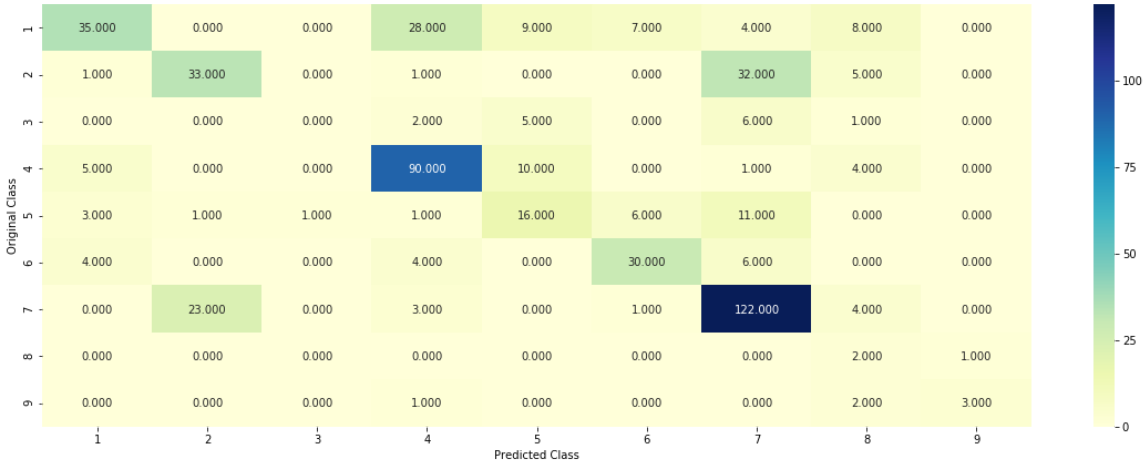
```
alpha = [100,200,500,1000,2000]
max_depth = [5,10,50,100,500]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth
=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='g
ini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The train lo
g loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The cross va
lidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The test log
 loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1606463559891251
for n_estimators = 100 and max depth =  10
Log Loss : 1.1772149723229564
for n_estimators = 100 and max depth =  50
Log Loss : 1.4180985092821543
for n_estimators = 100 and max depth =  100
Log Loss : 1.4180985092821543
for n_estimators = 100 and max depth =  500
Log Loss : 1.4180985092821543
for n_estimators = 200 and max depth =  5
Log Loss : 1.1317833472296108
for n_estimators = 200 and max depth =  10
Log Loss : 1.1759535613153378
for n_estimators = 200 and max depth =  50
Log Loss : 1.3656535219414765
for n_estimators = 200 and max depth =  100
Log Loss : 1.3656535219414765
for n_estimators = 200 and max depth =  500
Log Loss : 1.3656535219414765
for n_estimators = 500 and max depth =  5
Log Loss : 1.1536013299684527
for n_estimators = 500 and max depth =  10
Log Loss : 1.1683076785364523
for n_estimators = 500 and max depth =  50
Log Loss : 1.3167367273768968
for n_estimators = 500 and max depth =  100
Log Loss : 1.3167367273768968
for n_estimators = 500 and max depth =  500
Log Loss : 1.3167367273768968
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1686164975978568
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1708772577909605
for n_estimators = 1000 and max depth =  50
Log Loss : 1.3317244334909024
for n_estimators = 1000 and max depth =  100
Log Loss : 1.3317244334909024
for n_estimators = 1000 and max depth =  500
Log Loss : 1.3317244334909024
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1762765322650437
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1827974167004383
for n_estimators = 2000 and max depth =  50
Log Loss : 1.3401038500607334
for n_estimators = 2000 and max depth =  100
Log Loss : 1.3401038500607334
for n_estimators = 2000 and max depth =  500
Log Loss : 1.3401038500607334
5
For values of best estimator =  200 The train log loss is: 0.0623386
8059945624
For values of best estimator =  200 The cross validation log loss i
s: 1.1317833472296108
For values of best estimator =  200 The test log loss is: 1.17903560
95620158
```

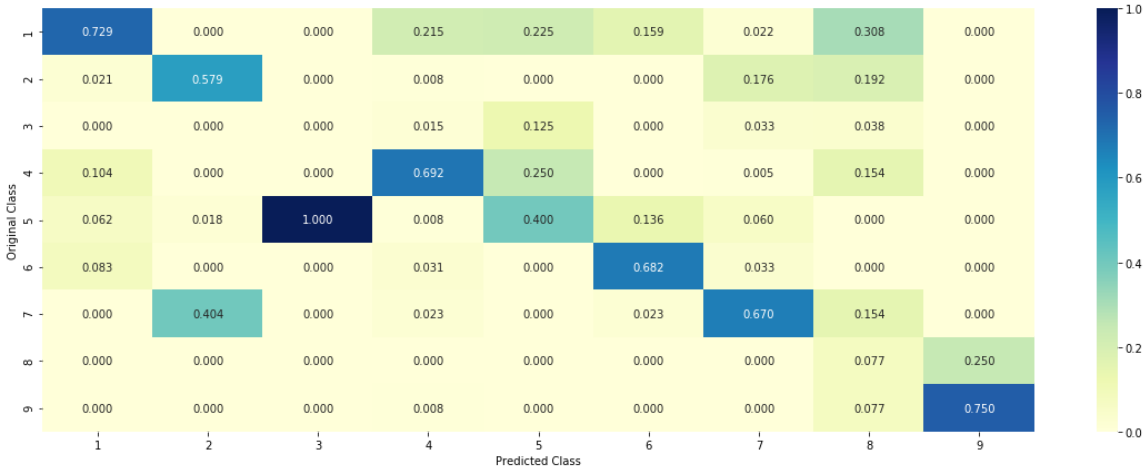
In [86]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='gini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.1317833472296108
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

Logistic Regression with trigrams and fourgrams

In [50]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(2,4),max_features=2000)
train_text_feature_onehotCoding  = tf_idf_vect.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = tf_idf_vect.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = tf_idf_vect.transform(cv_df['TEXT'])
feature_names = tf_idf_vect.get_feature_names()
```

In [51]:

```
train_text_feature_onehotCoding.shape
```

Out[51]:

```
(2124, 2000)
```

In [52]:

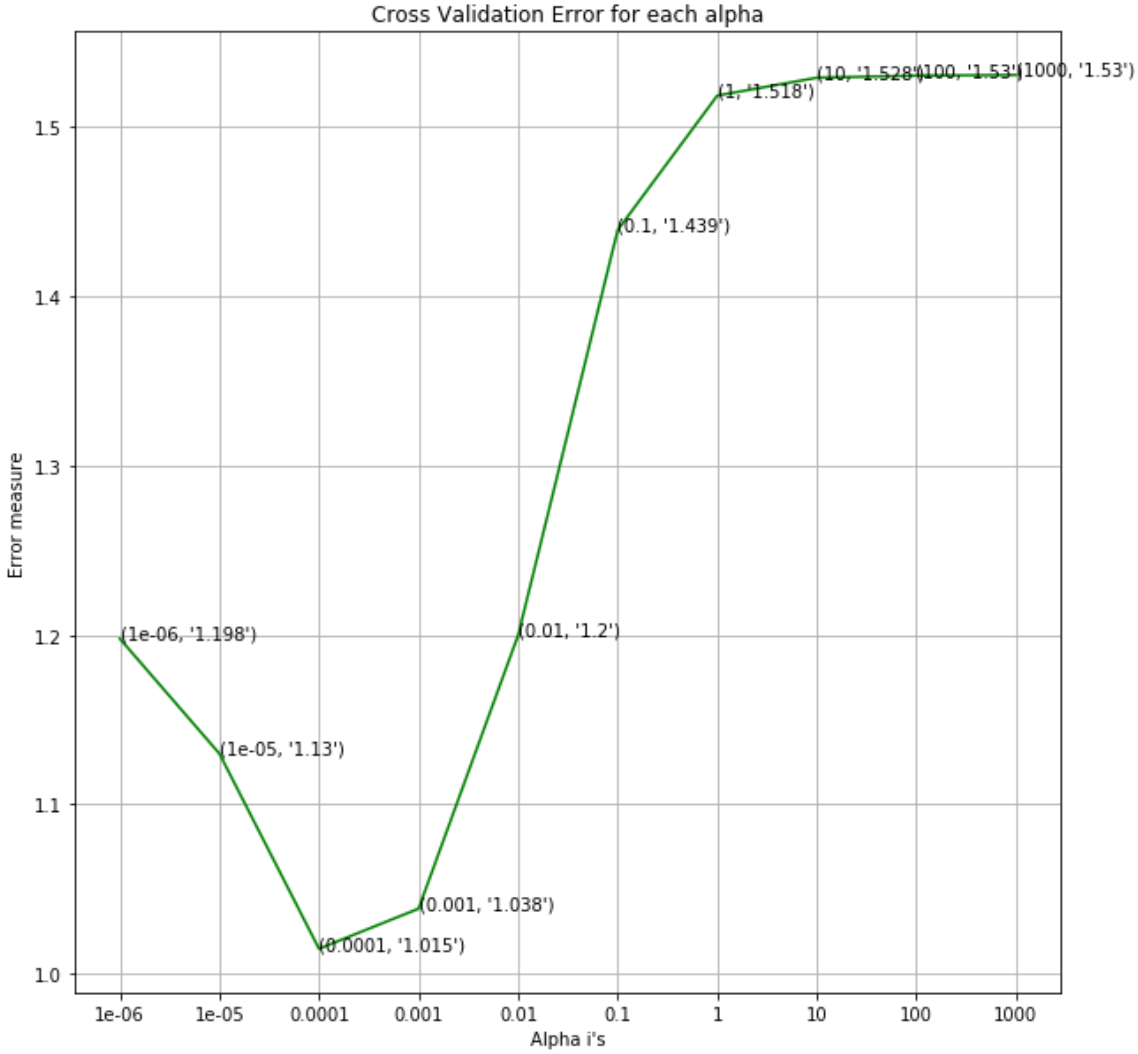
```
#stacking the gene ,variation and text feature together
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_vari
ation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
on_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea
ture_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_o
nehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_oneh
otCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCodi
ng)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

Logistic Regression

In [53]:

```
alpha = [10 ** x for x in range(-6, 4)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='lo
g', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probabil
ity estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.197668110252756
for alpha = 1e-05
Log Loss : 1.1298663462942546
for alpha = 0.0001
Log Loss : 1.014817882211951
for alpha = 0.001
Log Loss : 1.0384547802430049
for alpha = 0.01
Log Loss : 1.1997574569510479
for alpha = 0.1
Log Loss : 1.438693075404033
for alpha = 1
Log Loss : 1.5179707627045
for alpha = 10
Log Loss : 1.5284984830032657
for alpha = 100
Log Loss : 1.5297461552353508
for alpha = 1000
Log Loss : 1.5301420835565767
```

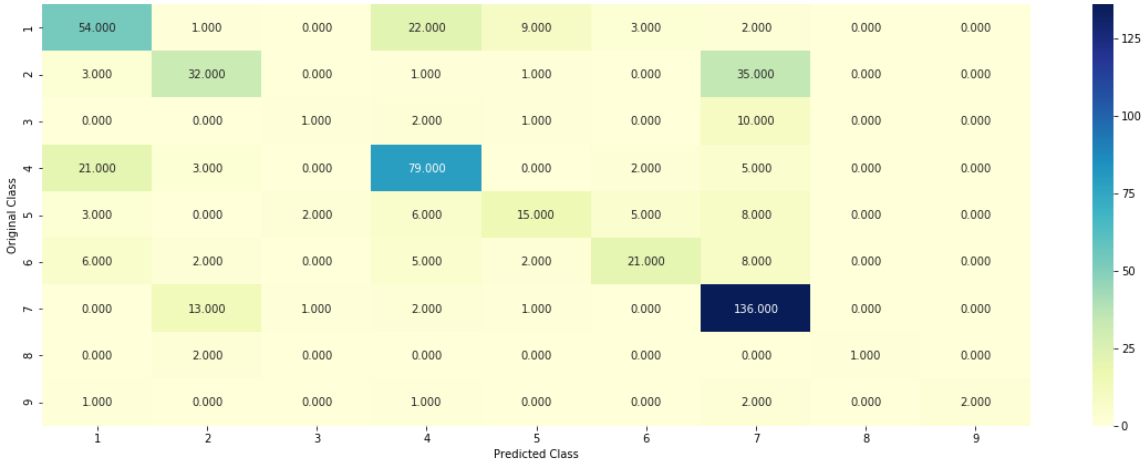


```
For values of best alpha =  0.0001 The train log loss is: 0.41640795
74478961
For values of best alpha =  0.0001 The cross validation log loss is:
1.014817882211951
For values of best alpha =  0.0001 The test log loss is: 0.979574406
3377371
```

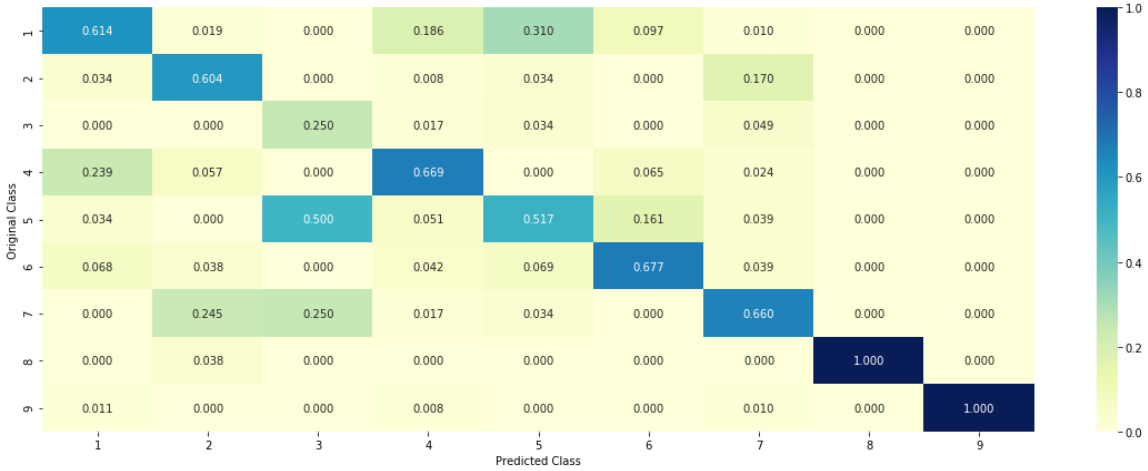
In [54]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
ng, cv_y, clf)
```

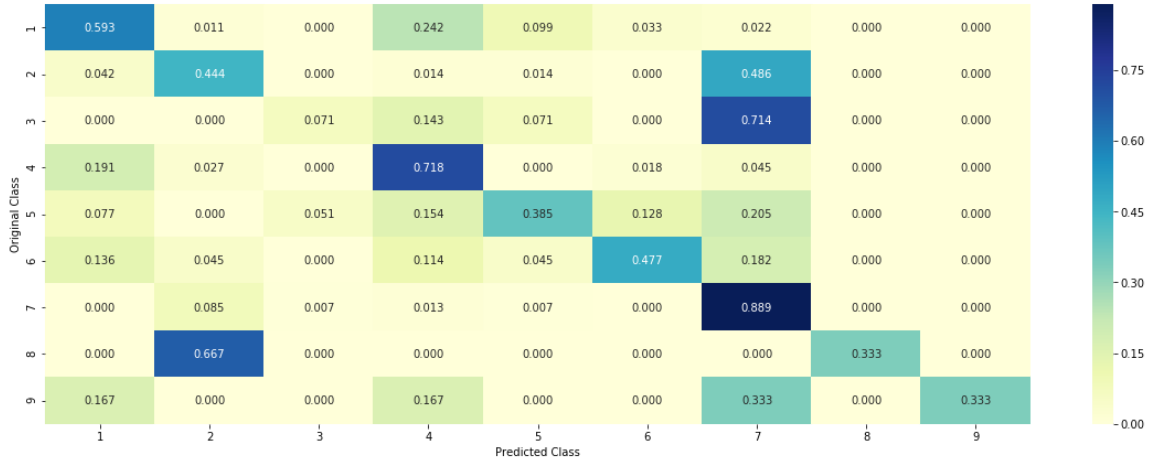
```
Log loss : 1.014817882211951
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

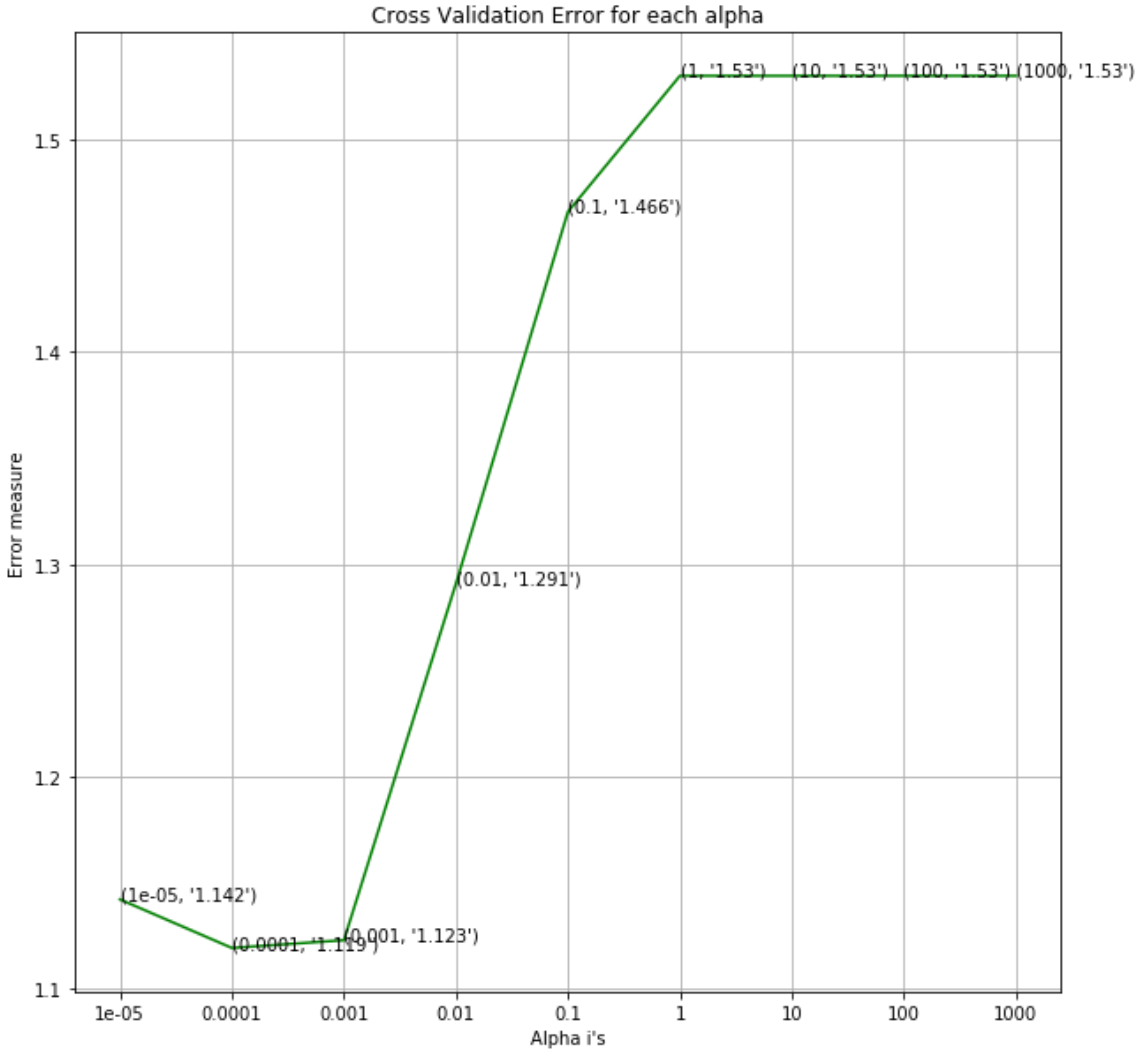

Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
0.593 0.011 0.000 0.242 0.099 0.033 0.022 0.000 0.000
0.042 0.444 0.000 0.014 0.014 0.000 0.486 0.000 0.000
0.000 0.000 0.071 0.143 0.071 0.000 0.714 0.000 0.000
0.191 0.027 0.000 0.718 0.000 0.018 0.045 0.000 0.000
0.077 0.000 0.051 0.154 0.385 0.128 0.205 0.000 0.000
0.136 0.045 0.000 0.114 0.045 0.477 0.182 0.000 0.000
0.000 0.085 0.007 0.013 0.007 0.000 0.889 0.000 0.000
0.000 0.667 0.000 0.000 0.000 0.000 0.000 0.333 0.000
0.167 0.000 0.000 0.167 0.000 0.000 0.333 0.000 0.333
0.75
0.60
0.45
0.30
0.15
0.00

Linear SVM

In [55]:

```
#Linear SVM
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hi
nge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
 eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots(figsize=(10,10))
a = list(range(0,len(alpha)))
ax.plot(a, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (a[i],cv_log_error_array[i]))
plt.grid()
ax.set_xticks(a)
ax.set_xticklabels(alpha)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.1419431840966083
for C = 0.0001
Log Loss : 1.1190491060239733
for C = 0.001
Log Loss : 1.1227091728687857
for C = 0.01
Log Loss : 1.2907171295373083
for C = 0.1
Log Loss : 1.465957649132745
for C = 1
Log Loss : 1.5300613440198698
for C = 10
Log Loss : 1.5300613742097036
for C = 100
Log Loss : 1.5300613479916958
for C = 1000
Log Loss : 1.53006135232054
```

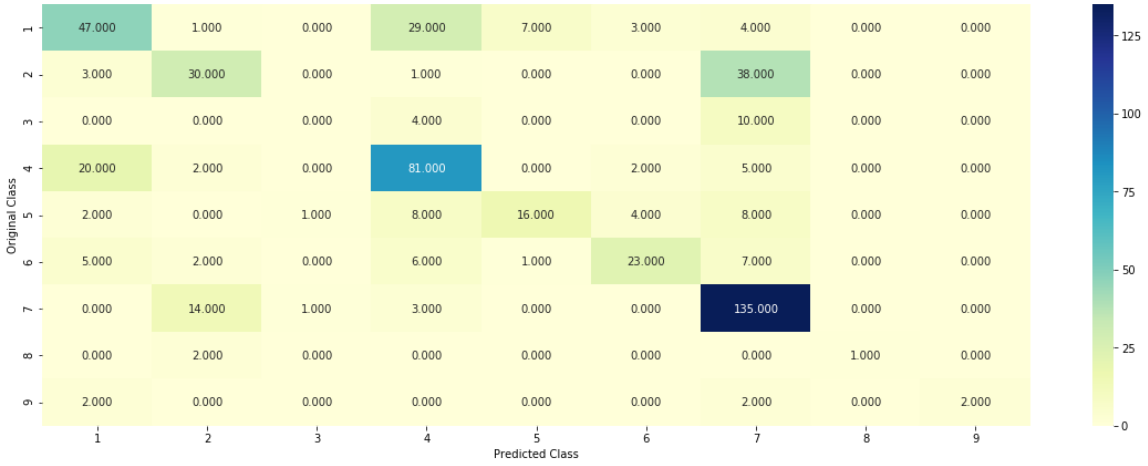

Cross Validation Error for each alpha
Error measure
Alpha i's
1.1
1.2
1.3
1.4
1.5
1e-05
0.0001
0.001
0.01
0.1
1
10
100
1000
(1e-05, '1.142')
(0.0001, '1.119')
(0.001, '1.123')
(0.01, '1.291')
(0.1, '1.466')
(1, '1.53')
(10, '1.53')
(100, '1.53')
(1000, '1.53')

```
For values of best alpha =  0.0001 The train log loss is: 0.47761212
369650036
For values of best alpha =  0.0001 The cross validation log loss is:
1.1190491060239733
For values of best alpha =  0.0001 The test log loss is: 1.069054438
867958
```

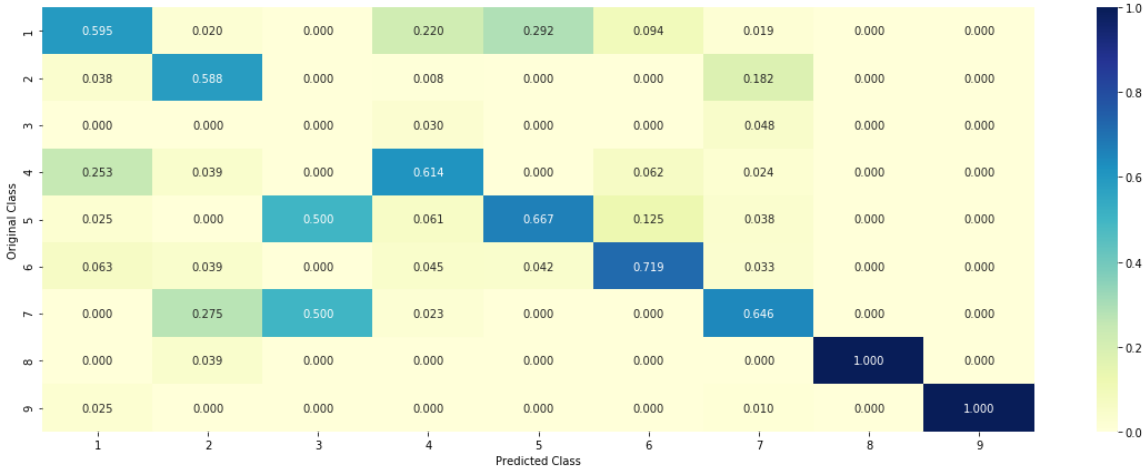
In [56]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
2', loss='hinge', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
ng, cv_y, clf)
```

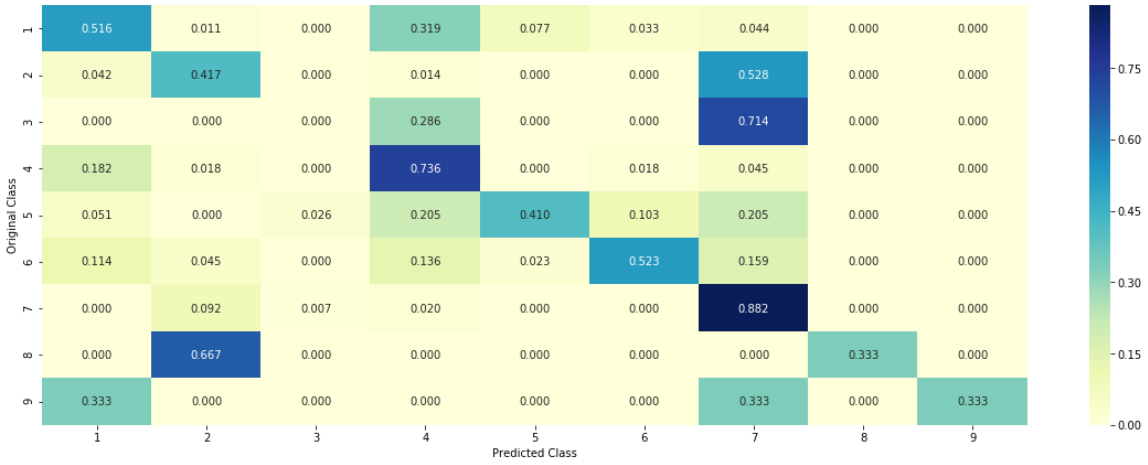
```
Log loss : 1.1190491060239733
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------
------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Random Forest

In [57]:

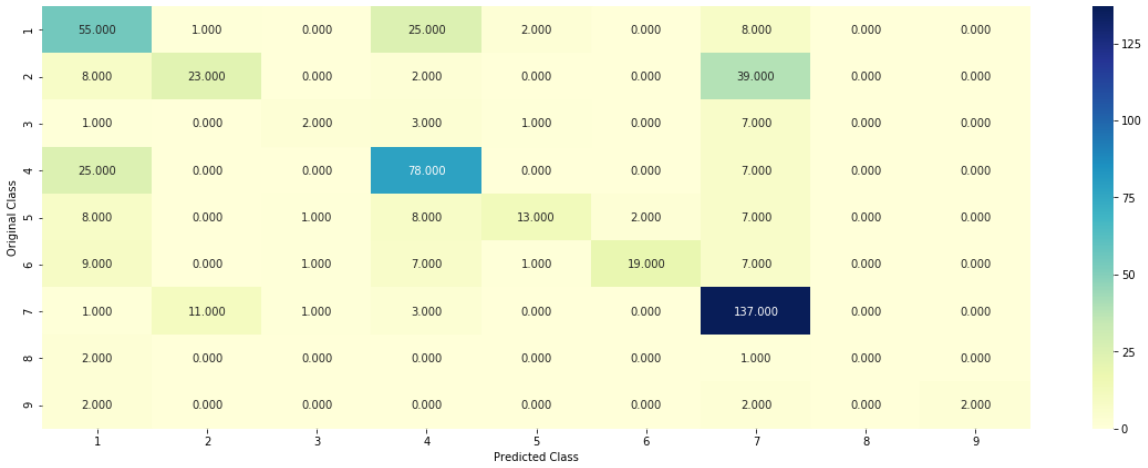
```
alpha = [100,200,500,1000,2000]
max_depth = [5,10,50,100,500]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth
=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.class
es_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='g
ini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The train lo
g loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The cross va
lidation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/5)], "The test log
 loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1824434321360868
for n_estimators = 100 and max depth =  10
Log Loss : 1.098298154006515
for n_estimators = 100 and max depth =  50
Log Loss : 1.1125010333121699
for n_estimators = 100 and max depth =  100
Log Loss : 1.112296095039519
for n_estimators = 100 and max depth =  500
Log Loss : 1.112296095039519
for n_estimators = 200 and max depth =  5
Log Loss : 1.1719330396347234
for n_estimators = 200 and max depth =  10
Log Loss : 1.0959973642275145
for n_estimators = 200 and max depth =  50
Log Loss : 1.1035004352284032
for n_estimators = 200 and max depth =  100
Log Loss : 1.1033863673239215
for n_estimators = 200 and max depth =  500
Log Loss : 1.1033863673239215
for n_estimators = 500 and max depth =  5
Log Loss : 1.1676825682225374
for n_estimators = 500 and max depth =  10
Log Loss : 1.0919361170130 68
for n_estimators = 500 and max depth =  50
Log Loss : 1.1015656059031624
for n_estimators = 500 and max depth =  100
Log Loss : 1.1016458849893693
for n_estimators = 500 and max depth =  500
Log Loss : 1.1016458849893693
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1601697729322935
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0887469349706085
for n_estimators = 1000 and max depth =  50
Log Loss : 1.1007512187785498
for n_estimators = 1000 and max depth =  100
Log Loss : 1.1004662480898217
for n_estimators = 1000 and max depth =  500
Log Loss : 1.1004662480898217
for n_estimators = 2000 and max depth =  5
Log Loss : 1.158479016184022
for n_estimators = 2000 and max depth =  10
Log Loss : 1.0905092449003733
for n_estimators = 2000 and max depth =  50
Log Loss : 1.100300632417647
for n_estimators = 2000 and max depth =  100
Log Loss : 1.1000247492408652
for n_estimators = 2000 and max depth =  500
Log Loss : 1.1000247492408652
For values of best estimator =  1000 The train log loss is: 0.673192
247406603
For values of best estimator =  1000 The cross validation log loss i
s: 1.088746934970608
For values of best estimator =  1000 The test log loss is: 1.1154499
578631036
```

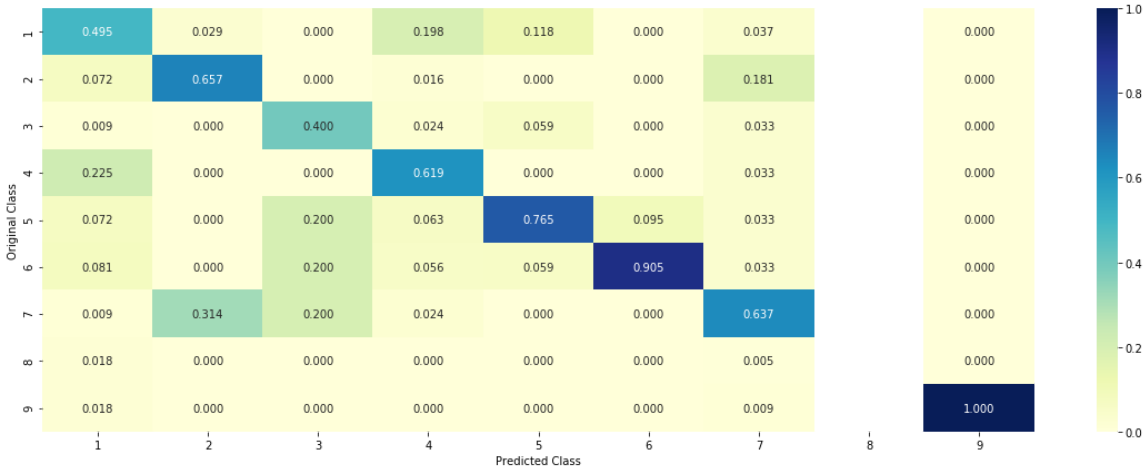
In [59]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/5)], criterion='gini', max_depth=max_depth[int(best_alpha%5)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

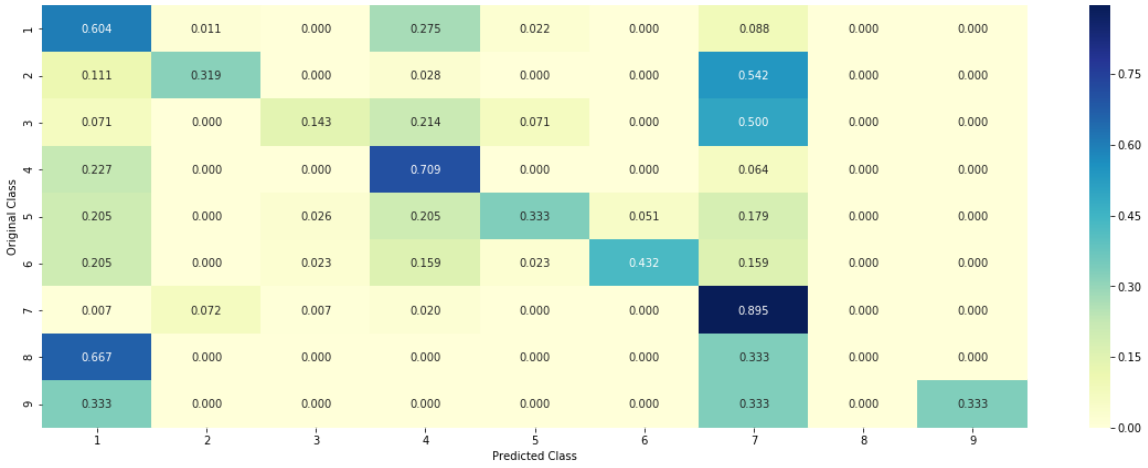
```
Log loss : 1.088746934970608
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Summary

A1 and A2

Algorithm	Hyperparamter	Log Loss
Naive Bayes	alpha =0.001	1.23
Logistic Regression	alpha = 0.0001	1.011
Logistic Regression (without class balancing)	alpha = 0.0001	1.013
Linear SVM	alpha =0.0001	1.178
Random Forest	n_estimators =500 max_depth =10	1.087

A3

Algorithm	Hyperparamter	Log Loss
Logistic Regression	alpha =0.001	1.34

A4

Algorithm	Vectorizer	Hyperparamter	Log Loss
Logistic Regression	One hot coding and AvgTfidfw2v(text)	alpha =0.0001 penalty = l1	1.14
Logistic Regression	Response coding and AvgTfidfw2v(text)	alpha = 0.001	1.18
Random Forest	One hot coding and AvgTfidfw2v(text)	n_estimators =500 max_depth = 10	1.04
Random Forest	Response coding and AvgTfidfw2v(text)	n_estimators = 200 max_depth = 5	1.13
Logistic Regression	Trigrams and Fourgrams(text)	alpha = 0.0001	0.97
Linear SVM	Trigrams and Fourgrams(text)	alpha = 0.0001	1.06
Random Forest	Trigrams and Fourgrams(text)	n_estimators =1000 max_depth =10	1.15

In [ ]: