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.model_selection 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 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('C:\\Users\\admin\\Downloads\\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("C:\\Users\\admin\\Downloads\\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...

3.1.3. Preprocessing of text

In [4]:
```
# 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 [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time,
"seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 268.8862924314848 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

	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...

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Varia
tion']
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

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 [10]:
```
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 outpu
t 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 dis
tribution 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

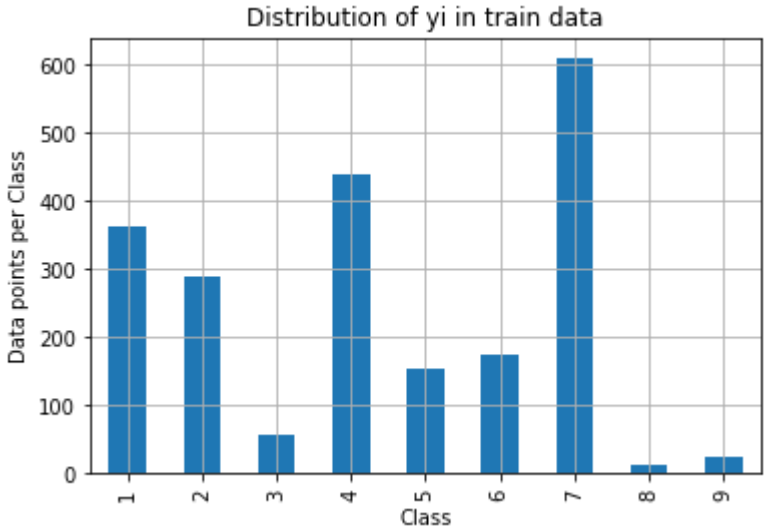
In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

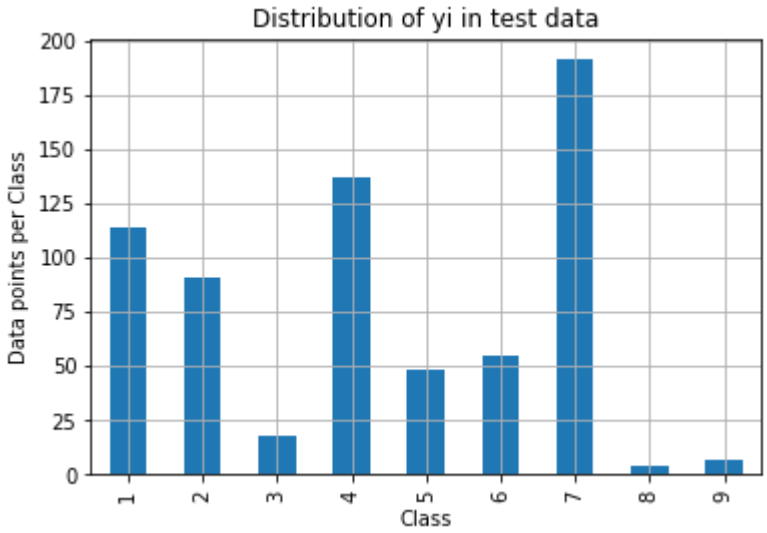
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:
```
# it returns a dict, keys as class labels and values as the number of data
 points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.a
rgsort.html
# -(train_class_distribution.values): the minus sign will give us in decrea
sing 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_distributi
on.values[i], '(', np.round((train_class_distribution.values[i]/train_df.sh
ape[0]*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.a
rgsort.html
# -(train_class_distribution.values): the minus sign will give us in decrea
sing 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_distributio
n.values[i], '(', np.round((test_class_distribution.values[i]/test_df.shape
[0]*100), 3), '%)')

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

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

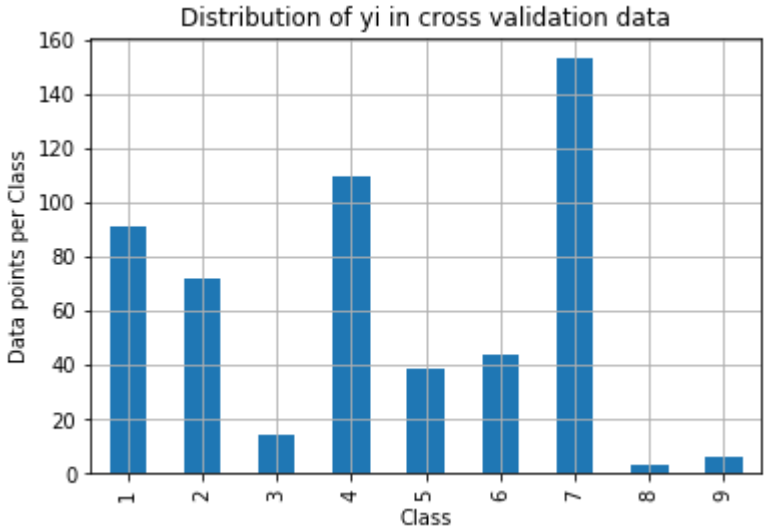
```
sing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.
values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*10
0), 3), '%)')
```




```
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 [13]:
```
# 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 rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # 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
, yticklabels=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
, yticklabels=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))
```

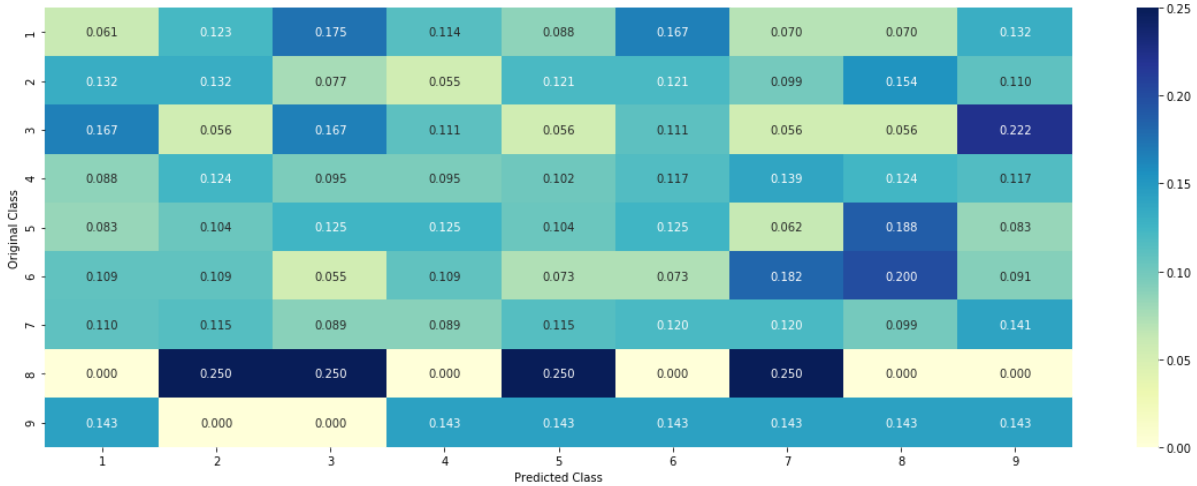
```
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels
, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [14]:
```
# 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 thei
r 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_
predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicte
d_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 2.467459274823523
Log loss on Test Data using Random Model 2.491713215993295
-------------------- Confusion matrix --------------------
```


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


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



3.3 Univariate Analysis

In [15]:
```
# 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 train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in
 class1 + 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) represen
tation 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 f
or each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature o
```

```
ccured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belon
gs 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']
=='BRCA1')])
            #         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[featu
re]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time t
hat particular feature occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alp
ha))

        # we are adding the gene/variation to the dict as key and vec as va
lue
        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, 0.0681818181
81818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.037878787878787
88, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897
95918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902,
 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06
8181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.062
5, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060
606060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546,
 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389
937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289,
 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211
9205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912,
 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333
33333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002,
 0.29999999999999999, 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
 feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it
 is there 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?

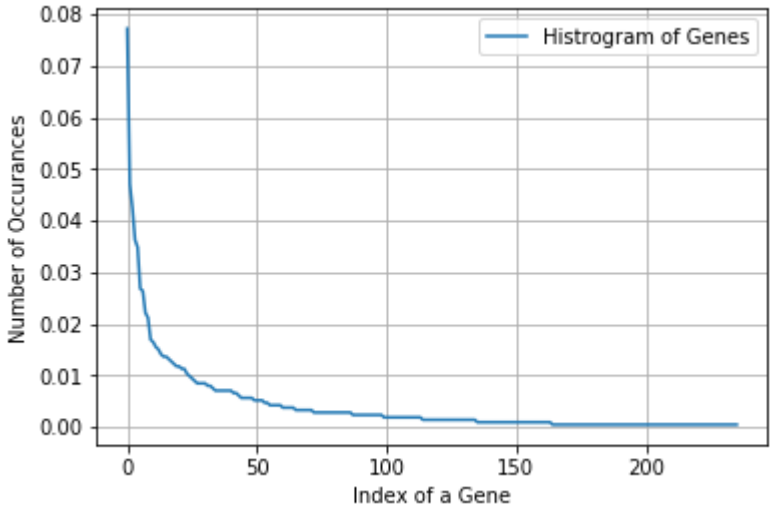
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 236
BRCA1     164
TP53      100
EGFR       90
BRCA2      77
PTEN       74
KIT        57
BRAF       56
ERBB2      47
ALK        45
PDGFRA     36
Name: Gene, dtype: int64
```

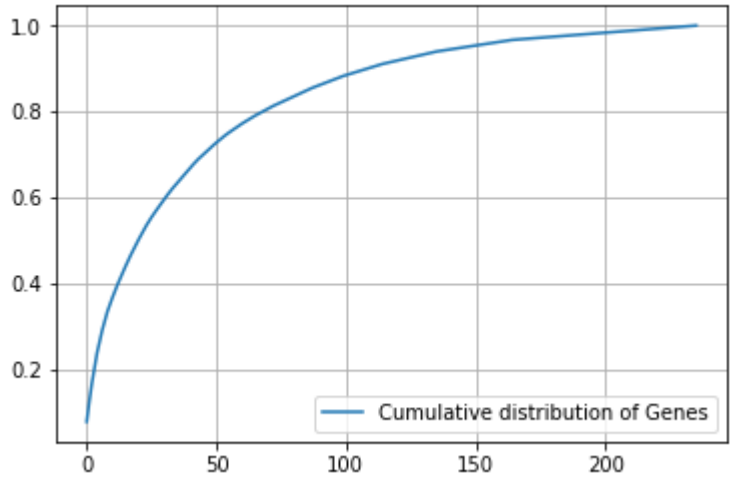
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

Ans: There are 236 different categories of genes in the train data, and they are distibuted as follows

In [18]:
```
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 [19]:
```
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 [20]:
```
#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", tra
in_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test
_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df
))
```

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
```

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
688     CDKN2A
946     PDGFRB
2491     BRCA1
2717      BRAF
995       TSC1
Name: Gene, dtype: object
```

In [24]:
```
gene_vectorizer.get_feature_names()
```

Out[24]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
```

```
                'eif1ax',
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fam58a',
                'fanca',
                'fat1',
                'fbxw7',
                'fgf3',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxo1',
                'foxp1',
                'fubp1',
                'gata3',
                'gli1',
                'gna11',
                'gnaq',
                'gnas',
                'h3f3a',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikzf1',
                'il7r',
                'inpp4b',
                'jak1',
                'jak2',
                'kdm5a',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
```

```
                'kmt2a',
                'kmt2b',
                'kmt2c',
                'kmt2d',
                'knstrn',
                'kras',
                'lats1',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'med12',
                'mef2b',
                'men1',
                'met',
                'mga',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'myod1',
                'ncor1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'notch2',
                'npm1',
                'nras',
                'nsd1',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pax8',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pik3r3',
                'pim1',
                'pms1',
                'pms2',
                'pole',
```

```
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'ros1',
'rras2',
'runx1',
'rxra',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stat3',
'stk11',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
```

```
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature:", train_gene_feature_onehotCoding.shap
e)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding m
ethod. The shape of gene feature: (2124, 236)
```

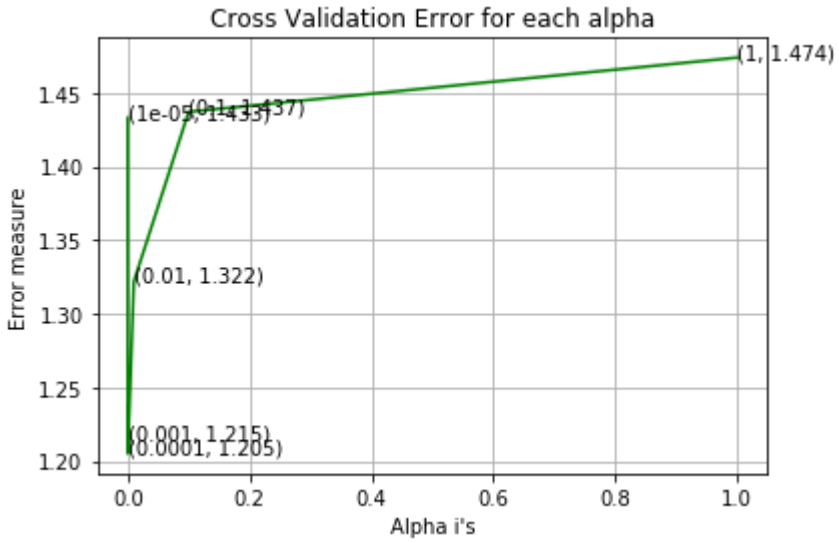
Q4. How good is this gene feature in predicting y_i?

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 [26]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/module
s/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fi
t_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learni
ng_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 Stoch
astic 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, pr
edict_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', rand
om_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(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 validati
on log 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: 1.4330459357840406
For values of alpha =  0.0001 The log loss is: 1.2049931344184501
For values of alpha =  0.001 The log loss is: 1.2145841560553023
For values of alpha =  0.01 The log loss is: 1.3217895993760151
For values of alpha =  0.1 The log loss is: 1.4371401277366276
For values of alpha =  1 The log loss is: 1.4739301360246393
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0567878366928105
For values of best alpha =  0.0001 The cross validation log loss is: 1.204993
1344184501
For values of best alpha =  0.0001 The test log loss is: 1.1952885509217699
```

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 [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",
unique_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], ":",(te
st_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  236  gen
es in train dataset?
Ans
1. In test data 646 out of 665 : 97.14285714285714
2. In cross validation data 520 out of  532 : 97.74436090225564
```

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?

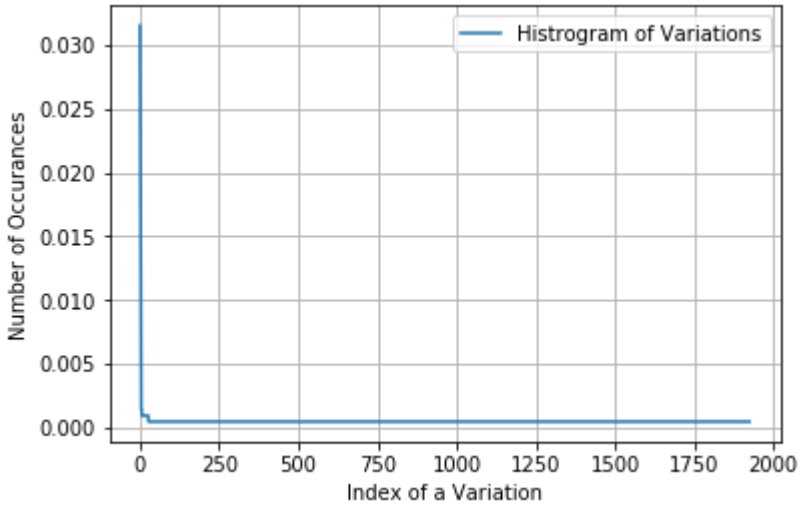
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1925
Truncating_Mutations    67
Deletion                49
Amplification           41
Fusions                 21
Overexpression           3
G12V                     3
P130S                    2
G12A                     2
T58I                     2
A146V                    2
Name: Variation, dtype: int64
```

In [29]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

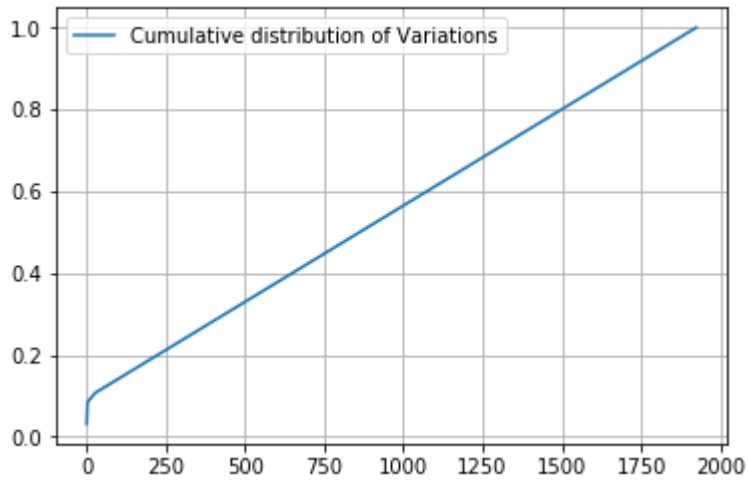
```
Ans: There are 1925 different categories of variations in the train data, and
they are distibuted as follows
```

In [30]:
```
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 [31]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.03154426 0.05461394 0.07391714 ... 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 [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [33]:
```
print("train_variation_feature_responseCoding is a converted feature using the response 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 [34]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [35]:
```
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, 1962)
```

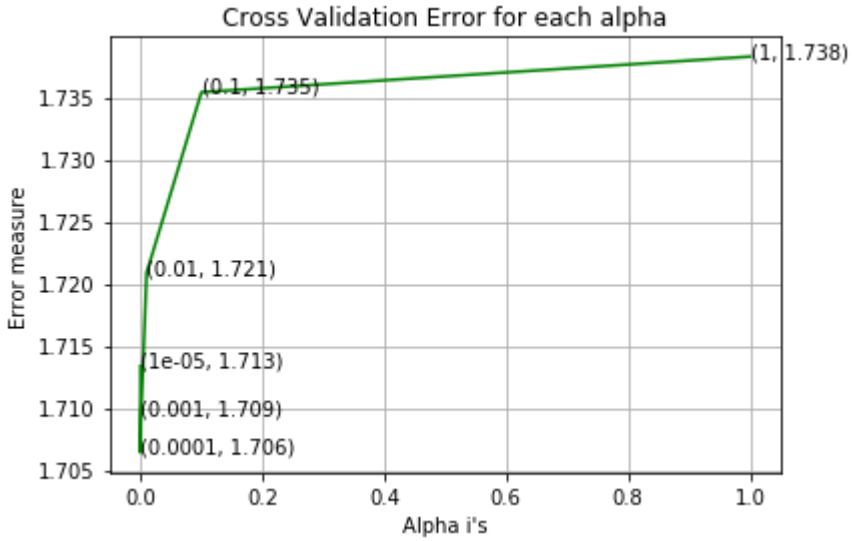
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [36]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/module
s/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fi
t_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learni
ng_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 Stoch
astic 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, pr
edict_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', rand
om_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(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 validati
on log 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: 1.7133634867662357
For values of alpha =  0.0001 The log loss is: 1.7063928606105725
For values of alpha =  0.001 The log loss is: 1.70942893628625
For values of alpha =  0.01 The log loss is: 1.7208702466850165
For values of alpha =  0.1 The log loss is: 1.735413103985182
For values of alpha =  1 The log loss is: 1.738277015675633
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7423323962986081
For values of best alpha =  0.0001 The cross validation log loss is: 1.706392
8606105725
For values of best alpha =  0.0001 The test log loss is: 1.7076499771369058
```

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 [37]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[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'])))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_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  1925  genes in test and cross
validation data sets?
Ans
1. In test data 73 out of 665 : 10.977443609022556
2. In cross validation data 49 out of  532 : 9.210526315789473
```

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 [38]:
```
# 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 [39]:
```
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/l
en(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [40]:
```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# 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 return
s (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
 times 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
))



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

```
Total number of unique words in train data : 1000
```

In [41]:
```
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 [42]:
```
#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 [43]:
```
# 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
_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [44]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=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, axi
s=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 [45]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1
] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [46]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({253.85472592979377: 1, 183.61315598541483: 1, 134.31328155390688: 1,
131.42058010117685: 1, 131.37452553001935: 1, 118.64949007951357: 1, 117.6151
4405964607: 1, 117.4842593947599: 1, 108.78267856452527: 1, 108.5682414600171
2: 1, 106.74270143925446: 1, 90.81367880274159: 1, 90.5961552973477: 1, 82.40
829302305777: 1, 81.44954123294214: 1, 80.05741213654336: 1, 79.8476798177453
3: 1, 79.03529147903217: 1, 77.79731353734692: 1, 76.84905284286371: 1, 75.45
753982392108: 1, 75.30476575942032: 1, 71.54022524569686: 1, 70.376042141475
7: 1, 68.80946517189231: 1, 67.8648497135477: 1, 66.99858509571406: 1, 65.000
89234226651: 1, 64.4994979092931: 1, 64.48339428373531: 1, 64.22615208347595:
1, 64.0695401704742: 1, 62.40330828692051: 1, 61.85172383226394: 1, 61.464505
86846852: 1, 60.21434604133533: 1, 57.67073243156497: 1, 56.68992818531152:
1, 56.675335732710174: 1, 52.05014833469474: 1, 50.247046378325614: 1, 49.689
596070229165: 1, 49.16069461304344: 1, 47.894625007429546: 1, 47.680522724000
09: 1, 47.629160743771735: 1, 47.14234960756658: 1, 46.01537197330936: 1, 45.
48687032548178: 1, 44.71128918598107: 1, 44.053050187498336: 1, 43.7342498417
3358: 1, 43.30242601252255: 1, 43.229421950546346: 1, 43.15180697991381: 1, 4
2.85843927307464: 1, 42.30386028387075: 1, 42.185538218165156: 1, 42.15504656
4230204: 1, 42.07128584379701: 1, 41.51099956603417: 1, 41.41522850116809: 1,
41.33780450664496: 1, 41.32926373513186: 1, 40.095283494183526: 1, 40.0441663
91715116: 1, 39.98992846138894: 1, 39.45686279215865: 1, 39.341366256091845:
1, 38.83324658486234: 1, 38.51652280555914: 1, 38.340903813880296: 1, 38.1991
5115487085: 1, 37.548203756727126: 1, 37.4464021443145: 1, 36.79797586704470
5: 1, 36.61007360138737: 1, 36.41420644450772: 1, 36.31143674705824: 1, 35.96
056092875907: 1, 35.55234088425858: 1, 35.35009724272548: 1, 34.811428122244
5: 1, 34.79245594811209: 1, 34.59398410202698: 1, 34.32026974591977: 1, 34.10
4981365332364: 1, 34.03874462334645: 1, 33.896027508946894: 1, 33.61471764092
626: 1, 33.46999074650181: 1, 33.0282863722076: 1, 33.017832129935286: 1, 32.
91696287606172: 1, 32.8689336139438: 1, 32.670248170982326: 1, 32.62507774300
3985: 1, 32.32284376084956: 1, 31.998335485344448: 1, 31.93452084629748: 1, 3
1.82158252828258: 1, 31.633985149728886: 1, 31.62283189217166: 1, 31.60530015
0755323: 1, 31.60351912271884: 1, 31.59565102401618: 1, 31.416866591354232:
1, 31.310718058170842: 1, 31.258469028325248: 1, 31.231025456199344: 1, 31.22
192394021683: 1, 31.154442825408033: 1, 31.129145989675095: 1, 30.96122435970
4765: 1, 30.918180267140663: 1, 30.856349584786578: 1, 30.790579512148827: 1,
30.644717161843854: 1, 30.614764406877455: 1, 30.604680774783084: 1, 30.29455
1603061034: 1, 30.291388997104757: 1, 30.27394044634792: 1, 30.04844589248926
6: 1, 29.64016128294162: 1, 29.51855109143011: 1, 29.36061881947653: 1, 29.30
0426946080755: 1, 29.257088537672317: 1, 28.9630384222879: 1, 28.789105009542
386: 1, 28.37401769052743: 1, 27.938364366963338: 1, 27.8774046853039: 1, 27.
834050413603318: 1, 27.791870460209886: 1, 27.756402999568962: 1, 27.68650914
5258533: 1, 27.500964308711133: 1, 27.359436614034692: 1, 27.2622892708659:
1, 26.979290594271063: 1, 26.958456133544598: 1, 26.787594077164805: 1, 26.75
5061259218973: 1, 26.351168258749375: 1, 26.28931850717165: 1, 26.26762405306
0494: 1, 26.031948714679245: 1, 26.021916969679967: 1, 26.010377008055727: 1,
25.904920193257546: 1, 25.75477387078562: 1, 25.67385649995869: 1, 25.6731226
44889784: 1, 25.64215995891992: 1, 25.519891604641074: 1, 25.359686098995844:
1, 25.312097388178678: 1, 25.00127806537085: 1, 24.92463508286752: 1, 24.9121
54031072717: 1, 24.82558507395981: 1, 24.81658918800779: 1, 24.71213138623944
5: 1, 24.67814689037456: 1, 24.534286341134855: 1, 24.47770874394808: 1, 24.4
6183712302472: 1, 24.422328383498677: 1, 24.397546861205853: 1, 24.3736370097
48013: 1, 24.131079633355235: 1, 24.11500582328209: 1, 24.09087623525415: 1,
24.044049388024042: 1, 24.033824553651254: 1, 24.026485634527695: 1, 24.00027
205448451: 1, 23.989130747511783: 1, 23.74674628383244: 1, 23.56582602202909
3: 1, 23.546873257649448: 1, 23.40696297956194: 1, 23.3681187254458: 1, 23.22
8484378142277: 1, 23.212878028078205: 1, 23.191097300262424: 1, 23.1397231858
0417: 1, 23.032335073931474: 1, 22.954162279314893: 1, 22.852128063832524: 1,
22.807278833768162: 1, 22.805505784777445: 1, 22.752918365524955: 1, 22.72505
```

603735834: 1, 22.725055354840002: 1, 22.72209073983693: 1, 22.63851507599602
2: 1, 22.532240429873013: 1, 22.494825847325608: 1, 22.46110037908247: 1, 22.
44079934705962: 1, 22.440361208951444: 1, 22.376449918027024: 1, 22.270695177
775675: 1, 22.24626209172947: 1, 22.217259391505678: 1, 22.18283106581108: 1,
22.17756581592672: 1, 22.174421246770848: 1, 22.138307746312464: 1, 22.113557
40312592: 1, 22.03095465765509: 1, 21.939138713138043: 1, 21.91805870984033:
1, 21.895586049884283: 1, 21.865769199853485: 1, 21.776095443773368: 1, 21.66
0319110918536: 1, 21.63918018676391: 1, 21.614702247775945: 1, 21.54590234731
754: 1, 21.52816024526122: 1, 21.439068429988914: 1, 21.382590078320916: 1, 2
1.26995936521888: 1, 21.2552928246447: 1, 21.18042212465077: 1, 21.0601534626
8273: 1, 21.038378145071857: 1, 20.940812938710515: 1, 20.854760796067158: 1,
20.787728956575993: 1, 20.63883965377128: 1, 20.587884279352725: 1, 20.478490
817047007: 1, 20.461352690627084: 1, 20.44839449379414: 1, 20.43894143192773
4: 1, 20.36784889361909: 1, 20.355259729505153: 1, 20.33468173812753: 1, 20.3
225925243392: 1, 20.279013000347625: 1, 20.270782592788347: 1, 20.23694180016
686: 1, 20.208095528201884: 1, 20.15906198506319: 1, 20.137328411573957: 1, 2
0.10276989532524: 1, 20.054518737086955: 1, 20.045695384043103: 1, 20.0242805
00368157: 1, 19.93772829534826: 1, 19.912416951545715: 1, 19.863145004455504:
1, 19.825459099798948: 1, 19.807424626185497: 1, 19.64381349404674: 1, 19.641
290221184654: 1, 19.582522968256466: 1, 19.573433309120592: 1, 19.55055273745
685: 1, 19.54518350439682: 1, 19.513222091716123: 1, 19.48329994920147: 1, 1
9.4140554296068: 1, 19.370306968920687: 1, 19.328070105312026: 1, 19.30686698
4651837: 1, 19.274297093870906: 1, 19.265018484576327: 1, 19.262048374700587:
1, 19.259002902646696: 1, 19.25709158950889: 1, 19.23201791191461: 1, 19.2254
74192435954: 1, 19.198725346626297: 1, 19.165045697553794: 1, 19.132499740792
4: 1, 19.084816810156735: 1, 19.028272351469003: 1, 19.000491505626815: 1, 1
8.93824601590173: 1, 18.92540259463911: 1, 18.8986508444414: 1, 18.7085096124
2584: 1, 18.70144698689399: 1, 18.68503665938295: 1, 18.665984675622283: 1, 1
8.63096986324031: 1, 18.571495500179036: 1, 18.55466505673964: 1, 18.53016621
755285: 1, 18.514045195097925: 1, 18.443365577361305: 1, 18.441311797080587:
1, 18.415590199896677: 1, 18.381005757132936: 1, 18.370027155795288: 1, 18.36
2444513347338: 1, 18.30136270652897: 1, 18.276241462331573: 1, 18.24843286606
425: 1, 18.23364338613229: 1, 18.13847317971105: 1, 18.132561232092293: 1, 1
8.08990306945418: 1, 18.051859916596538: 1, 18.046611920726313: 1, 18.0285476
49857217: 1, 18.025542670142407: 1, 17.963508425413444: 1, 17.81842020279325
6: 1, 17.760506512640443: 1, 17.76030150171245: 1, 17.75988309825059: 1, 17.7
3147647648176: 1, 17.72961436075669: 1, 17.72118462392147: 1, 17.713380283168
345: 1, 17.693299392846193: 1, 17.665694364268333: 1, 17.61354265099691: 1, 1
7.578277564162825: 1, 17.538447446187316: 1, 17.50958831897216: 1, 17.4991842
62553733: 1, 17.464933239360807: 1, 17.455071912914523: 1, 17.45286262086722
6: 1, 17.44798470873309: 1, 17.39914063085675: 1, 17.37727137425398: 1, 17.36
760623253496: 1, 17.360315388699334: 1, 17.343306202685596: 1, 17.27170837337
6743: 1, 17.233950796833064: 1, 17.210907013376808: 1, 17.15954889459835: 1,
17.15875383179838: 1, 17.102765466069492: 1, 17.079112559997256: 1, 17.066876
1312892: 1, 17.04388285012143: 1, 17.001340367450318: 1, 16.977358209656312:
1, 16.92330503934576: 1, 16.861076340792312: 1, 16.816052502660508: 1, 16.804
253240187077: 1, 16.7982085450531: 1, 16.78630987923258: 1, 16.76766482469908
2: 1, 16.69533968013994: 1, 16.559798511190372: 1, 16.556364954962394: 1, 16.
54643255740469: 1, 16.544276275550125: 1, 16.54218542369692: 1, 16.5087712946
56288: 1, 16.50526151141417: 1, 16.425983106383526: 1, 16.382600072360844: 1,
16.373327318069585: 1, 16.352011643975636: 1, 16.338465041162593: 1, 16.33702
666542796: 1, 16.299159214147267: 1, 16.28517558356301: 1, 16.25772395765665
5: 1, 16.24852727730768: 1, 16.194494714649377: 1, 16.19047810090522: 1, 16.1
36800781576564: 1, 16.066649208347602: 1, 16.062523184969315: 1, 16.057661070
29841: 1, 16.050618453073167: 1, 16.048093537861174: 1, 15.983990632377187:
1, 15.960666547623083: 1, 15.950355883755291: 1, 15.906147470053536: 1, 15.89
6475489411964: 1, 15.853343076615399: 1, 15.841148213475776: 1, 15.8326877530

1163: 1, 15.831965751784658: 1, 15.800913862849798: 1, 15.789533326038713: 1,
15.766560054228954: 1, 15.741472833620806: 1, 15.673933567203573: 1, 15.66049
5068232978: 1, 15.65813883846676: 1, 15.628924241337284: 1, 15.61779365977119
7: 1, 15.591938024215443: 1, 15.534110066665917: 1, 15.5161364197691: 1, 15.4
84519834413895: 1, 15.482990334447859: 1, 15.322456500942623: 1, 15.312253709
883036: 1, 15.296188534595407: 1, 15.267736536923557: 1, 15.209265359533353:
1, 15.203833540196449: 1, 15.170029524914513: 1, 15.140131360682943: 1, 15.12
9299325192028: 1, 15.121925122038116: 1, 15.093527661007176: 1, 15.0677453655
60343: 1, 15.067722115431158: 1, 15.056409058372191: 1, 15.025150952249408:
1, 14.991252607951786: 1, 14.982864484770674: 1, 14.9738872848769: 1, 14.9683
24732256246: 1, 14.963907537860184: 1, 14.962432666903174: 1, 14.939541428481
556: 1, 14.904568204115405: 1, 14.858924788508745: 1, 14.856015136323444: 1,
14.834877902833043: 1, 14.832157072065167: 1, 14.815015305490427: 1, 14.80123
8704572631: 1, 14.780541351658814: 1, 14.77472642475864: 1, 14.6964290964502
5: 1, 14.685565128364296: 1, 14.674429483110222: 1, 14.664548172221124: 1, 1
4.662534407158278: 1, 14.654372594536817: 1, 14.64616681083947: 1, 14.6158124
81512595: 1, 14.605221344158078: 1, 14.574402956253072: 1, 14.54157125407733
3: 1, 14.533857272629772: 1, 14.524699989108825: 1, 14.501798741127475: 1, 1
4.493358278274: 1, 14.477611035159146: 1, 14.47580518271494: 1, 14.4695758886
07746: 1, 14.436754803220783: 1, 14.419808887941665: 1, 14.414095667720229:
1, 14.398275194759774: 1, 14.395971022575289: 1, 14.371749263326837: 1, 14.36
711334709881: 1, 14.33265832751196: 1, 14.323112523947978: 1, 14.253736639661
408: 1, 14.22382831319778: 1, 14.206811387762203: 1, 14.193104907571028: 1, 1
4.183161466819087: 1, 14.137670128079629: 1, 14.09931928310159: 1, 14.0826814
04991098: 1, 14.073055095437752: 1, 14.063953086423815: 1, 14.05236316480921
1: 1, 14.048494128229: 1, 14.029978685089409: 1, 14.007395855445688: 1, 13.98
7673522359433: 1, 13.97580541021175: 1, 13.966413115904018: 1, 13.93681147273
4075: 1, 13.89534918110312: 1, 13.861178697794514: 1, 13.831871996530529: 1,
13.74276390609319: 1, 13.734287526302667: 1, 13.715190439100603: 1, 13.689576
32063428: 1, 13.6770103809989: 1, 13.622214999270199: 1, 13.501950028578447:
1, 13.455791168262778: 1, 13.434984346498691: 1, 13.43197575220311: 1, 13.405
553460771563: 1, 13.316570608678086: 1, 13.315484734645272: 1, 13.30298135730
0384: 1, 13.28565124266038: 1, 13.266123883657778: 1, 13.2610688656881: 1, 1
3.253559238284012: 1, 13.250556695678013: 1, 13.21022727763969: 1, 13.2047548
3197904: 1, 13.197261050155875: 1, 13.193735253175353: 1, 13.126053382846168:
1, 13.080823599432781: 1, 13.070368600429223: 1, 13.066547079044048: 1, 13.02
77334645035: 1, 13.017398417281806: 1, 12.950758681646056: 1, 12.913752425167
5: 1, 12.846060027201466: 1, 12.831609427622483: 1, 12.831497817241512: 1, 1
2.81277087684841: 1, 12.807587086053298: 1, 12.803956246933904: 1, 12.7720902
8235967: 1, 12.762664035823759: 1, 12.755468267320104: 1, 12.754703374312195:
1, 12.70845552796984: 1, 12.707214949998297: 1, 12.659261278667163: 1, 12.637
45286355892: 1, 12.633742101940964: 1, 12.613179250848605: 1, 12.569553582380
37: 1, 12.56025649188652: 1, 12.556426862946646: 1, 12.511136017341054: 1, 1
2.50489987911921: 1, 12.485261313587007: 1, 12.473970202372616: 1, 12.4703194
9489811: 1, 12.468499177497698: 1, 12.397644874298445: 1, 12.393207972192984:
1, 12.391058509913774: 1, 12.389238163025869: 1, 12.387009624186016: 1, 12.38
1501450423363: 1, 12.359540044952706: 1, 12.331220134754313: 1, 12.3299296257
28422: 1, 12.298671246916628: 1, 12.279724014636713: 1, 12.256370104704132:
1, 12.25574986284503: 1, 12.252793397887595: 1, 12.251525534133581: 1, 12.239
607054995371: 1, 12.231816921946844: 1, 12.228570370867027: 1, 12.20036839976
4969: 1, 12.195766923290586: 1, 12.19269398365085: 1, 12.183640219513505: 1,
12.151847761995793: 1, 12.147417487778883: 1, 12.1454934685885: 1, 12.1378532
19778123: 1, 12.131244400735886: 1, 12.130178173204502: 1, 12.09802823380547
7: 1, 12.088536654865573: 1, 12.045856551482961: 1, 12.028960420892451: 1, 1
2.013539168152818: 1, 12.003485990049768: 1, 12.00084642988405: 1, 11.9970262
99505496: 1, 11.980883671613167: 1, 11.980224756295426: 1, 11.97910731493826
3: 1, 11.913588701693113: 1, 11.911052865745301: 1, 11.898637809323152: 1, 1

1.884074948275284: 1, 11.870177661135084: 1, 11.836971858594382: 1, 11.814949
428430637: 1, 11.801623919649558: 1, 11.797342751429717: 1, 11.78443284186226
8: 1, 11.772028367082486: 1, 11.764600445608721: 1, 11.73703029487872: 1, 11.
667441723169734: 1, 11.641847777204314: 1, 11.635635824760836: 1, 11.61942273
7883596: 1, 11.599774654271123: 1, 11.572286067251772: 1, 11.569954246009612:
1, 11.560936787674745: 1, 11.555818097228695: 1, 11.532819696893911: 1, 11.53
150763313995: 1, 11.44838683988851: 1, 11.44049269470859: 1, 11.4202901818643
49: 1, 11.406405493155267: 1, 11.391103751302781: 1, 11.37299584236188: 1, 1
1.361578757236153: 1, 11.358462661218677: 1, 11.331505294534388: 1, 11.331208
0197191: 1, 11.311655427310908: 1, 11.307490394042949: 1, 11.2994423847343:
1, 11.296115688618814: 1, 11.286693754710653: 1, 11.28565848607295: 1, 11.278
561072943143: 1, 11.240060579781733: 1, 11.238412474455743: 1, 11.22216475104
7268: 1, 11.214131462320358: 1, 11.19225241244591: 1, 11.186046714558321: 1,
11.180298589133315: 1, 11.161543162956509: 1, 11.152309148842209: 1, 11.14893
1285597142: 1, 11.134658496156348: 1, 11.109836248065584: 1, 11.0975904512326
7: 1, 11.078838505799048: 1, 11.070932414657674: 1, 11.05732201096228: 1, 11.
055356011371584: 1, 11.037673509711533: 1, 11.019908077755627: 1, 11.01601162
8678164: 1, 11.013889778471636: 1, 11.01350620911677: 1, 10.99198738176902:
1, 10.97226339257677: 1, 10.971242812087599: 1, 10.951799236808338: 1, 10.939
435519672333: 1, 10.933557437557665: 1, 10.926122836095194: 1, 10.92249179743
6914: 1, 10.919921093956228: 1, 10.917297248787722: 1, 10.915949980583557: 1,
10.910370020857071: 1, 10.868825609029656: 1, 10.862138285203505: 1, 10.83576
1538531807: 1, 10.8288334075309: 1, 10.802872866272358: 1, 10.78328806065583
7: 1, 10.782432135437185: 1, 10.757756814440395: 1, 10.75071050353995: 1, 10.
744745214755413: 1, 10.726564418599674: 1, 10.707062424980522: 1, 10.69296282
0203624: 1, 10.689539905456241: 1, 10.677879392331452: 1, 10.673011379074694:
1, 10.670522537208505: 1, 10.664533023848222: 1, 10.621246154279126: 1, 10.60
8402274815111: 1, 10.600325352449788: 1, 10.588645713006585: 1, 10.5769171439
23405: 1, 10.570324149904382: 1, 10.569146589274375: 1, 10.563308837262726:
1, 10.562739719122108: 1, 10.553354394362538: 1, 10.539363209951185: 1, 10.53
3278191507753: 1, 10.502603789208464: 1, 10.461524241474171: 1, 10.4261515339
49739: 1, 10.419999329800762: 1, 10.416615989778157: 1, 10.387724968669621:
1, 10.380188040909802: 1, 10.371480969448045: 1, 10.361769746054504: 1, 10.34
1890558193166: 1, 10.334219041793444: 1, 10.329827333674684: 1, 10.3122418498
82943: 1, 10.304081300661009: 1, 10.303926527549189: 1, 10.299621275988118:
1, 10.29191391232052: 1, 10.283823100742596: 1, 10.267003901568042: 1, 10.258
426531087954: 1, 10.25815493438342: 1, 10.256705668962525: 1, 10.252236403224
481: 1, 10.23838028002355: 1, 10.20862443174968: 1, 10.199950788561418: 1, 1
0.19026883954898: 1, 10.158239750053541: 1, 10.12266046405398: 1, 10.10988944
024965: 1, 10.10564101852161: 1, 10.104866051879617: 1, 10.093350890024066:
1, 10.057682472628398: 1, 10.057312511890167: 1, 10.056077596683341: 1, 10.04
999057087892: 1, 10.048059300849037: 1, 10.043682928944559: 1, 10.00602218090
2553: 1, 9.996001344014836: 1, 9.989277881332061: 1, 9.98258392007464: 1, 9.9
7875832054429: 1, 9.975223162431059: 1, 9.951285826514727: 1, 9.9493617963875
44: 1, 9.935518810730809: 1, 9.92073920352857: 1, 9.913458031725606: 1, 9.911
11605311534: 1, 9.905811348174572: 1, 9.889212243315288: 1, 9.88665968625444
3: 1, 9.883127385491377: 1, 9.877635398874862: 1, 9.875915268876811: 1, 9.865
865501988193: 1, 9.85510947872764: 1, 9.783389076521097: 1, 9.75785611823855
7: 1, 9.739807151401177: 1, 9.736379703852426: 1, 9.725712243338073: 1, 9.707
084752306367: 1, 9.705883759612783: 1, 9.698783102957524: 1, 9.6814546552798
7: 1, 9.657941007744913: 1, 9.641214973038224: 1, 9.630591495390012: 1, 9.605
187056400727: 1, 9.604428061258156: 1, 9.595841795135833: 1, 9.57927924441302
6: 1, 9.577926973096508: 1, 9.572778795196387: 1, 9.558524750894003: 1, 9.556
81955874339: 1, 9.551733593412317: 1, 9.53384639883566: 1, 9.52091770202146:
1, 9.510579378255283: 1, 9.509165168365207: 1, 9.499555041782234: 1, 9.489504
352489643: 1, 9.486095599567184: 1, 9.481032042866838: 1, 9.47838868027922:
1, 9.467403252194355: 1, 9.466453065288537: 1, 9.465325068995709: 1, 9.453984

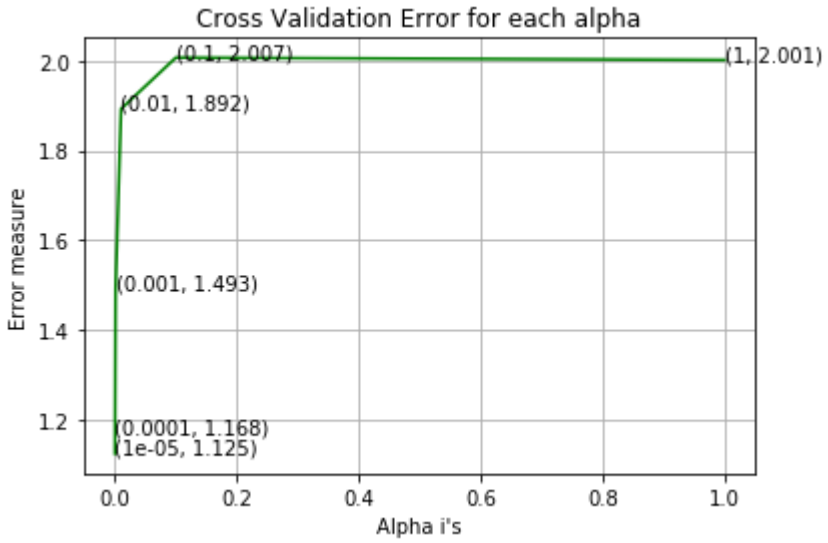
481555949: 1, 9.450165402220687: 1, 9.449880158083259: 1, 9.441820238144158:
1, 9.42293257095529: 1, 9.40428235631894: 1, 9.39999485777495: 1, 9.397924135
70823: 1, 9.389023860890212: 1, 9.385337255616395: 1, 9.382432515240522: 1,
9.380100982327697: 1, 9.374876035398598: 1, 9.359617862744095: 1, 9.356977055
537953: 1, 9.349087388059639: 1, 9.341354535394773: 1, 9.3389526434733: 1, 9.
335810333252603: 1, 9.331709843631154: 1, 9.318078552357365: 1, 9.28802778087
2106: 1, 9.287009769780525: 1, 9.283811601856284: 1, 9.271057251771722: 1, 9.
250397597526142: 1, 9.248023411862983: 1, 9.235708717110175: 1, 9.22478453451
3978: 1, 9.223201301692333: 1, 9.191688861087766: 1, 9.190682275015563: 1, 9.
189131793680321: 1, 9.188380521504339: 1, 9.174900854032757: 1, 9.17198557852
8984: 1, 9.16751912503272: 1, 9.154161665809077: 1, 9.154125681486274: 1, 9.1
10984768890994: 1, 9.110206062863162: 1, 9.093804724442036: 1, 9.077789374092
788: 1, 9.073537345491284: 1, 9.065049577222378: 1, 9.055509854804203: 1, 9.0
54785638943468: 1, 9.042990754529521: 1, 9.018645957560073: 1, 9.003245182656
523: 1, 8.987455905790885: 1, 8.97905777731065: 1, 8.973193511722586: 1, 8.96
9453051319801: 1, 8.947523276152834: 1, 8.91691894373859: 1, 8.9162692125115
2: 1, 8.906292321521615: 1, 8.890647907882022: 1, 8.890279155181114: 1, 8.879
666819604264: 1, 8.849352145198386: 1, 8.839084319957133: 1, 8.83647627516504
7: 1, 8.832535920543213: 1, 8.828810191855382: 1, 8.826849504713486: 1, 8.770
778701475027: 1, 8.768813860815772: 1, 8.766409669544862: 1, 8.76618182947004
6: 1, 8.759336972033276: 1, 8.758236743115509: 1, 8.750155452513727: 1, 8.749
358086978086: 1, 8.748632954762519: 1, 8.748495900869345: 1, 8.7388774792759
1: 1, 8.73782605586867: 1, 8.724065072354456: 1, 8.718368001554074: 1, 8.7072
4597756142: 1, 8.699380068252115: 1, 8.694146211930663: 1, 8.65892102647181:
1, 8.657618684969393: 1, 8.65113885619279: 1, 8.6355596474269: 1, 8.633106091
534923: 1, 8.610681630599496: 1, 8.579588667623037: 1, 8.57725782300889: 1,
8.570051606444213: 1, 8.52740264600878: 1, 8.52617041386543: 1, 8.51304186355
5435: 1, 8.500724681117568: 1, 8.49187799146575: 1, 8.470075582688256: 1, 8.4
36020858094654: 1, 8.426387351964006: 1, 8.408588860807571: 1, 8.399042002702
657: 1, 8.37711626466487: 1, 8.374510423140018: 1, 8.349127392225927: 1, 8.34
682134308601: 1, 8.336442764050627: 1, 8.333361879308319: 1, 8.33206250981304
7: 1, 8.331686090559076: 1, 8.327889698515714: 1, 8.322958174936987: 1, 8.278
961198111546: 1, 8.274396599241099: 1, 8.269542470576528: 1, 8.25360747596204
7: 1, 8.248707294845191: 1, 8.248587893373925: 1, 8.243742903213743: 1, 8.238
93238832179: 1, 8.231470560654568: 1, 8.203284135383331: 1, 8.18779869150782
4: 1, 8.176958628389253: 1, 8.173523384293382: 1, 8.171309445575655: 1, 8.165
005861205826: 1, 8.151095949607491: 1, 8.136521580995431: 1, 8.11790204437533
6: 1, 8.116687653943174: 1, 8.113934671849002: 1, 8.10483726607574: 1, 8.0978
83038222875: 1, 8.054588422613538: 1, 8.043524392343699: 1, 8.02126155406269
5: 1, 8.003143264604823: 1, 7.986346564603411: 1, 7.981857304249641: 1, 7.979
241405341871: 1, 7.970911617216507: 1, 7.96806826733513: 1, 7.951965834875406
5: 1, 7.941581044405896: 1, 7.930791317184849: 1, 7.924504787832427: 1, 7.908
887784278071: 1, 7.908451452504693: 1, 7.904718728948511: 1, 7.90470286387243
9: 1, 7.895645445511015: 1, 7.856461602982365: 1, 7.8311147558429095: 1, 7.82
716807371965: 1, 7.812784251872369: 1, 7.809652685118723: 1, 7.80701138445241
6: 1, 7.802488022308964: 1, 7.79047883989421: 1, 7.768664419641586: 1, 7.7646
67460493008: 1, 7.763358787638232: 1, 7.747623940478052: 1, 7.74436695350597:
1, 7.736083991345167: 1, 7.73207851967559: 1, 7.729212513269465: 1, 7.7274061
152437925: 1, 7.7240166008277535: 1, 7.7158723353190615: 1, 7.698591066838365
5: 1, 7.69772115823524: 1, 7.689948030166464: 1, 7.615840858449021: 1, 7.5950
9624539242: 1, 7.544467865568635: 1, 7.540233115737403: 1, 7.536408651047097:
1, 7.529751547109881: 1, 7.437398883437731: 1, 7.430523561720216: 1, 7.423137
181989063: 1, 7.413002798073885: 1, 7.410675373537033: 1, 7.395497469833948:
1, 7.378181071420048: 1, 7.377472237550159: 1, 7.342362831307769: 1, 7.335884
4288563665: 1, 7.33412823317105: 1, 7.327687710197005: 1, 7.311706241383929:
1, 7.310948626920256: 1, 7.29000842089378: 1, 7.279686078708525: 1, 7.2629648
9439598: 1, 7.132204868141112: 1, 7.129186407516236: 1, 7.100719187352202: 1,

```
7.094024191222651: 1, 7.093726990148904: 1, 7.054309831447967: 1, 7.024799544
817146: 1, 6.991052889803881: 1, 6.983149814009753: 1, 6.9791196679212515: 1,
6.960489914011119: 1, 6.930053079375017: 1, 6.921948479214439: 1, 6.885122427
270136: 1, 6.868830575309099: 1, 6.86051939542645: 1, 6.827084986762227: 1,
6.8184607276902: 1, 6.766418948820842: 1, 6.7383152964602555: 1, 6.7150600042
22928: 1, 6.661775413493007: 1, 6.618600664963325: 1, 6.523016851772004: 1,
6.483369348435566: 1, 6.321317923921494: 1})
```

In [47]:
```
# 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/module
s/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fi
t_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learni
ng_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 Stoch
astic 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, pr
edict_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', rand
om_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(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 validati
on log 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: 1.1247373975259722
For values of alpha =  0.0001 The log loss is: 1.1684605363001976
For values of alpha =  0.001 The log loss is: 1.492688662666901
For values of alpha =  0.01 The log loss is: 1.8923232486472448
For values of alpha =  0.1 The log loss is: 2.006500608840076
For values of alpha =  1 The log loss is: 2.0012330027725733
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7894093294625585
For values of best alpha =  1e-05 The cross validation log loss is: 1.1247373
975259722
For values of best alpha =  1e-05 The test log loss is: 1.1062789750449955
```

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

Ans. Yes, it seems like!

In [48]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    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 [49]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train
 data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in
 train data")
```

```
94.1 % of word of test data appeared in train data
93.4 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:
```
#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 bel
ongs 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 [51]:
```
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 [52]:
```
# 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 [{}]".f
ormat(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 [{}]".f
ormat(word,yes_no))

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

Stacking the three types of features

In [53]:
```
# 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
_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_va
riation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variatio
n_feature_onehotCoding))

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCodin
g,train_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_v
ariation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_te
xt_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_
feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_featur
e_responseCoding))
```

In [54]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.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, 3198)
(number of data points * number of features) in test data =  (665, 3198)
(number of data points * number of features) in cross validation data = (532, 3198)
```

In [55]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.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, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model

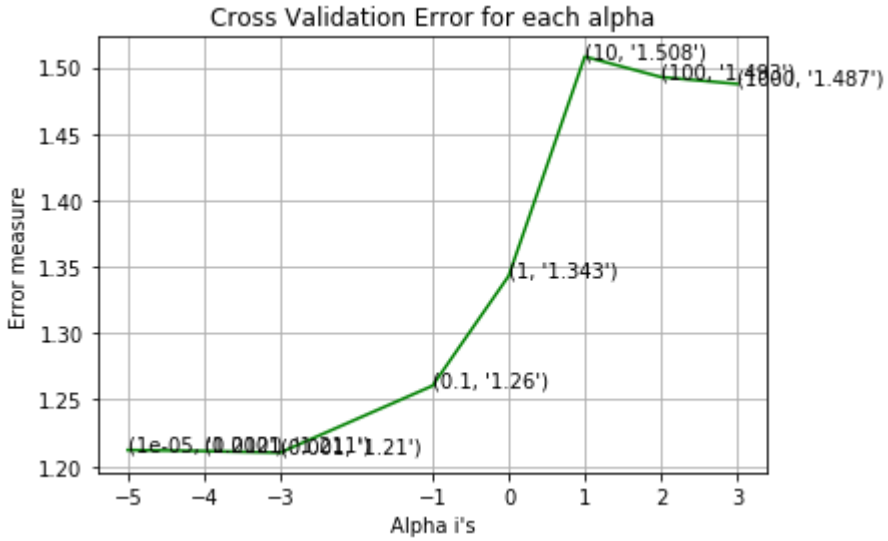
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [56]:
```
# 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 vecto
r X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-onli
ne/lessons/naive-bayes-algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/st
able/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='s
igmoid', 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-onli
ne/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.clas
ses_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-pro
bability 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 validati
on 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 = ', 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 : 1.2121675202969273
for alpha = 0.0001
Log Loss : 1.2114320662173117
for alpha = 0.001
Log Loss : 1.21020483263532
for alpha = 0.1
Log Loss : 1.260445134193953
for alpha = 1
Log Loss : 1.3429752823187375
for alpha = 10
Log Loss : 1.5079178909241846
for alpha = 100
Log Loss : 1.4925606745817082
for alpha = 1000
Log Loss : 1.487397594355744
```

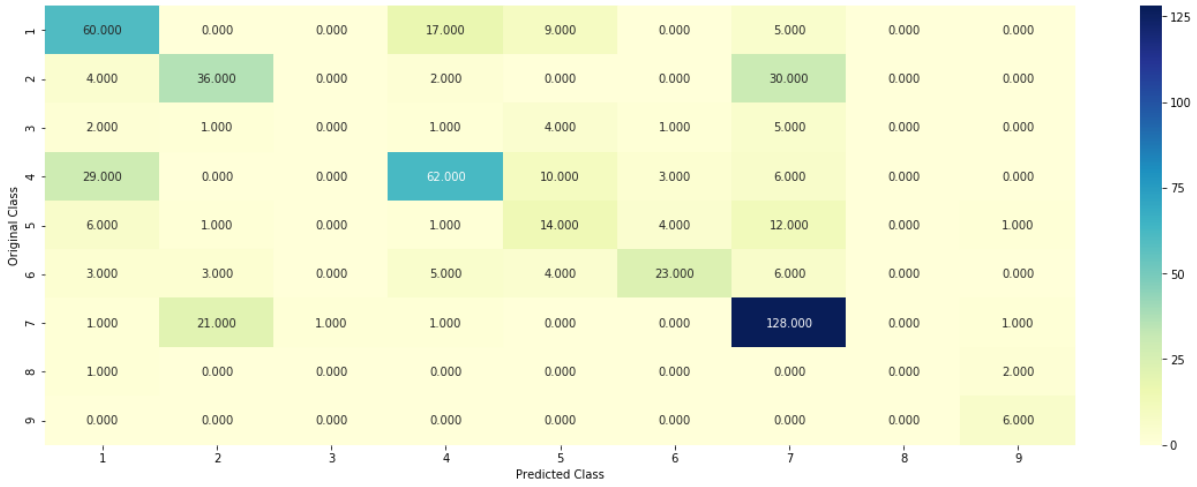



```
For values of best alpha =  0.001 The train log loss is: 0.532888131953395
For values of best alpha =  0.001 The cross validation log loss is: 1.2102048
3263532
For values of best alpha =  0.001 The test log loss is: 1.181352005765107
```

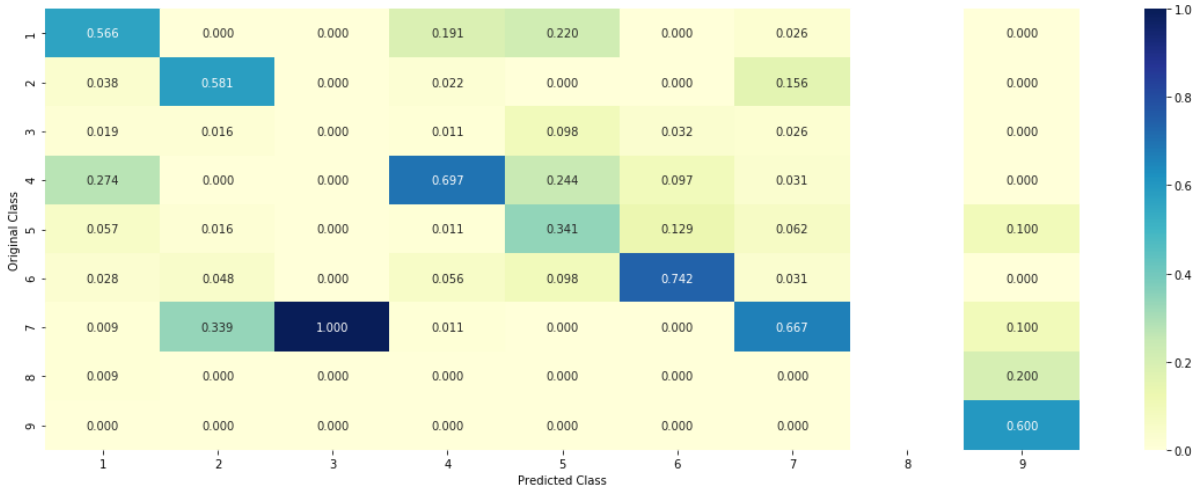
4.1.1.2. Testing the model with best hyper paramters

In [57]:
```
# 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()))
```

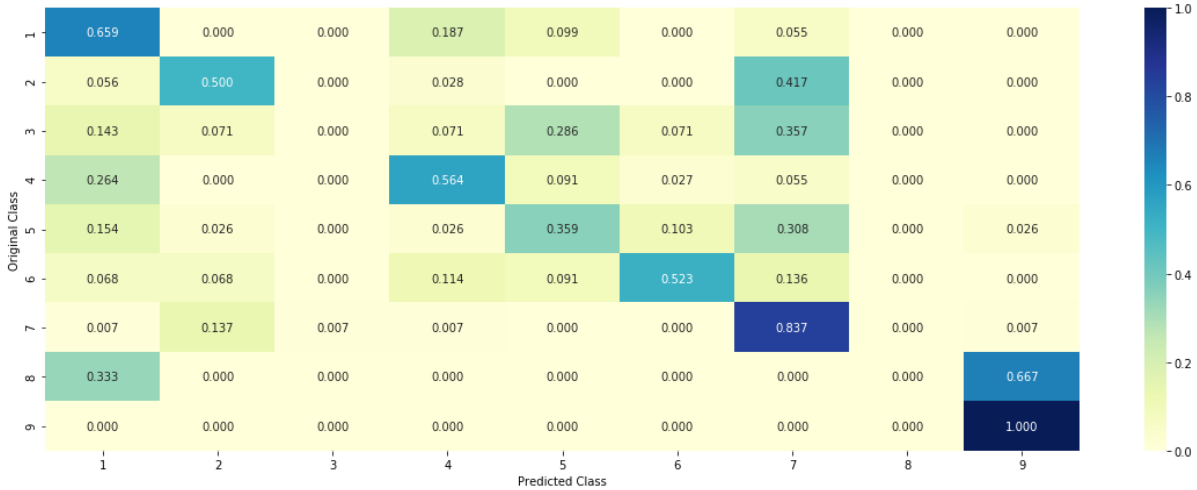
```
Log Loss : 1.21020483263532
Number of missclassified point : 0.3815789473684211
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [58]:
```
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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0625 0.049  0.0105 0.0687 0.0296 0.0286 0.
7442 0.0046 0.0023]]
Actual Class : 7
--------------------------------------------------
36 Text feature [114] present in test data point [True]
49 Text feature [117] present in test data point [True]
60 Text feature [130] present in test data point [True]
74 Text feature [11] present in test data point [True]
79 Text feature [110] present in test data point [True]
99 Text feature [106] present in test data point [True]
Out of the top  100  features  6 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:
```
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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.063  0.0668 0.0133 0.0646 0.0308 0.0298 0.
7244 0.0048 0.0024]]
Actual Class : 7
--------------------------------------------------
74 Text feature [11] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

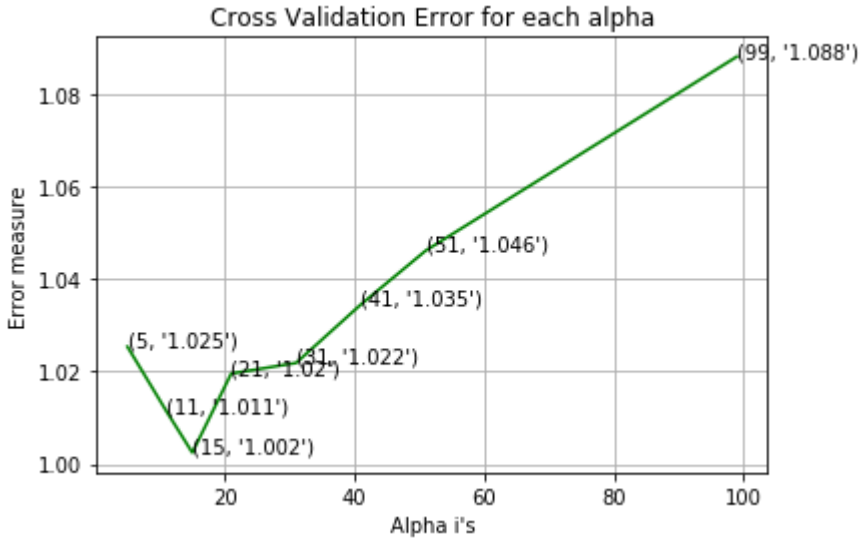
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [83]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto',
 leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


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-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 = 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 validati
on 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[best_alpha], "The test log loss
 is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0254040126027122
for alpha = 11
Log Loss : 1.0114376021287863
for alpha = 15
Log Loss : 1.002444969002307
for alpha = 21
Log Loss : 1.0195860069386846
for alpha = 31
Log Loss : 1.02188287913295
for alpha = 41
Log Loss : 1.0345527745182799
for alpha = 51
Log Loss : 1.0462259378800565
for alpha = 99
Log Loss : 1.0882805268305706
```

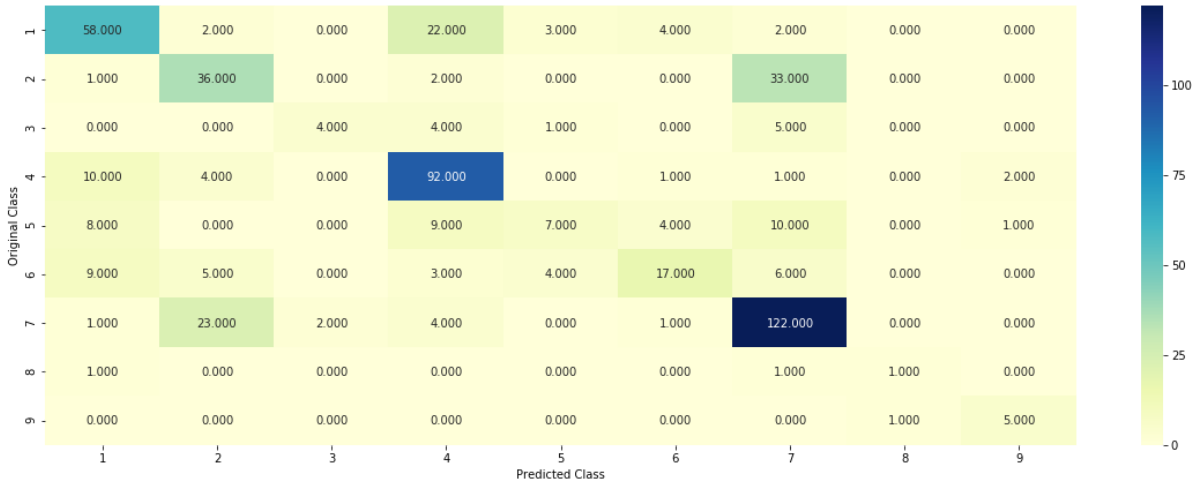



```
For values of best alpha =  15 The train log loss is: 0.6955823832520973
For values of best alpha =  15 The cross validation log loss is: 1.0024449690
02307
For values of best alpha =  15 The test log loss is: 1.0994540919206754
```

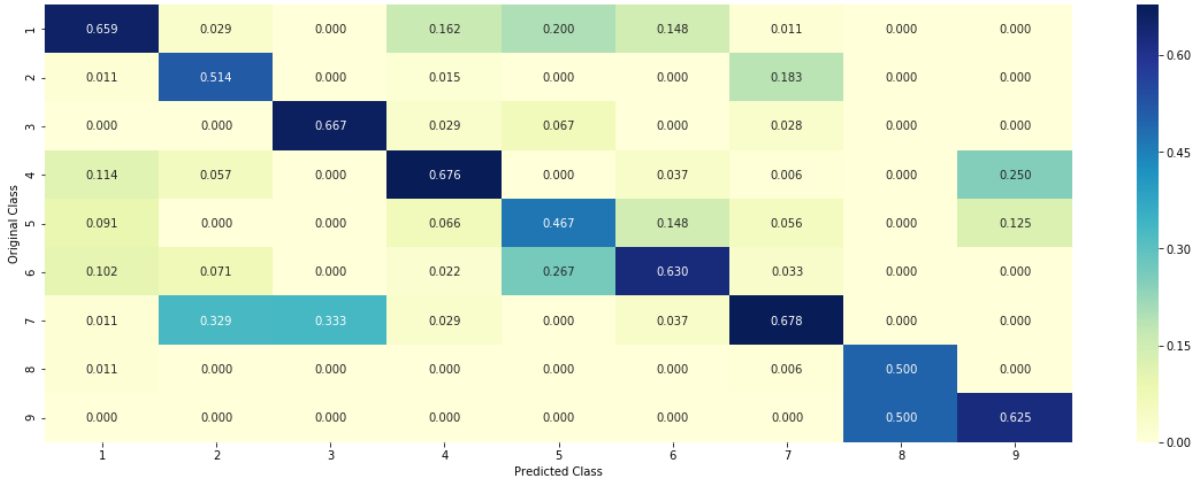
4.2.2. Testing the model with best hyper paramters

In [84]:
```
# 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)
```

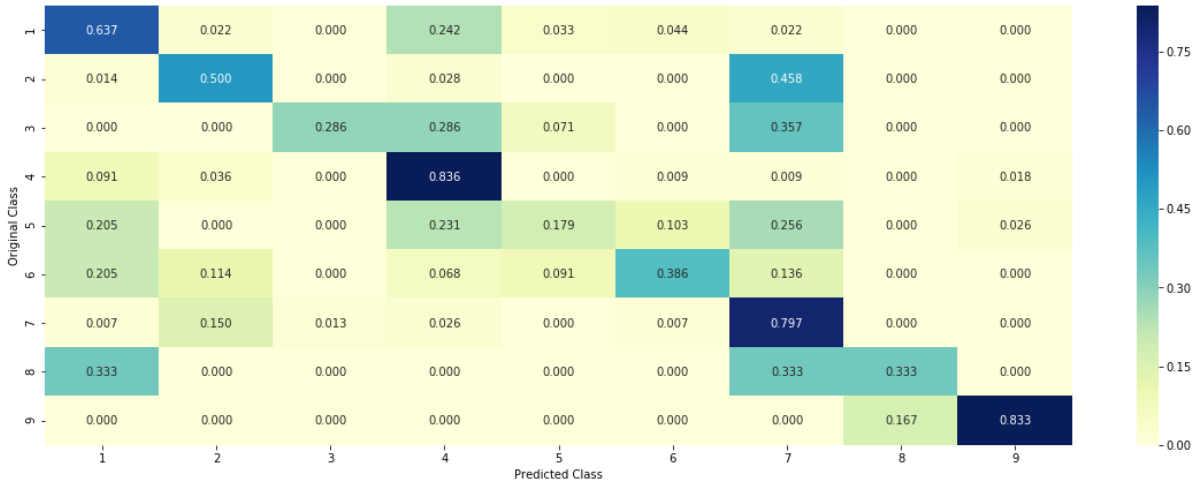
```
Log loss : 1.002444969002307
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [85]:
```
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
 to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 1
The  15  nearest neighbours of the test points belongs to classes [1 1 1 1 4
4 4 4 1 1 1 1 1 1 1]
Fequency of nearest points : Counter({1: 11, 4: 4})
```

4.2.4. Sample Query Point-2

In [86]:
```
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].reshap
e(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 o
f the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 2
the k value for knn is 15 and the nearest neighbours of the test points belon
gs to classes [2 2 2 2 2 2 2 1 2 2 7 2 2 8 2]
Fequency of nearest points : Counter({2: 12, 1: 1, 7: 1, 8: 1})
```

4.3. Logistic Regression

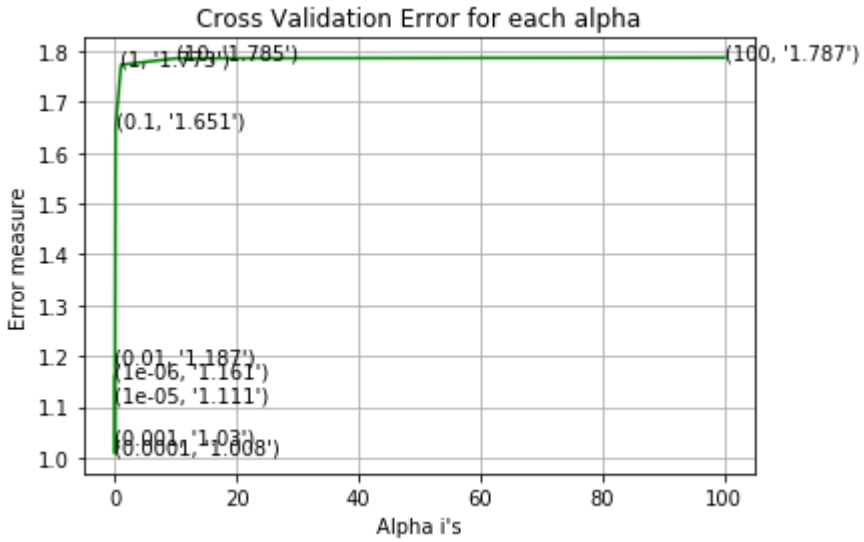
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [87]:
```
# 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], penal
ty='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)

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 validati
on 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 = ', 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 : 1.1606135137247384
for alpha = 1e-05
Log Loss : 1.1114105288260316
for alpha = 0.0001
Log Loss : 1.0080198815848118
for alpha = 0.001
Log Loss : 1.030198911248978
for alpha = 0.01
Log Loss : 1.1871596843655539
for alpha = 0.1
Log Loss : 1.6508501605551202
for alpha = 1
Log Loss : 1.772762870119923
for alpha = 10
Log Loss : 1.7854754264378672
for alpha = 100
Log Loss : 1.7869227091967115
```

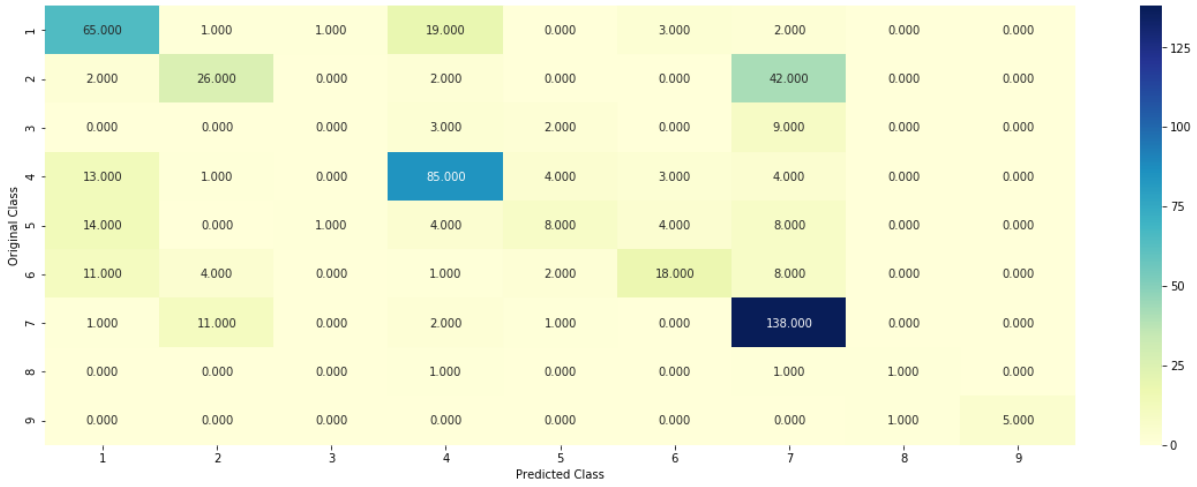



```
For values of best alpha =  0.0001 The train log loss is: 0.45414080100555276
For values of best alpha =  0.0001 The cross validation log loss is: 1.008019
8815848118
For values of best alpha =  0.0001 The test log loss is: 1.085766661069436
```

4.3.1.2. Testing the model with best hyper paramters

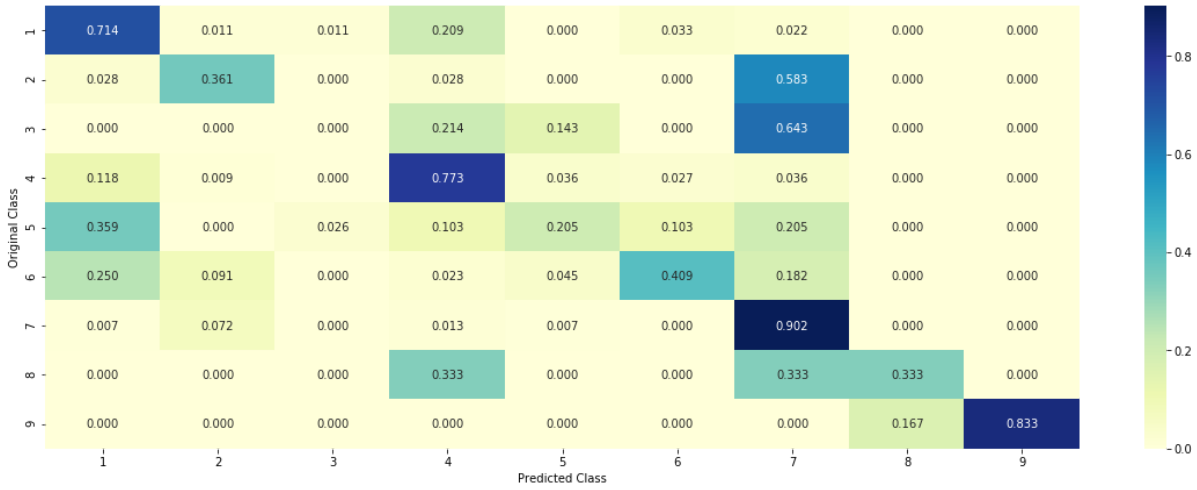
In [88]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, 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 : 1.0080198815848118
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




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


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



4.3.1.3. Feature Importance

In [89]:
```
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], ye
s_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query poin
t")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," cl
ass:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or
 Not']))
```

4.3.1.3.1. Correctly Classified point

In [90]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', 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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[7.133e-01 1.200e-03 7.000e-04 2.500e-01 4.10
0e-03 1.200e-03 2.830e-02
  7.000e-04 4.000e-04]]
Actual Class : 1
--------------------------------------------------
479 Text feature [01] present in test data point [True]
493 Text feature [12] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [91]:
```
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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[3.140e-01 6.162e-01 1.200e-03 3.500e-03 8.50
0e-03 7.800e-03 4.290e-02
  5.600e-03 4.000e-04]]
Actual Class : 2
--------------------------------------------------
205 Text feature [100] present in test data point [True]
256 Text feature [007] present in test data point [True]
498 Text feature [130] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

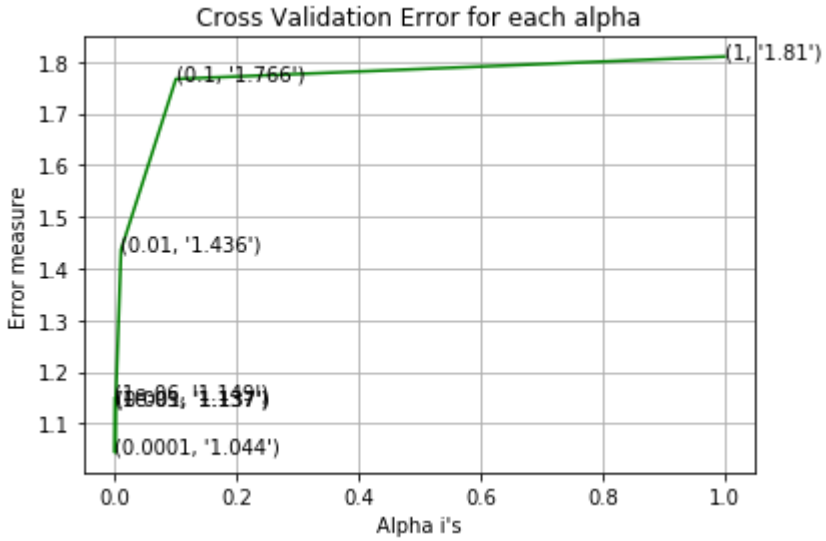
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [92]:
```
# 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', rand
om_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 validati
on 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 = ', 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 : 1.1489024799541627
for alpha = 1e-05
Log Loss : 1.1369452931007258
for alpha = 0.0001
Log Loss : 1.0439245420295757
for alpha = 0.001
Log Loss : 1.137345428345129
for alpha = 0.01
Log Loss : 1.4364673945364395
for alpha = 0.1
Log Loss : 1.7663719394560455
for alpha = 1
Log Loss : 1.809640751267211
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4449326181903318
For values of best alpha =  0.0001 The cross validation log loss is: 1.043924
5420295757
For values of best alpha =  0.0001 The test log loss is: 1.1144306955241083
```

4.3.2.2. Testing model with best hyper parameters

In [93]:
```
# 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 : 1.0439245420295757
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```


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


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

4.3.2.3. Feature Importance, Correctly Classified point

In [94]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', 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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[6.841e-01 1.000e-03 6.000e-04 2.733e-01 3.20
0e-03 1.000e-03 3.660e-02
  1.000e-04 1.000e-04]]
Actual Class : 1
--------------------------------------------------
478 Text feature [12] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [95]:
```
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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[3.297e-01 6.050e-01 1.400e-03 3.800e-03 8.00
0e-03 8.200e-03 4.120e-02
  2.500e-03 1.000e-04]]
Actual Class : 2
--------------------------------------------------
199 Text feature [100] present in test data point [True]
252 Text feature [007] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

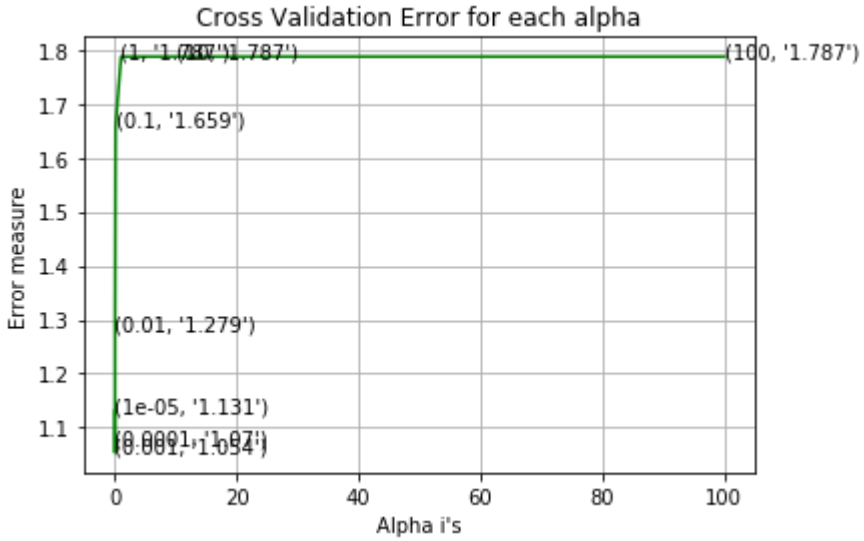
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [96]:
```
# 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, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/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='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(-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='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)
    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], penal
ty='l2', 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 validati
on 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 = ', 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 : 1.1305189006522445
for C = 0.0001
Log Loss : 1.0704526192304331
for C = 0.001
Log Loss : 1.0535154592796154
for C = 0.01
Log Loss : 1.27927436726999
for C = 0.1
Log Loss : 1.659327426281127
for C = 1
Log Loss : 1.7873552689367513
for C = 10
Log Loss : 1.7873964859737401
for C = 100
Log Loss : 1.7873965282492525
```




```
For values of best alpha =  0.001 The train log loss is: 0.6100553816346953
For values of best alpha =  0.001 The cross validation log loss is: 1.0535154
592796154
For values of best alpha =  0.001 The test log loss is: 1.1570737313499444
```

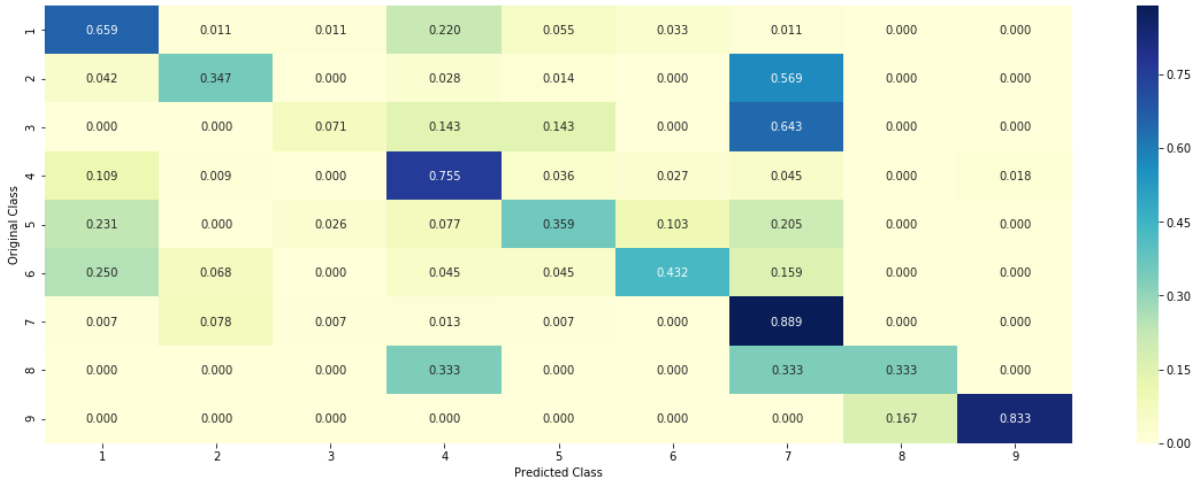
4.4.2. Testing model with best hyper parameters

In [97]:
```
# read more about support vector machines with linear kernals here http://scik
it-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,
 probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_func
tion_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing data.
# predict(X)    Perform classification on samples in X.
# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
lessons/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', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

```
Log loss : 1.0535154592796154
Number of mis-classified points : 0.3533834586466165
-------------------- 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 [98]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6621 0.0038 0.0034 0.2389 0.0105 0.0061 0.
0734 0.0009 0.0011]]
Actual Class : 1
--------------------------------------------------
436 Text feature [12] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [99]:
```
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_
onehotCoding[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.3658 0.4821 0.0096 0.0135 0.0168 0.0172 0.
0887 0.0046 0.0017]]
Actual Class : 2
--------------------------------------------------
195 Text feature [100] present in test data point [True]
380 Text feature [007] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [100]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gin
i', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', ma
x_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, rando
m_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tr
aining 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-onli
ne/lessons/random-forest-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/st
able/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='s
igmoid', 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.
classes_, eps=1e-15))
```

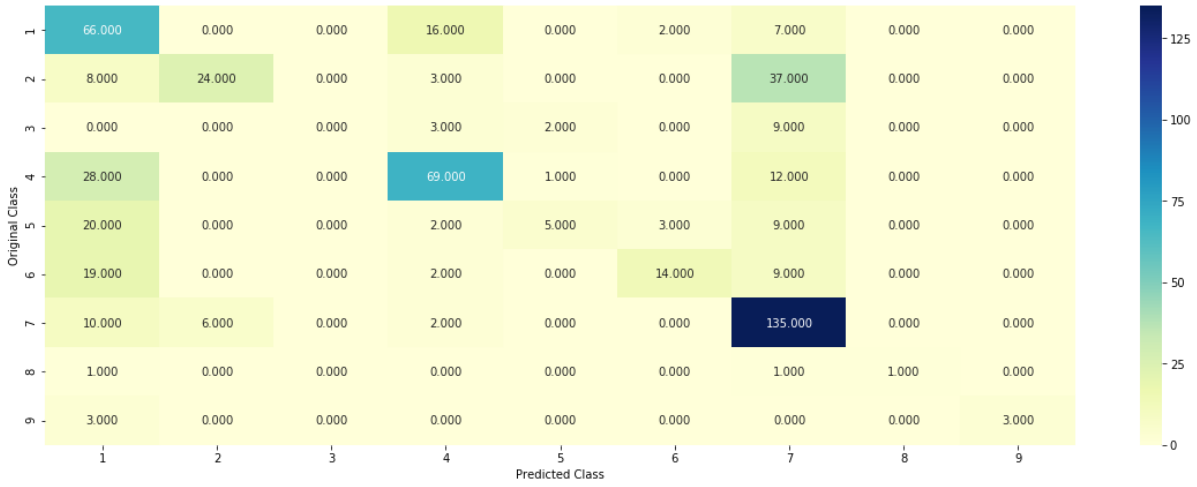
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''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)], criteri
on='gini', 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 tra
in log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-1
5))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cro
ss 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 estimator = ', alpha[int(best_alpha/2)], "The tes
t 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 : 1.2197473597430186
for n_estimators = 100 and max depth =  10
Log Loss : 1.2545533786251462
for n_estimators = 200 and max depth =  5
Log Loss : 1.2115629351379342
for n_estimators = 200 and max depth =  10
Log Loss : 1.245005728263018
for n_estimators = 500 and max depth =  5
Log Loss : 1.2051687599657477
for n_estimators = 500 and max depth =  10
Log Loss : 1.226881295568145
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2024090061217978
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2256288509371314
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1977331525635675
for n_estimators = 2000 and max depth =  10
Log Loss : 1.226825408244335
For values of best estimator =  2000 The train log loss is: 0.862688610335302
1
For values of best estimator =  2000 The cross validation log loss is: 1.1977
331525635675
For values of best estimator =  2000 The test log loss is: 1.2328726593828045
```

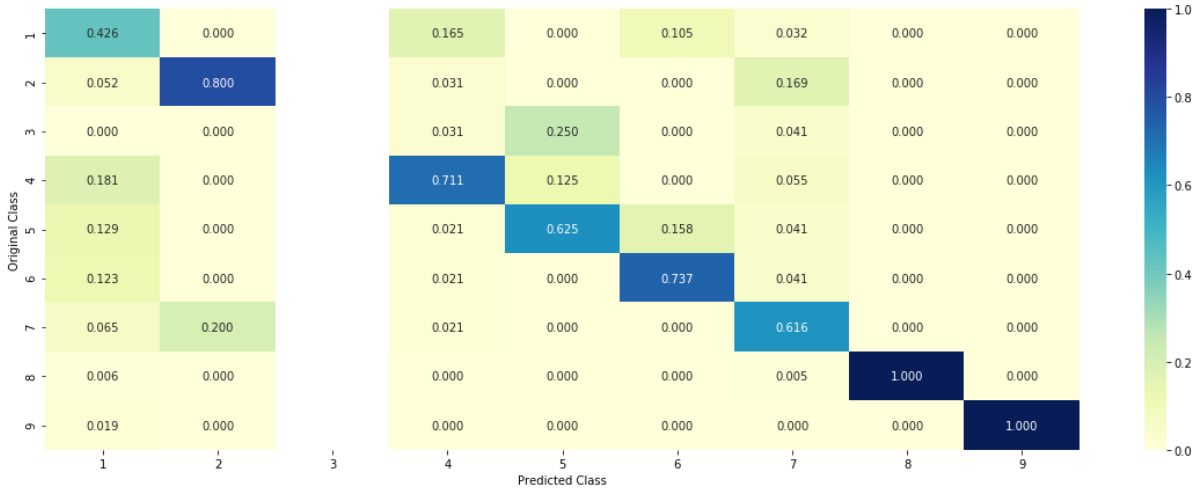
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [101]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gin
i', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', ma
x_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, rando
m_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tr
aining 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-onli
ne/lessons/random-forest-and-their-construction-2/
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criteri
on='gini', 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_onehot
Coding,cv_y, clf)
```

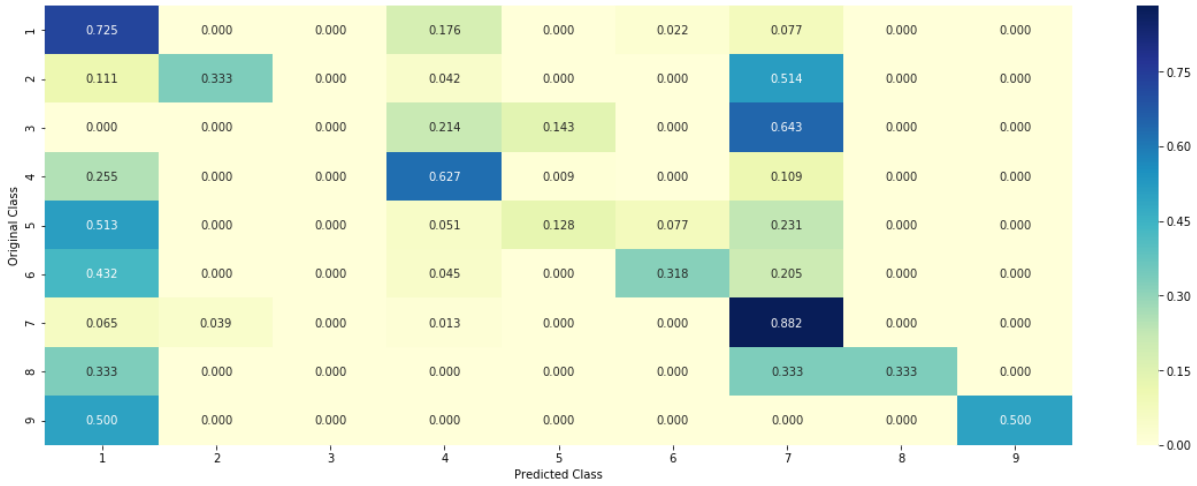
```
Log loss : 1.1977331525635675
Number of mis-classified points : 0.4041353383458647
-------------------- Confusion matrix --------------------
```




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




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [102]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion=
'gini', 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_
onehotCoding[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_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4581 0.0275 0.0117 0.3406 0.0425 0.0389 0.
0542 0.0067 0.0199]]
Actual Class : 1
--------------------------------------------------
51 Text feature [01] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [103]:
```
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_
onehotCoding[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_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.2706 0.3429 0.014  0.0505 0.0491 0.043  0.
1682 0.0533 0.0084]]
Actuall Class : 2
--------------------------------------------------
40 Text feature [016] present in test data point [True]
51 Text feature [01] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [104]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gin
i', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', ma
x_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, rando
m_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given tr
aining 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-onli
ne/lessons/random-forest-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/st
able/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='s
igmoid', 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.
classes_, 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)], criteri
on='gini', 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 l
og 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[int(best_alpha/4)], "The cross v
alidation 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 lo
g loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0739449723694237
for n_estimators = 10 and max depth =  3
Log Loss : 1.917175248913769
for n_estimators = 10 and max depth =  5
Log Loss : 1.4849130320861939
for n_estimators = 10 and max depth =  10
Log Loss : 1.8515036146895019
for n_estimators = 50 and max depth =  2
Log Loss : 1.6980994399320763
for n_estimators = 50 and max depth =  3
Log Loss : 1.4546012684762124
for n_estimators = 50 and max depth =  5
Log Loss : 1.3652066289905578
for n_estimators = 50 and max depth =  10
Log Loss : 1.7137187273288592
for n_estimators = 100 and max depth =  2
Log Loss : 1.5528883159699538
for n_estimators = 100 and max depth =  3
Log Loss : 1.4711142034126652
for n_estimators = 100 and max depth =  5
Log Loss : 1.3394631195447033
for n_estimators = 100 and max depth =  10
Log Loss : 1.6095756674027097
for n_estimators = 200 and max depth =  2
Log Loss : 1.600015478362364
for n_estimators = 200 and max depth =  3
Log Loss : 1.4581907374982428
for n_estimators = 200 and max depth =  5
Log Loss : 1.3939859621476538
for n_estimators = 200 and max depth =  10
Log Loss : 1.5961384874683142
for n_estimators = 500 and max depth =  2
Log Loss : 1.6730148401917242
for n_estimators = 500 and max depth =  3
Log Loss : 1.5441070633752967
for n_estimators = 500 and max depth =  5
Log Loss : 1.3899759720431726
for n_estimators = 500 and max depth =  10
Log Loss : 1.6324415430949262
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6508458879057308
for n_estimators = 1000 and max depth =  3
Log Loss : 1.558795017418978
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3836282668889868
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6108336152167182
For values of best alpha =  100 The train log loss is: 0.05745816213449988
For values of best alpha =  100 The cross validation log loss is: 1.339463119
5447033
For values of best alpha =  100 The test log loss is: 1.4474501410435336
```

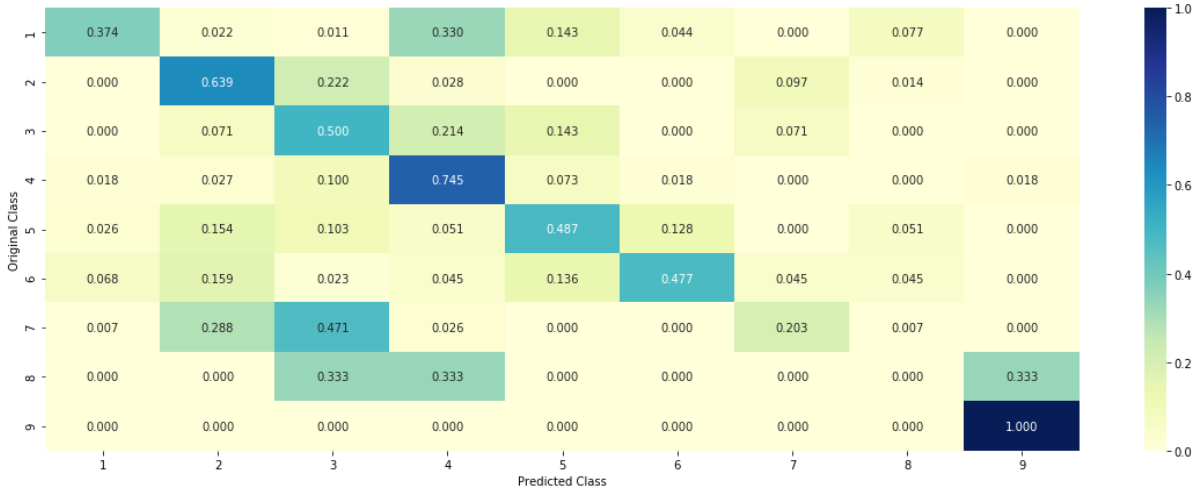
4.5.4. Testing model with best hyper parameters (Response Coding)

In [105]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.3394631195447033
Number of mis-classified points : 0.5375939849624061
-------------------- 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 [106]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion=
'gini', 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].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[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 : 4
Predicted Class Probabilities: [[0.2354 0.0146 0.1212 0.4869 0.0288 0.0353 0.
0085 0.0337 0.0356]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene 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
Gene is important feature
Gene is important feature
Variation is important feature
Text 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
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [107]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[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 : 2
Predicted Class Probabilities: [[0.0336 0.522  0.0793 0.0372 0.0277 0.073  0.
0782 0.0853 0.0638]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene 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
Gene is important feature
Gene is important feature
Variation is important feature
Text 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
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [108]:
```
# 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/
#------------------------------


# 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,
 probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/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_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
```

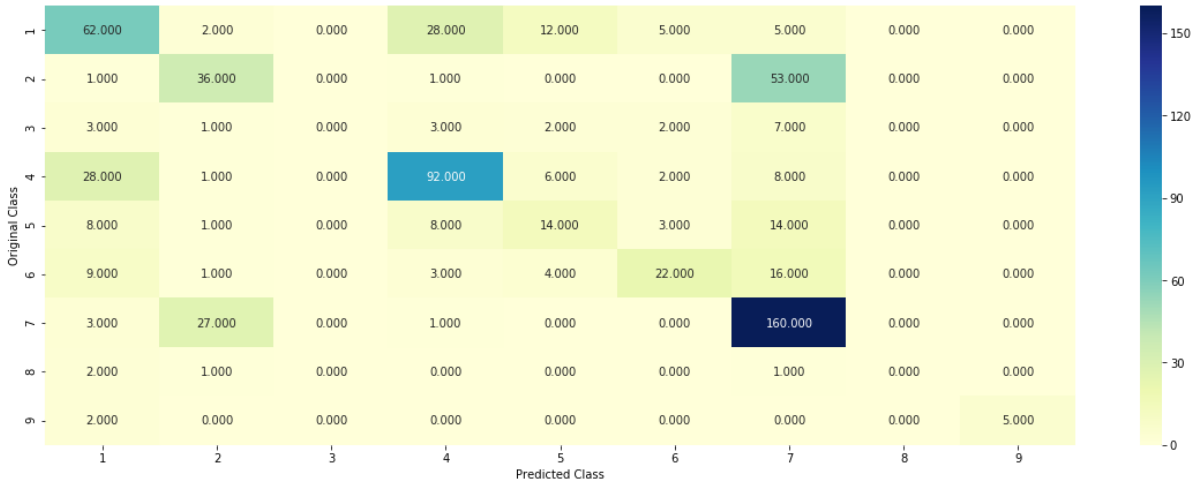
```
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


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: 1.02
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.18
------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.028
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.486
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.154
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.357
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.802
```

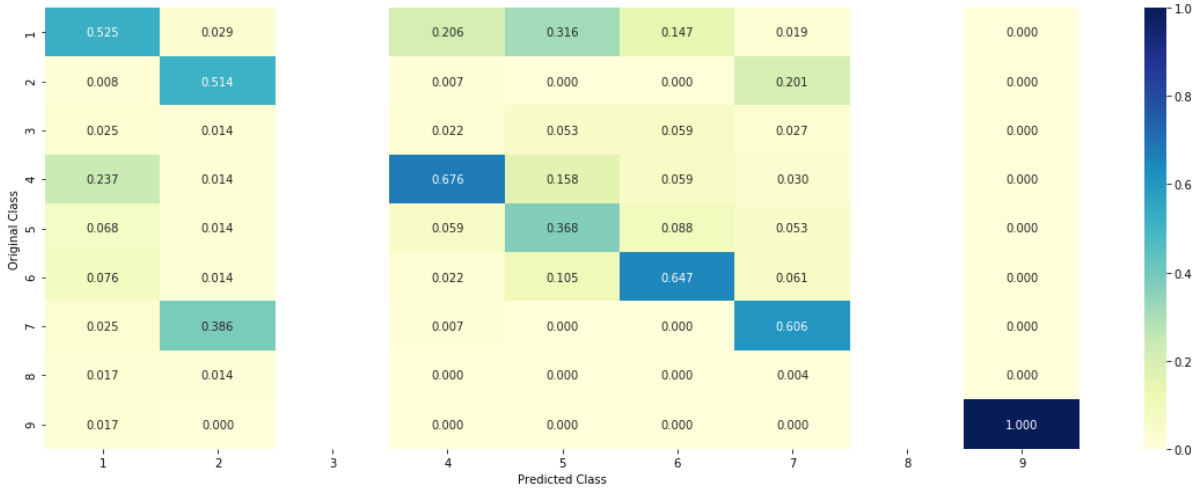
4.7.2 testing the model with the best hyper parameters

In [109]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=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_onehotCodin
g))
```

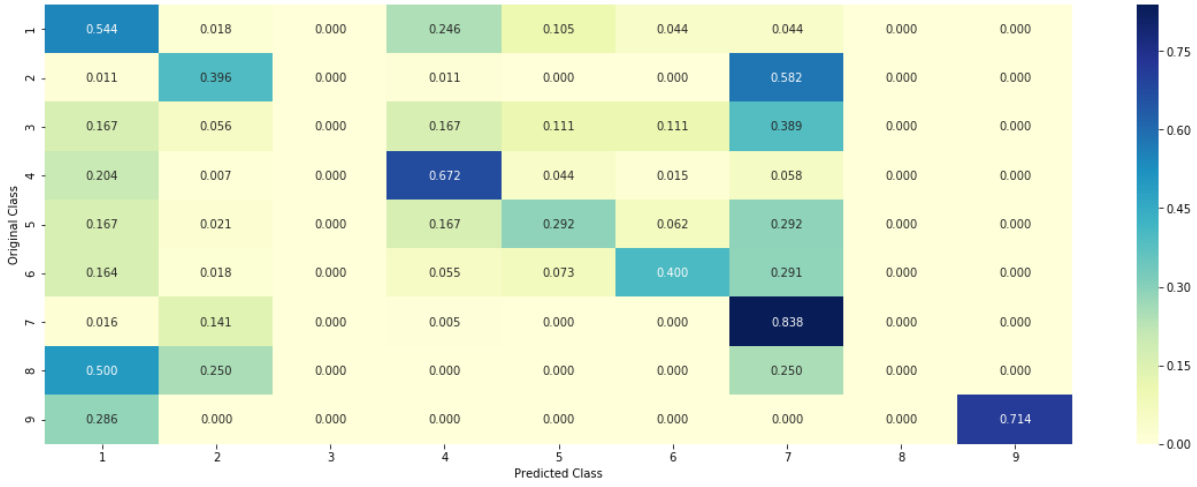
```
Log loss (train) on the stacking classifier : 0.5440122399697161
Log loss (CV) on the stacking classifier : 1.1536881218851502
Log loss (test) on the stacking classifier : 1.2323038106015556
Number of missclassified point : 0.4120300751879699
-------------------- Confusion matrix --------------------
```




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



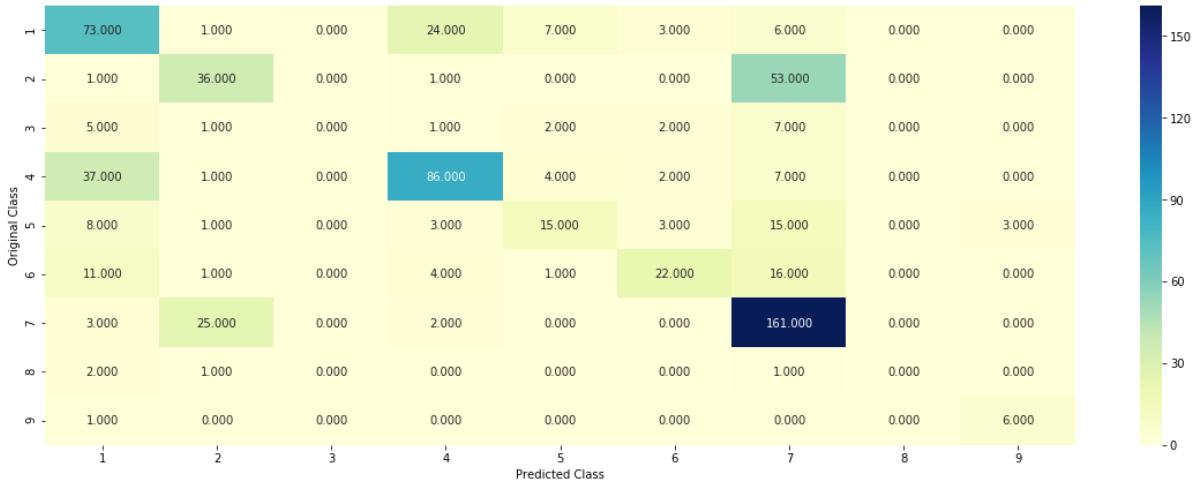

```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.7.3 Maximum Voting classifier

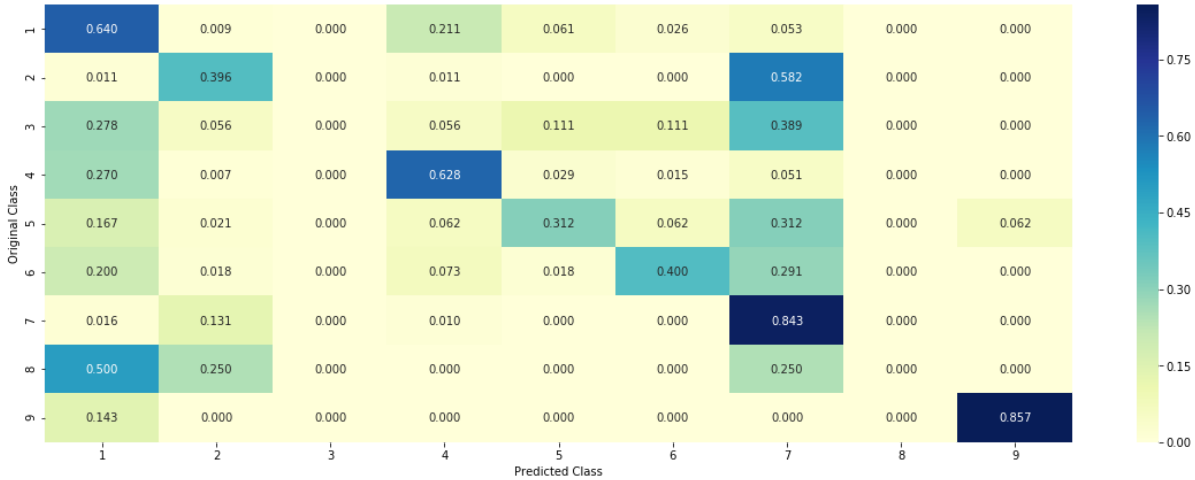
In [110]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
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 : 0.8300230398903303
Log loss (CV) on the VotingClassifier : 1.1831603660033636
Log loss (test) on the VotingClassifier : 1.2441864226666837
Number of missclassified point : 0.4
-------------------- 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

Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [60]:
```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variatio
n'].values;cv_variation=cv_df['Variation'].values
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv
_df['Gene'].values
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv
_df['TEXT'].values

from sklearn.feature_extraction.text import CountVectorizer
encode=CountVectorizer(ngram_range=(1, 2))
train_variation=encode.fit_transform(train_variation);test_variation=encode.tr
ansform(test_variation);cv_variation=encode.transform(cv_variation)
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_ge
ne);cv_gene=encode.transform(cv_gene)
train_variation=normalize(train_variation,axis=0);test_variation=normalize(tes
t_variation,axis=0);cv_variation=normalize(cv_variation,axis=0);
train_gene=normalize(train_gene,axis=0);test_gene=normalize(test_gene,axis=0);
cv_gene=normalize(cv_gene,axis=0);
print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_te
xt);cv_text=encode.transform(cv_text)
train_text=normalize(train_text,axis=0);test_text=normalize(test_text,axis=0);
cv_text=normalize(cv_text,axis=0)
print(train_text.shape)
```

```
(2124, 236)
  (0, 176)      0.23570226039551587
(2124, 2058)
  (0, 1081)     1.0
(2124, 2373961)
```

In [61]:
```
from scipy.sparse import hstack
print(train_variation.shape,train_gene.shape,train_text.shape)
train_data=hstack([train_variation,train_gene,train_text]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text]).tocsr()
test_data=hstack([test_variation,test_gene,test_text]).tocsr()
print(train_data.shape,cv_data.shape,test_data.shape)
train_y = y_test
test_y = y_test
cv_y = y_cv
```

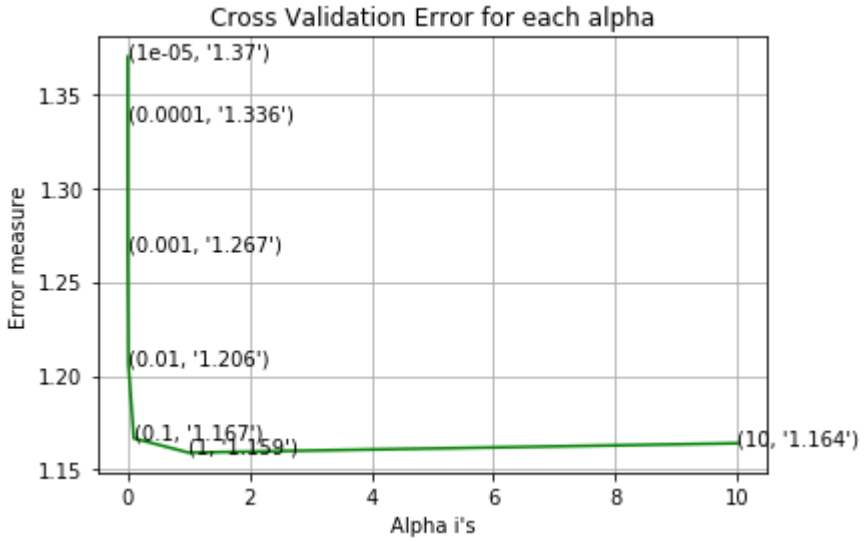
```
(2124, 2058) (2124, 236) (2124, 2373961)
(2124, 2376255) (532, 2376255) (665, 2376255)
```

In [63]:
```
cv_scores=[]
alpha=[10 ** x for x in range(-5, 2)]
for c in alpha:
    print("for alpha =", c)
    lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_jobs
=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_op=clf.predict_proba(cv_data)
    print(c,log_loss(y_cv, cv_op))
    cv_scores.append(log_loss(y_cv, cv_op))
```

```
for alpha = 1e-05
1e-05 1.3697693377839029
for alpha = 0.0001
0.0001 1.3360783630381698
for alpha = 0.001
0.001 1.267264941559578
for alpha = 0.01
0.01 1.2063816360415758
for alpha = 0.1
0.1 1.1667435762242009
for alpha = 1
1 1.1591281398367685
for alpha = 10
10 1.1641198594655746
```

In [64]:
```
print(alpha,cv_scores)
print(len(alpha),len(cv_scores))
print(type(cv_scores))
fig, ax = plt.subplots()
ax.plot(alpha, cv_scores,c='g')
for i, txt in enumerate(np.round(cv_scores,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_scores[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10] [1.3697693377839029, 1.3360783630381
698, 1.267264941559578, 1.2063816360415758, 1.1667435762242009, 1.15912813983
67685, 1.1641198594655746]
7 7
<class 'list'>
```



In [66]:
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1
)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
print("Log Loss value for the test data is(10) ",log_loss(y_test, test_op))
```

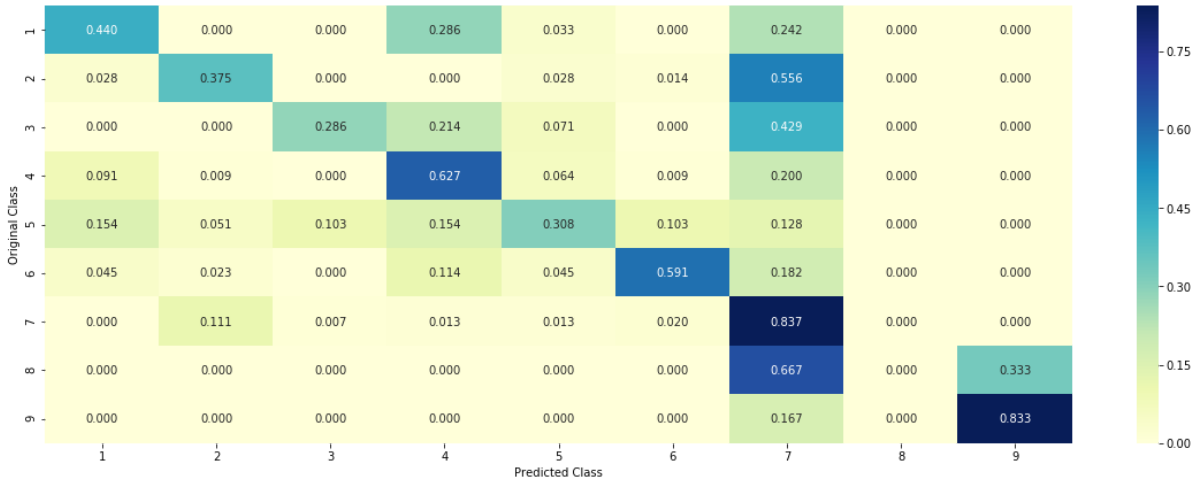
```
Log Loss value for the test data is(10)  1.1820860678013823
```

In [68]:
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1
)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
predict_and_plot_confusion_matrix(train_data, y_train, cv_data,y_cv, clf)
```

```
Log loss : 1.1528530709437153
Number of mis-classified points : 0.41541353383458646
-------------------- Confusion matrix --------------------
```


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


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



Logistic Regression with tf-idf vectorizer and top 1000 max features.

In [117]:
```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variatio
n'].values;cv_variation=cv_df['Variation'].values
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv
_df['Gene'].values
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv
_df['TEXT'].values

encode=TfidfVectorizer(max_features=1000)
train_variation=encode.fit_transform(train_variation);test_variation=encode.tr
ansform(test_variation);cv_variation=encode.transform(cv_variation)
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_ge
ne);cv_gene=encode.transform(cv_gene)
train_variation=normalize(train_variation,axis=0);test_variation=normalize(tes
t_variation,axis=0);cv_variation=normalize(cv_variation,axis=0);
train_gene=normalize(train_gene,axis=0);test_gene=normalize(test_gene,axis=0);
cv_gene=normalize(cv_gene,axis=0);
print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_te
xt);cv_text=encode.transform(cv_text)
train_text=normalize(train_text,axis=0);test_text=normalize(test_text,axis=0);
cv_text=normalize(cv_text,axis=0)
print(train_text.shape)
```

```
(2124, 236)
  (0, 176)      0.23570226039551587
(2124, 1000)
  (0, 63)       1.0
(2124, 1000)
```

In [118]:
```
from scipy.sparse import hstack
print(train_variation.shape,train_gene.shape,train_text.shape)
train_data=hstack([train_variation,train_gene,train_text]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text]).tocsr()
test_data=hstack([test_variation,test_gene,test_text]).tocsr()
print(train_data.shape,cv_data.shape,test_data.shape)
train_y = y_test
test_y = y_test
cv_y = y_cv
```

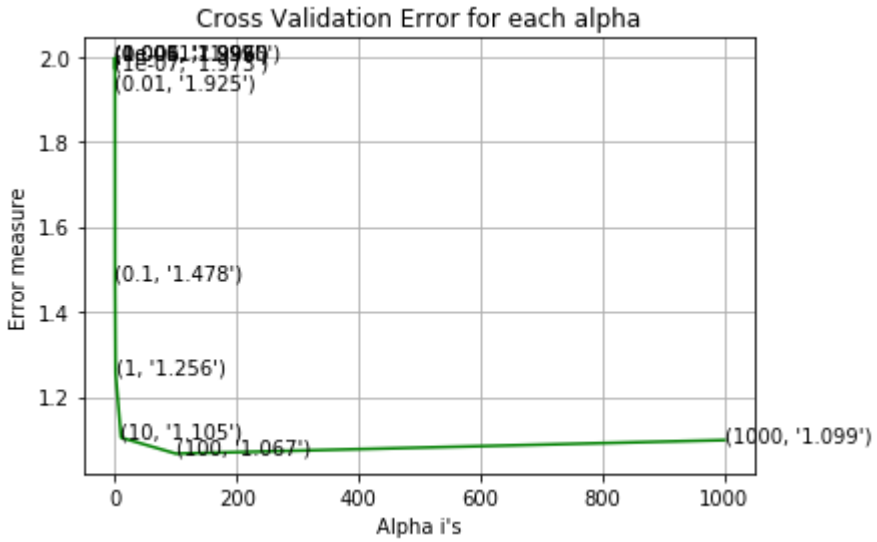
```
(2124, 1000) (2124, 236) (2124, 1000)
(2124, 2236) (532, 2236) (665, 2236)
```

In [119]:
```
cv_scores=[]
alpha=[10 ** x for x in range(-7, 4)]
for c in alpha:
    print("for alpha =", c)
    lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_jobs
=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_op=clf.predict_proba(cv_data)
    print(c,log_loss(y_cv, cv_op))
    cv_scores.append(log_loss(y_cv, cv_op))
```

```
for alpha = 1e-07
1e-07 1.9726712835140503
for alpha = 1e-06
1e-06 1.995957643382
for alpha = 1e-05
1e-05 1.995996494451589
for alpha = 0.0001
0.0001 1.996034427257036
for alpha = 0.001
0.001 1.9973098010071164
for alpha = 0.01
0.01 1.9247563289266012
for alpha = 0.1
0.1 1.4775939654330232
for alpha = 1
1 1.256027520547887
for alpha = 10
10 1.104602796438512
for alpha = 100
100 1.0668936060841459
for alpha = 1000
1000 1.0986756638305286
```

In [120]:
```
print(alpha,cv_scores)
print(len(alpha),len(cv_scores))
print(type(cv_scores))
fig, ax = plt.subplots()
ax.plot(alpha, cv_scores,c='g')
for i, txt in enumerate(np.round(cv_scores,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_scores[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
[1e-07, 1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000] [1.97267128
35140503, 1.995957643382, 1.995996494451589, 1.996034427257036, 1.99730980100
71164, 1.9247563289266012, 1.4775939654330232, 1.256027520547887, 1.104602796
438512, 1.0668936060841459, 1.0986756638305286]
11 11
<class 'list'>
```

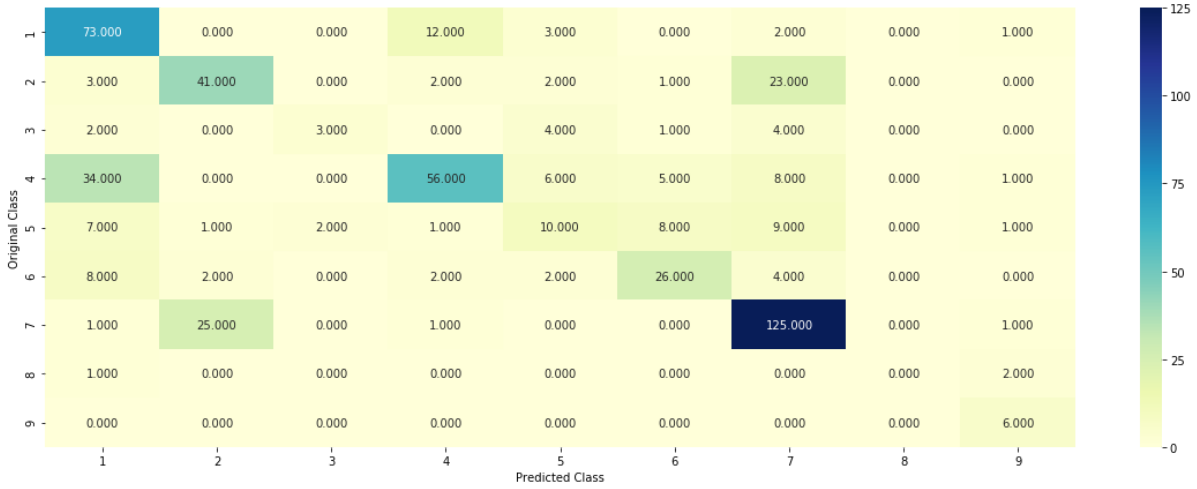


In [121]:
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1
)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
print("Log Loss value for the test data is(10) ",log_loss(y_test, test_op))
```

```
Log Loss value for the test data is(10)  1.0486989424144908
```

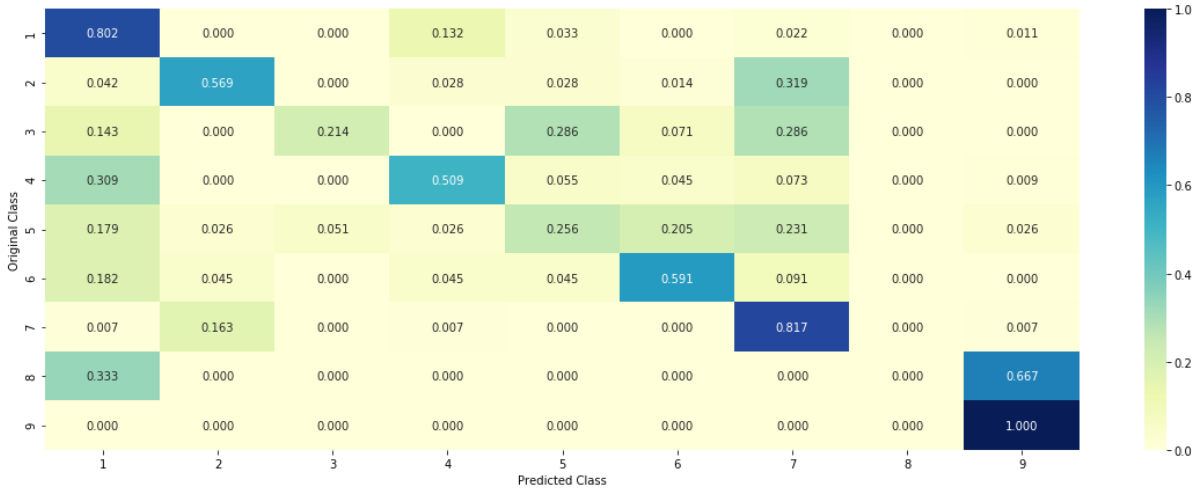
In [74]:
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1
)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
predict_and_plot_confusion_matrix(train_data, y_train, cv_data,y_cv, clf)
```

```
Log loss : 1.0972874415539198
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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


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



Logistic Regression with (1-4)ngrams and top 2000 features

In [101]:
```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variatio
n'].values;cv_variation=cv_df['Variation'].values
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv
_df['Gene'].values
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv
_df['TEXT'].values

encode=TfidfVectorizer(ngram_range=(1, 4),max_features=2000)
train_variation=encode.fit_transform(train_variation);test_variation=encode.tr
ansform(test_variation);cv_variation=encode.transform(cv_variation)
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_ge
ne);cv_gene=encode.transform(cv_gene)
train_variation=normalize(train_variation,axis=0);test_variation=normalize(tes
t_variation,axis=0);cv_variation=normalize(cv_variation,axis=0);
train_gene=normalize(train_gene,axis=0);test_gene=normalize(test_gene,axis=0);
cv_gene=normalize(cv_gene,axis=0);
print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_te
xt);cv_text=encode.transform(cv_text)
train_text=normalize(train_text,axis=0);test_text=normalize(test_text,axis=0);
cv_text=normalize(cv_text,axis=0)
print(train_text.shape)
```

```
(2124, 236)
  (0, 176)      0.23570226039551587
(2124, 2000)
  (0, 1022)     1.0
(2124, 2000)
```

In [102]:
```
from scipy.sparse import hstack
print(train_variation.shape,train_gene.shape,train_text.shape)
train_data=hstack([train_variation,train_gene,train_text]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text]).tocsr()
test_data=hstack([test_variation,test_gene,test_text]).tocsr()
print(train_data.shape,cv_data.shape,test_data.shape)
train_y = y_test
test_y = y_test
cv_y = y_cv
```

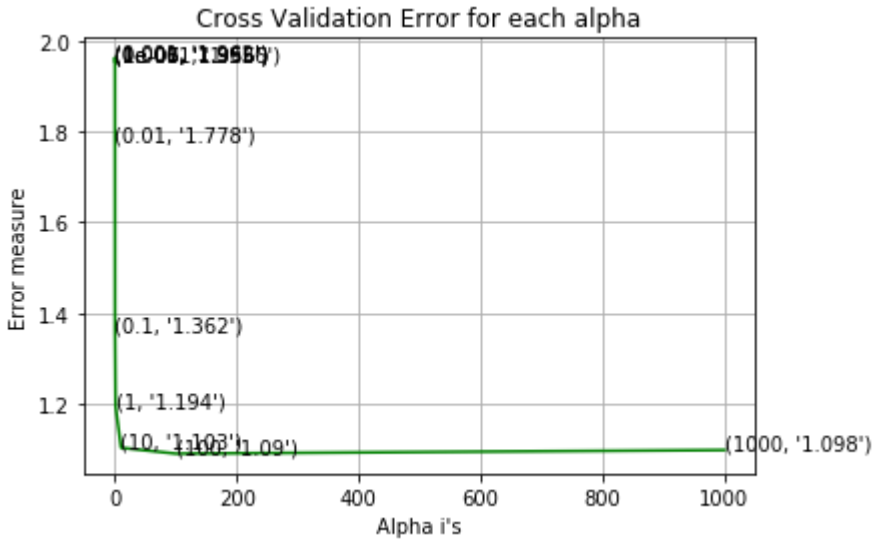
```
(2124, 2000) (2124, 236) (2124, 2000)
(2124, 4236) (532, 4236) (665, 4236)
```

In [113]:
```
cv_scores=[]
alpha=[10 ** x for x in range(-7, 4)]
for c in alpha:
    print("for alpha =", c)
    lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_j
obs=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_op=clf.predict_proba(cv_data)
    print(c,log_loss(y_cv, cv_op))
    cv_scores.append(log_loss(y_cv, cv_op))
```

```
for alpha = 1e-07
1e-07 1.9546053755901789
for alpha = 1e-06
1e-06 1.9553058499034912
for alpha = 1e-05
1e-05 1.9557262289822985
for alpha = 0.0001
0.0001 1.9564280674572203
for alpha = 0.001
0.001 1.9623893909124739
for alpha = 0.01
0.01 1.7784731918008843
for alpha = 0.1
0.1 1.3615658299544349
for alpha = 1
1 1.1943980965018823
for alpha = 10
10 1.1029507632799167
for alpha = 100
100 1.0899707293005236
for alpha = 1000
1000 1.0978931968556114
```

In [114]:
```
print(alpha,cv_scores)
print(len(alpha),len(cv_scores))
print(type(cv_scores))
fig, ax = plt.subplots()
ax.plot(alpha, cv_scores,c='g')
for i, txt in enumerate(np.round(cv_scores,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_scores[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
[1e-07, 1e-06, 1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000] [1.95460537
55901789, 1.9553058499034912, 1.9557262289822985, 1.9564280674572203, 1.96238
93909124739, 1.7784731918008843, 1.3615658299544349, 1.1943980965018823, 1.10
29507632799167, 1.0899707293005236, 1.0978931968556114]
11 11
<class 'list'>
```

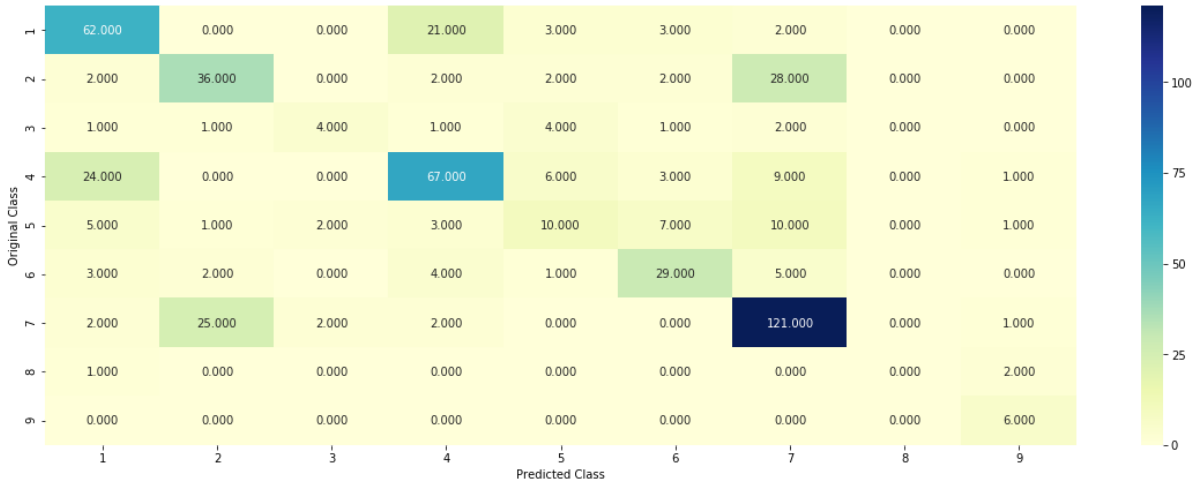


In [115]:
```
lr = LogisticRegression(random_state=0, C=100,class_weight='balanced',n_jobs=-
1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
print("Log Loss value for the test data is(10) ",log_loss(y_test, test_op))
```

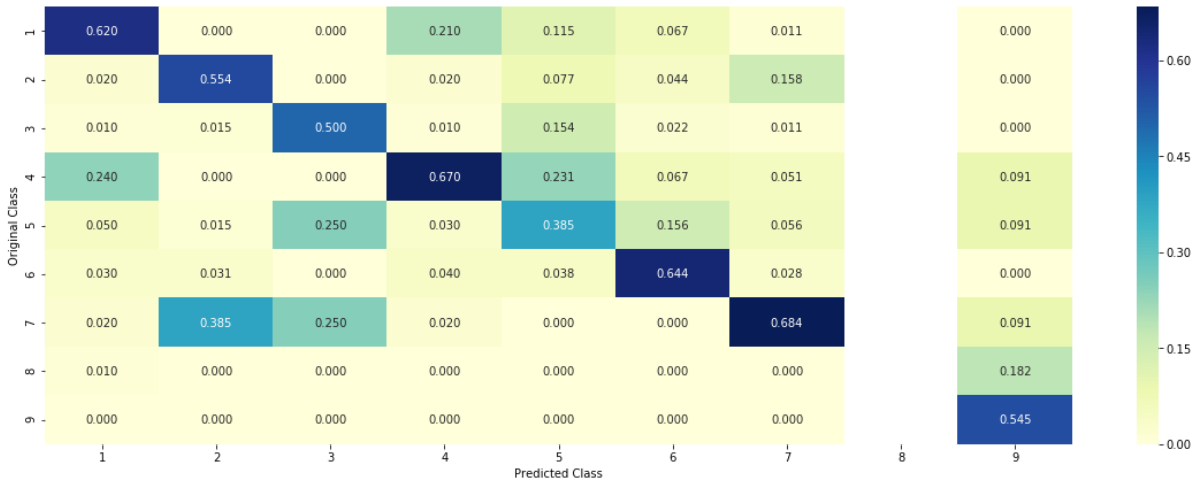
```
Log Loss value for the test data is(10)  0.9968369435179198
```

In [116]:
```
lr = LogisticRegression(random_state=0, C=100,class_weight='balanced',n_jobs=-
1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
predict_and_plot_confusion_matrix(train_data, y_train, cv_data,y_cv, clf)
```

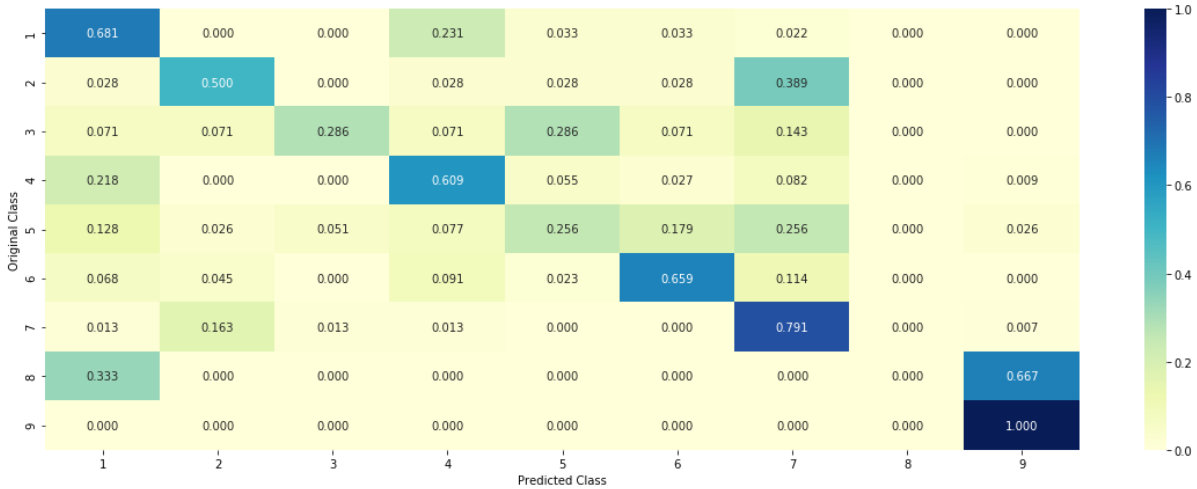
```
Log loss : 1.089630632796889
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




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




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



Stacking 3 features and using (1-4) ngrams with 2000 max_features in TF-IDF and using Logistic Regression

In [91]:
```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train, test_size=0.2)
```

In [92]:
```
def get_gv_fea_dict(alpha, feature, df):
    value_count = x_train[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    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])
    return gv_fea
```

In [93]:
```
alpha = 1

# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_t
rain))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_te
st))

# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_cv))
```

In [94]:
```
gene_vectorizer = TfidfVectorizer(ngram_range=(1, 3),max_features=2000)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'
])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

In [95]:
```
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Varia
tion", x_train))

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variat
ion", x_test))

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
n", x_cv))
```

In [96]:
```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_tr
ain['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['V
ariation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Varia
tion'])
```

In [97]:
```
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


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_di
ct.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(
row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [98]:
```
text_vectorizer = TfidfVectorizer(ngram_range=(1, 3),max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['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 return
s (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
 times 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 : 2000
```

In [99]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['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(x_train)


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 [100]:
```
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# 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
_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [101]:
```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['TEXT'])
```

In [102]:
```
gene_variation = []

for gene in data['Gene'].values:
    gene_variation.append(gene)

for variation in data['Variation'].values:
    gene_variation.append(variation)
```

In [103]:
```
tfidfVectorizer = TfidfVectorizer(ngram_range=(1, 3),max_features=2000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
cv_text = tfidfVectorizer.transform(x_cv['TEXT'])
```

In [104]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train
_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_va
riation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variatio
n_feature_onehotCoding))

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_one
hotCoding)).tocsr()
train_y = np.array(list(x_train['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehot
Coding)).tocsr()
test_y = np.array(list(x_test['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding
)).tocsr()
cv_y = np.array(list(x_cv['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCodin
g,train_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_v
ariation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_te
xt_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_
feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_featur
e_responseCoding))
```

In [105]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_onehotCoding.shape)
```

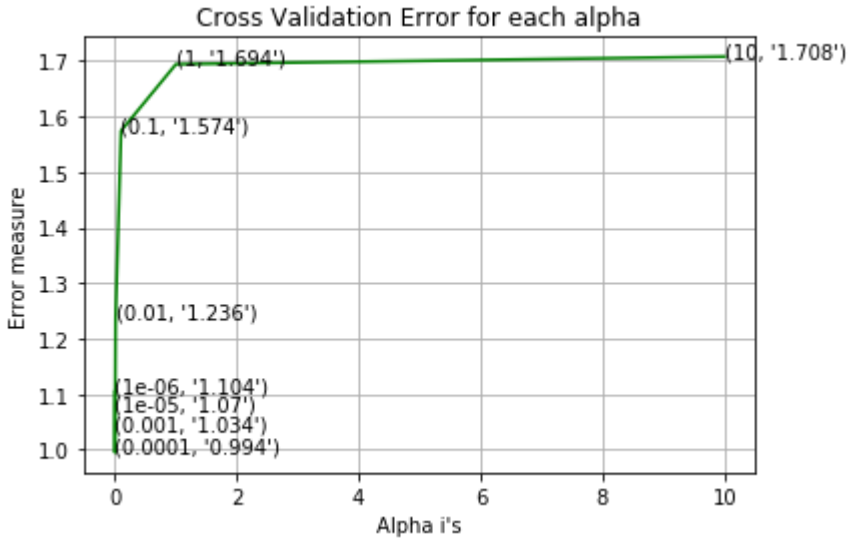
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 6188)
(number of data points * number of features) in test data =  (665, 6188)
(number of data points * number of features) in cross validation data = (532,
6188)
```

In [106]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", trai
n_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_
x_responseCoding.shape)
print("(number of data points * number of features) in cross validation dat
a =", 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, 27)
(number of data points * number of features) in cross validation data = (532,
27)
```

In [107]:
```
alpha = [10 ** x for x in range(-6, 2)]
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=41)
    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-probab
ility 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=
'l2', loss='log',)
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 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 = ',
      alpha[best_alpha], "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

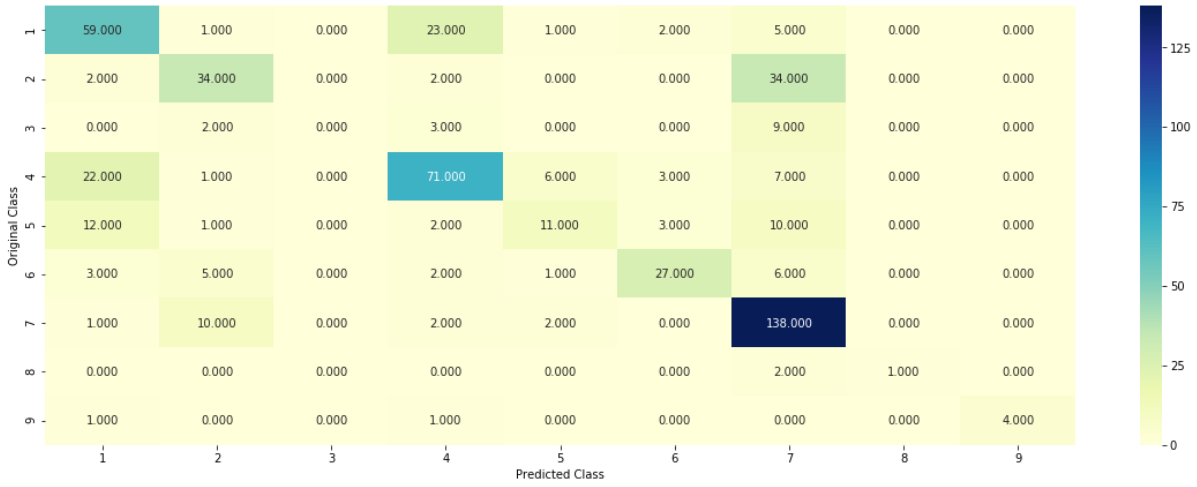
```
for alpha = 1e-06
Log Loss : 1.1043497266852318
for alpha = 1e-05
Log Loss : 1.0701351426809702
for alpha = 0.0001
Log Loss : 0.9944815805808248
for alpha = 0.001
Log Loss : 1.033929211472418
for alpha = 0.01
Log Loss : 1.2364021988820038
for alpha = 0.1
Log Loss : 1.5741336944624982
for alpha = 1
Log Loss : 1.6940041972753461
for alpha = 10
Log Loss : 1.707754812882125
```



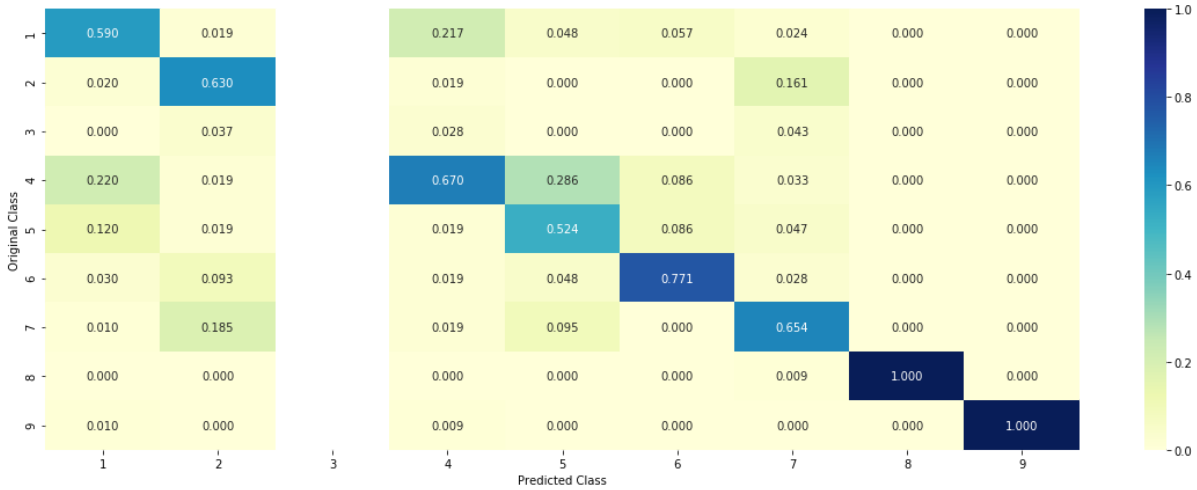

```
For values of best alpha =  0.0001 The train log loss is: 0.47122286805214036
For values of best alpha =  0.0001 The cross validation log loss is: 0.990483
0734953544
For values of best alpha =  0.0001 The test log loss is: 0.9996015940329241
```

In [108]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log',)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCo
ding, cv_y, clf)
```

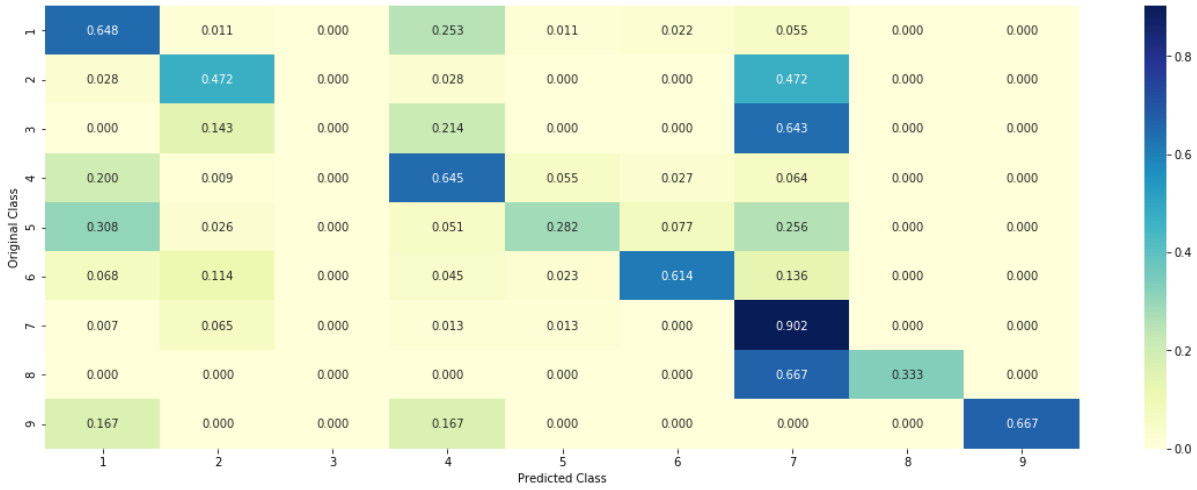
```
Log loss : 0.9991179842743142
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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




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



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [111]:
```
from prettytable import PrettyTable
from termcolor import colored
print(colored('Performance Table of Models with TFIDF vectorizer ','red'))
x = PrettyTable()
x.field_names =["Models","Train","Test","Misclassified(%)"]

x.add_row(["Naive Bayes (One hot coding)",0.52,1.26,0.36])
x.add_row(["KNN (Response)",0.69,1.09,0.35])
x.add_row(["LR(Class balanced) one hot coding",0.45,1.08,0.34])
x.add_row(["LR(Class unbalanced) one hot coding",0.44,1.11,0.35])
x.add_row(["Lr SVM one hot encoding",0.6,1.15,0.35])
x.add_row(["Random Forest one hot coding",0.86,1.23,0.40])
x.add_row(["Random Forest Response coding",0.05,1.44,0.53])
x.add_row(["Stacking classifier",0.54,1.23,0.41])
x.add_row(["Maximum Voting Classifier",0.83,1.24,0.4])
x.add_row(["LR(Class balanced) CountVectorizer",0.45,1.08,0.34])
x.add_row(["LR(Class balanced) TF-IDF Vectorizer and top 1000 max features",1.
09,1.04,0.36])
x.add_row(["LR(Class balanced) TF-IDF Vectorizer and (1-4)ngrams and top 2000
 features",1.08,0.99,0.37])
x.add_row(["Stacking 3 features and using (1-4) ngrams with 2000 max_features
 in TF-IDF \n and using Logistic Regression with class balancing",0.47,0.99,0.
35])
print(x)
print("\n")
```

```
Performance Table of Models with TFIDF vectorizer
+-------------------------------------------------------------------------------
--+-------+------+-----------------+
|                                    Models
  | Train | Test | Misclassified(%) |
+-------------------------------------------------------------------------------
--+-------+------+-----------------+
|                          Naive Bayes (One hot coding)
  |  0.52 | 1.26 |       0.36       |
|                                  KNN (Response)
  |  0.69 | 1.09 |       0.35       |
|                        LR(Class balanced) one hot coding
  |  0.45 | 1.08 |       0.34       |
|                       LR(Class unbalanced) one hot coding
  |  0.44 | 1.11 |       0.35       |
|                           Lr SVM one hot encoding
  |  0.6  | 1.15 |       0.35       |
|                           Random Forest one hot coding
  |  0.86 | 1.23 |       0.4        |
|                           Random Forest Response coding
  |  0.05 | 1.44 |       0.53       |
|                               Stacking classifier
  |  0.54 | 1.23 |       0.41       |
|                             Maximum Voting Classifier
  |  0.83 | 1.24 |       0.4        |
|                        LR(Class balanced) CountVectorizer
  |  0.45 | 1.08 |       0.34       |
|        LR(Class balanced) TF-IDF Vectorizer and top 1000 max features
  |  1.09 | 1.04 |       0.36       |
|  LR(Class balanced) TF-IDF Vectorizer and (1-4)ngrams and top 2000 features
  |  1.08 | 0.99 |       0.37       |
| Stacking 3 features and using (1-4) ngrams with 2000 max_features in TF-IDF
  |  0.47 | 0.99 |       0.35       |
|              and using Logistic Regression with class balancing
  |       |      |                  |
+-------------------------------------------------------------------------------
--+-------+------+-----------------+
```

In [ ]: