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
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. 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
- 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

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]:

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

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import os
print(os.listdir('../input'))
```

```
Using TensorFlow backend.
```

```
['stage2_test_variants.csv', 'test_text', 'stage1_solution_filtered.csv', 'training_variants', 'tr
aining_text', 'stage2_sample_submission.csv', 'stage2_test_text.csv', 'test_variants',
'submissionFile']
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv("../input/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 [5]:

```
# note the seprator in this file
data_text =pd.read_csv("../input/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skip
rows=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[5]:

	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 [6]:

```
#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 : 358.503868 seconds
```

In [7]:

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

Out[7]:

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

In [8]:

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

Out[8]:

	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 [9]:

```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [10]:

```
result[result['ID']==1109]
```

Out[10]:

	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 [15]:

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

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

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

In [31]:

```
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])
```

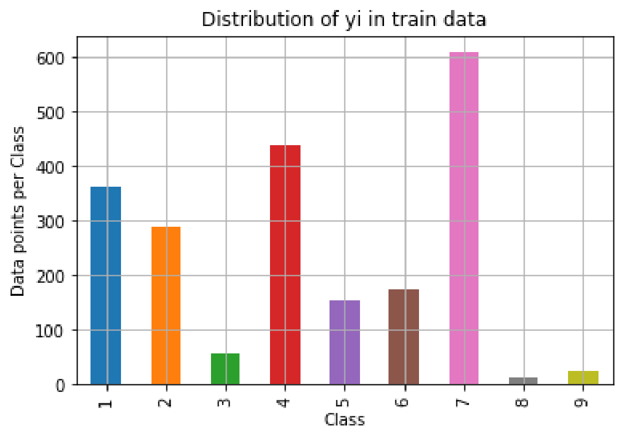
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

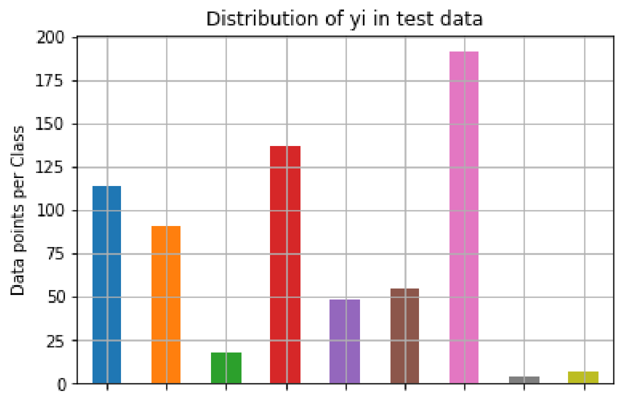
In [32]:

```
# 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().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
```

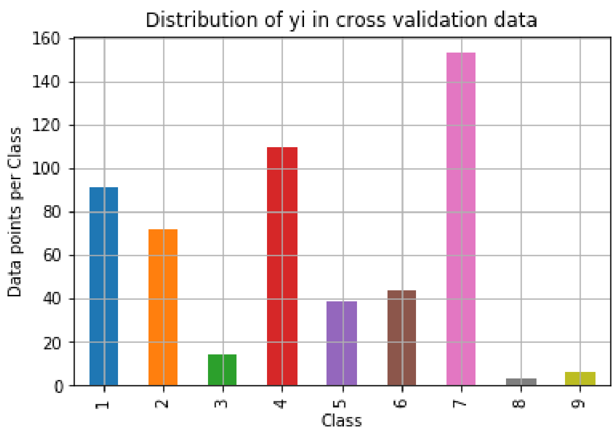
```
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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



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



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



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

from above plots we can see classes 3,9,8 are less distributed. Lets use a random model for preditcion so that it's accuaracy can be used as cutoff to declare model good or not

3.2 Prediction using a 'Random' Model

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

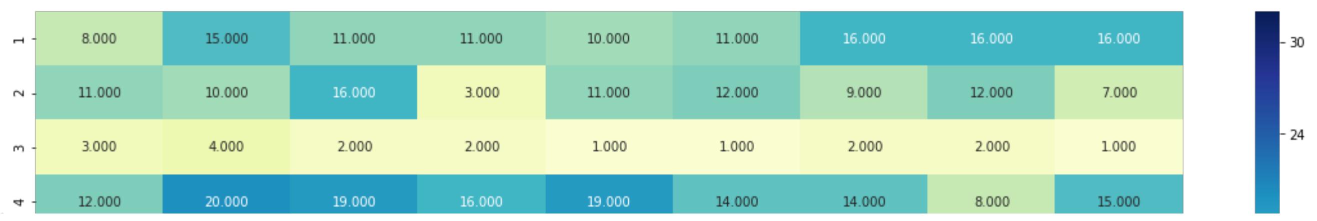
In [33]:

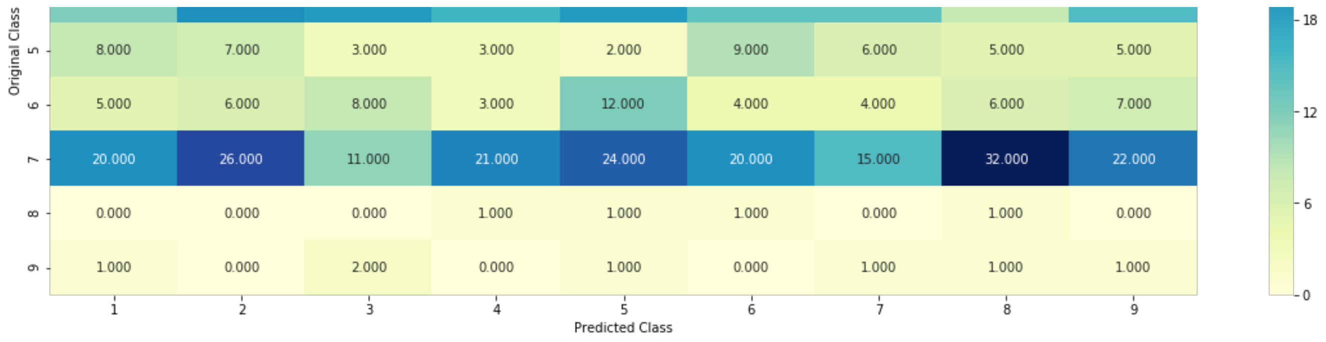
```python
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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 [34]:

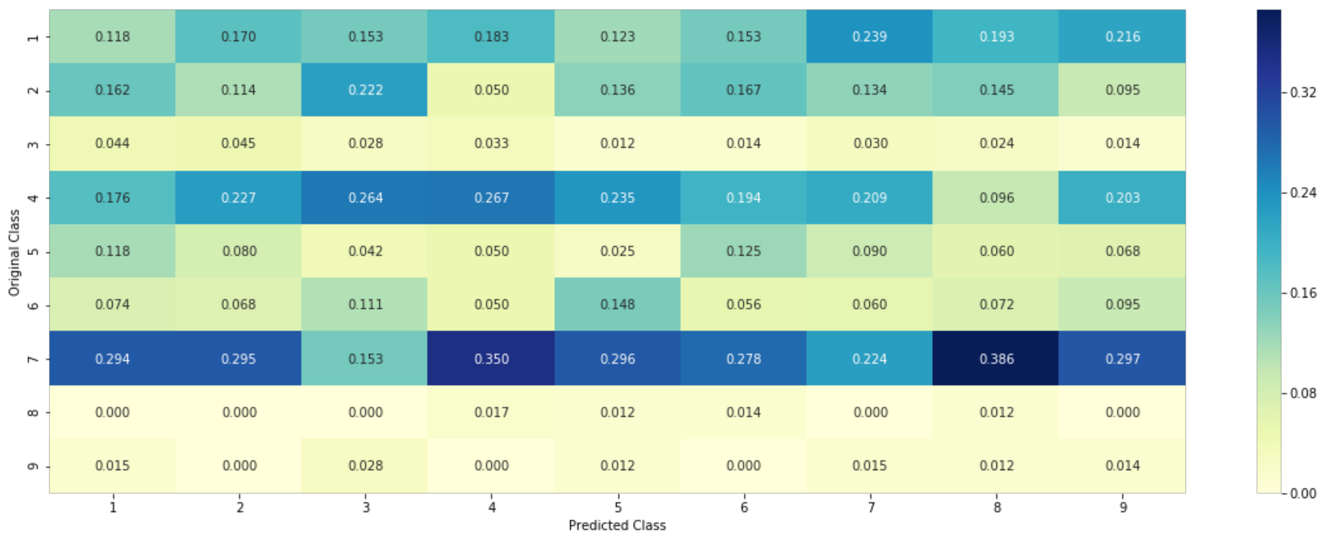
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.4244120324327243
Log loss on Test Data using Random Model 2.4962499542536625
-------------------- Confusion matrix --------------------
```

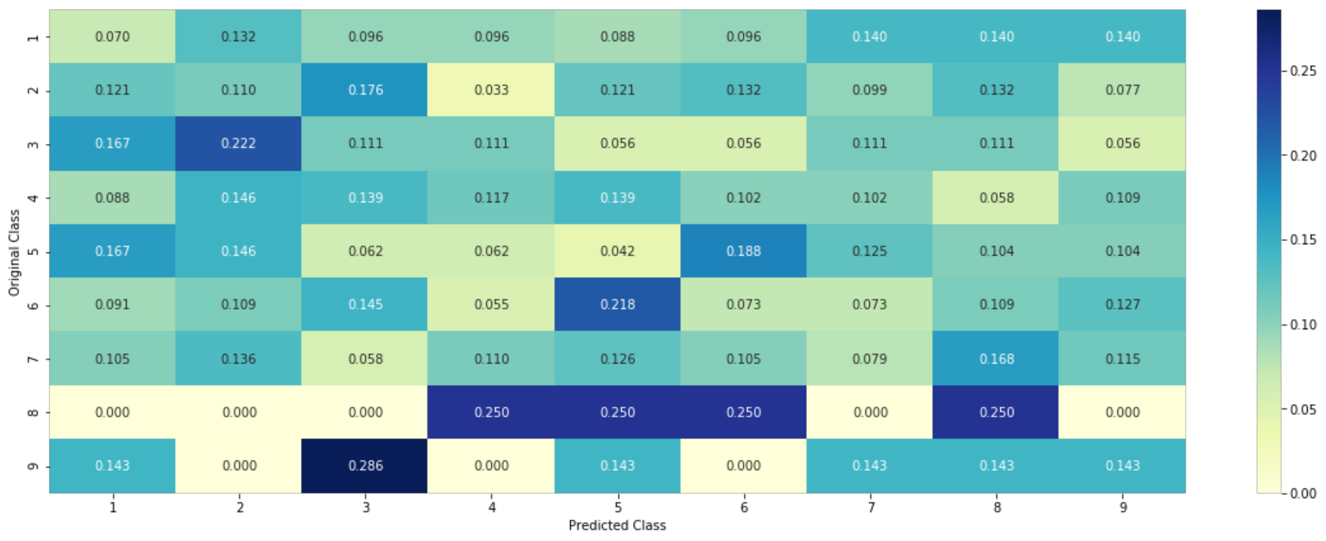




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



In [ ]:

```
we are getting logg loss of 2.5 on test data using random model
```

3.3 Univariate Analysis

In [35]:

```
# 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 / nu
mber 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) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='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[feature]==i)]

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
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 da
ta
    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 [36]:

```
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 : 229
BRCA1     162
TP53       94
BRCA2      86
EGFR       86
PTEN       76
KIT        66
BRAF       60
ALK        49
ERBB2      46
PIK3CA     39
Name: Gene, dtype: int64
```

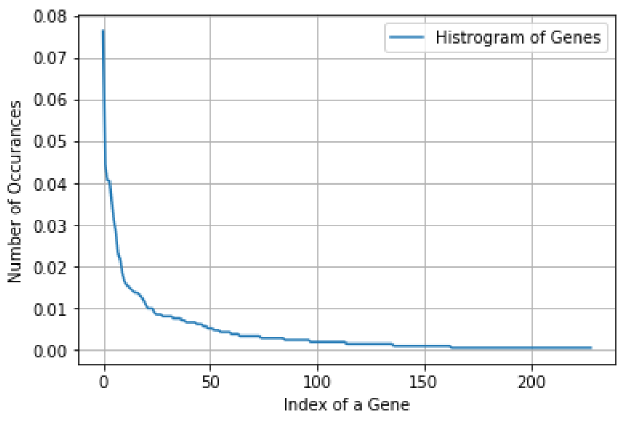
In [37]:

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

```
Ans: There are 229 different categories of genes in the train data, and they are distibuted as fol
lows
```

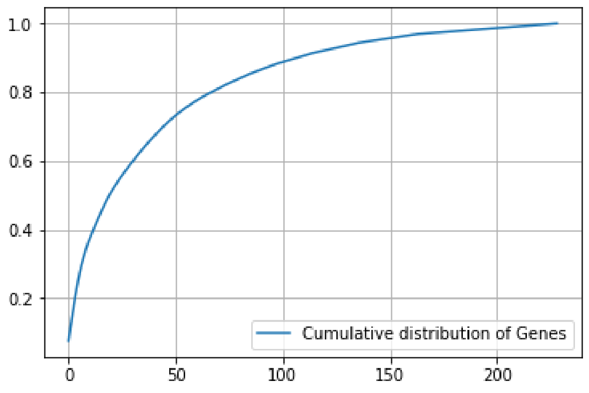
In [38]:

```
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 [39]:

```
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 three 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
3. TFIDF Vectorizer

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 [40]:

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

In [41]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The sha
pe of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [23]:

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

In [42]:

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

Out[42]:

```
1713      POLE
3264       RET
2941    NFKBIA
2544     BRCA1
537      SMAD2
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',
 'asxl1',
 'asxl2',
 'atm',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf3',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'flt1',
 'flt3',
 'foxa1',
 'foxp1',
 'fubp1',
 'gata3',
 'gna11',
 'gnaq',
 'gnas',
 'h3f3a',
 'hist1h1c',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'ikzf1',
 'il7r',
 'inpp4b',
 'jak1',
 'jak2',
 'jun',
 'kdm5a',
 'kdm5c',
 'kdr',
 'keap1',
 'kit',
 'kmt2a',
 'kmt2b',
 'kmt2c',
 'knstrn',
 'kras',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'med12',
 'mef2b',
 'men1',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms2',
 'pole',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [43]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 229)
```

In [44]:

```
# Tfidf of  of Gene feature.
from sklearn.feature_extraction.text import TfidfVectorizer
gene_tfidf = TfidfVectorizer(max_features=1000)
train_gene_feature_tfidf = gene_tfidf.fit_transform(train_df['Gene'])
test_gene_feature_tfidf = gene_tfidf.transform(test_df['Gene'])
cv_gene_feature_tfidf = gene_tfidf.transform(cv_df['Gene'])
```

In [ ]:

```
gene_tfidf.get_feature_names()
```

In [45]:

```
print("train_gene_feature_tfidf is converted feature using tfidf method. The shape of gene
feature:", train_gene_feature_tfidf.shape)
```

```
train_gene_feature_tfidf is converted feature using tfidf method. The shape of gene feature:
(2124, 229)
```

In [47]:

```
gene_vectorizer_bi = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding_bi = gene_vectorizer_bi.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_bi = gene_vectorizer_bi.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_bi = gene_vectorizer_bi.transform(cv_df['Gene'])
```

In [48]:

```
print("train_gene_feature_onehotCoding_bi is converted feature using tfidf method. The shape of ge
ne feature:", train_gene_feature_onehotCoding_bi.shape)
```

```
train_gene_feature_onehotCoding_bi is converted feature using tfidf method. The shape of gene
feature: (2124, 229)
```

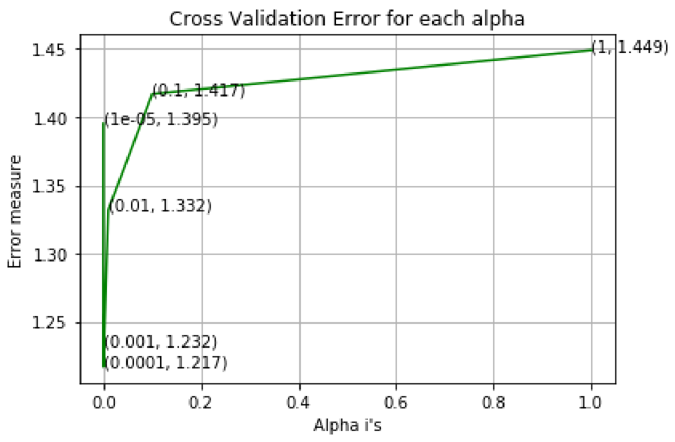
Q4. How good is this gene feature in predicting y_i?

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

In [49]:

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

# 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_i
ter=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:
#------------------------------


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

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.395333211136634
For values of alpha =  0.0001 The log loss is: 1.2166587326334652
For values of alpha =  0.001 The log loss is: 1.2324694684217294
For values of alpha =  0.01 The log loss is: 1.331912935108324
For values of alpha =  0.1 The log loss is: 1.4169189737409058
For values of alpha =  1 The log loss is: 1.449115487792002
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0347687185828929
For values of best alpha =  0.0001 The cross validation log loss is: 1.2166587326334652
For values of best alpha =  0.0001 The test log loss is: 1.2024671012964327
```

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 [50]:

```
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], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  229  genes in train dataset?
Ans
1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 507 out of  532 : 95.30075187969925
```

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 [51]:

```
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 : 1919
Truncating_Mutations    63
Deletion                51
Amplification           39
Fusions                 26
Overexpression           4
E17K                     3
Q61H                     3
T58I                     3
A146V                    2
C618R                    2
Name: Variation, dtype: int64
```

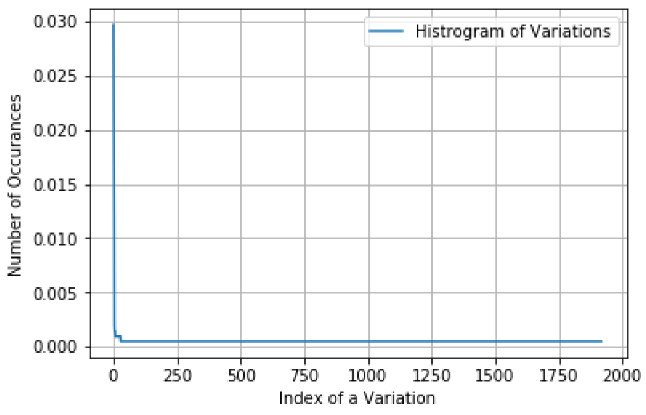
In [52]:

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

```
Ans: There are 1919 different categories of variations in the train data, and they are distibuted
as follows
```

In [53]:

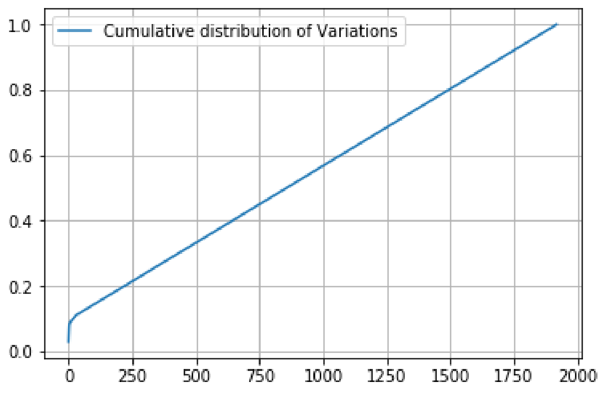
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [54]:

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

```
[0.02966102 0.05367232 0.0720339  ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans. There are three 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
3. Tfidf vectorizer

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

In [55]:

```
# 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 [56]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [25]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [26]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
od. 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, 1950)
```

In [57]:

```
# tfifd of variation feature.
variation_tfidf = TfidfVectorizer(max_features=1000)
train_variation_feature_tfidf = variation_tfidf.fit_transform(train_df['Variation'])
test_variation_feature_tfidf = variation_tfidf.transform(test_df['Variation'])
cv_variation_feature_tfidf = variation_tfidf.transform(cv_df['Variation'])
```

In [58]:

```
print("train_variation_feature_tfidf is converted feature using the tfidf method. The shape of Var
iation feature:", train_variation_feature_tfidf.shape)
```

```
train_variation_feature_tfidf is converted feature using the tfidf method. The shape of Variation
feature: (2124, 1000)
```

In [59]:

```
#bigrams of variation feature.

variation_vectorizer_bi = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding_bi =
variation_vectorizer_bi.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_bi = variation_vectorizer_bi.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_bi = variation_vectorizer_bi.transform(cv_df['Variation'])
```

In [60]:

```
print("train_variation_feature_onehotEncoded_bi is converted feature using bagof words with bigram
method. The shape of Variation feature:", train_variation_feature_onehotCoding_bi.shape)
```

```
train_variation_feature_onehotEncoded_bi is converted feature using bagof words with bigram
method. The shape of Variation feature: (2124, 2055)
```

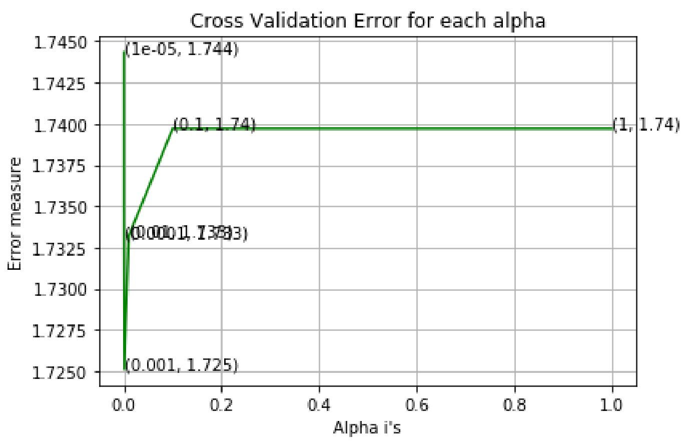
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [61]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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:
#------------------------------


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7443420603532283
For values of alpha =  0.0001 The log loss is: 1.7331468722475414
For values of alpha =  0.001 The log loss is: 1.7250928385483555
For values of alpha =  0.01 The log loss is: 1.7332045811118348
For values of alpha =  0.1 The log loss is: 1.739707472112112
For values of alpha =  1 The log loss is: 1.7397063312335734
```



```
For values of best alpha =  0.001 The train log loss is: 1.3921573421530042
For values of best alpha =  0.001 The cross validation log loss is: 1.7250928385483555
For values of best alpha =  0.001 The test log loss is: 1.7341982225542956
```

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 [62]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
st 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'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q12. How many data points are covered by total  1919  genes in test and cross validation data
sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
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 [70]:

```
# 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 [71]:

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

In [67]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) 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 : 126130
```

In [68]:

```
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 [72]:

```
#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 [74]:

```
# 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_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [75]:

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

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

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

In [76]:

```
#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 [77]:

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

```
Counter({1: 44690, 2: 16768, 3: 8318, 4: 6465, 5: 3923, 6: 3512, 8: 2585, 10: 2263, 7: 2219, 9: 217
2, 12: 1587, 11: 1189, 14: 1111, 15: 952, 13: 912, 16: 908, 18: 878, 17: 795, 20: 761, 21: 608, 24
: 603, 19: 485, 28: 469, 22: 466, 27: 430, 42: 399, 25: 380, 29: 367, 23: 359, 26: 358, 30: 354, 3
6: 341, 32: 328, 47: 294, 33: 279, 40: 271, 34: 251, 31: 246, 37: 245, 35: 221, 48: 220, 38: 219,
39: 213, 54: 190, 50: 179, 43: 176, 45: 174, 41: 174, 44: 172, 51: 168, 60: 159, 49: 155, 46: 149,
55: 148, 52: 148, 67: 143, 57: 141, 53: 137, 72: 132, 56: 128, 70: 126, 66: 124, 58: 117, 63: 116,
61: 115, 68: 113, 59: 110, 62: 99, 64: 97, 69: 95, 84: 90, 76: 90, 65: 90, 90: 87, 71: 86, 74: 80
, 88: 78, 75: 78, 73: 78, 80: 76, 87: 75, 86: 75, 79: 75, 77: 72, 108: 71, 96: 71, 81: 70, 78: 70
, 94: 69, 83: 69, 89: 68, 95: 67, 100: 65, 92: 64, 85: 64, 102: 63, 91: 62, 82: 62, 99: 61, 98: 6
1, 97: 59, 103: 57, 112: 56, 104: 56, 111: 51, 93: 51, 135: 49, 115: 49, 101: 49, 110: 48, 107: 48
, 106: 48, 141: 46, 145: 44, 133: 44, 126: 44, 132: 43, 109: 43, 123: 42, 117: 42, 129: 41, 124: 4
1, 114: 41, 130: 40, 127: 40, 122: 40, 116: 40, 105: 40, 139: 39, 131: 39, 120: 39, 143: 38, 136:
38, 134: 38, 118: 38, 147: 36, 140: 36, 150: 35, 138: 35, 125: 35, 128: 34, 153: 33, 137: 33, 180:
32, 113: 32, 165: 31, 148: 31, 146: 31, 144: 31, 119: 31, 213: 30, 194: 30, 158: 30, 208: 29, 184:
29, 155: 29, 142: 29, 121: 29, 205: 28, 161: 28, 157: 28, 152: 28, 149: 28, 188: 27, 216: 26, 199:
26, 168: 26, 151: 26, 225: 25, 198: 25, 177: 25, 163: 25, 154: 25, 211: 24, 183: 24, 166: 24, 162:
24, 160: 24, 218: 23, 187: 23, 182: 23, 175: 23, 159: 23, 156: 23, 246: 22, 242: 22, 220: 22, 253:
21, 251: 21, 229: 21, 210: 21, 190: 21, 179: 21, 171: 21, 221: 20, 217: 20, 206: 20, 202: 20, 197:
20, 196: 20, 191: 20, 172: 20, 169: 20, 281: 19, 259: 19, 240: 19, 235: 19, 234: 19, 230: 19, 228:
19, 214: 19, 204: 19, 200: 19, 181: 19, 178: 19, 174: 19, 164: 19, 290: 18, 282: 18, 269: 18, 215:
18, 212: 18, 207: 18, 201: 18, 273: 17, 270: 17, 257: 17, 244: 17, 176: 17, 378: 16, 316: 16, 288:
16, 278: 16, 249: 16, 248: 16, 247: 16, 243: 16, 223: 16, 203: 16, 195: 16, 193: 16, 192: 16, 186:
16, 291: 15, 280: 15, 271: 15, 261: 15, 258: 15, 252: 15, 236: 15, 209: 15, 173: 15, 167: 15, 333:
14, 320: 14, 313: 14, 287: 14, 276: 14, 275: 14, 264: 14, 241: 14, 219: 14, 189: 14, 185: 14, 170:
14, 354: 13, 348: 13, 343: 13, 314: 13, 311: 13, 303: 13, 286: 13, 277: 13, 233: 13, 232: 13, 231:
13, 227: 13, 226: 13, 224: 13, 501: 12, 439: 12, 339: 12, 327: 12, 308: 12, 300: 12, 298: 12, 297:
12, 296: 12, 289: 12, 284: 12, 272: 12, 267: 12, 266: 12, 260: 12, 239: 12, 238: 12, 438: 11, 435:
11, 401: 11, 373: 11, 368: 11, 351: 11, 336: 11, 322: 11, 321: 11, 305: 11, 295: 11, 285: 11, 279:
11, 256: 11, 255: 11, 250: 11, 237: 11, 495: 10, 443: 10, 382: 10, 381: 10, 359: 10, 331: 10, 330:
10, 310: 10, 309: 10, 245: 10, 524: 9, 448: 9, 433: 9, 404: 9, 391: 9, 386: 9, 370: 9, 360: 9, 356
: 9, 355: 9, 350: 9, 341: 9, 337: 9, 334: 9, 329: 9, 319: 9, 312: 9, 306: 9, 304: 9, 302: 9, 301:
9, 294: 9, 283: 9, 268: 9, 262: 9, 222: 9, 550: 8, 549: 8, 536: 8, 505: 8, 492: 8, 475: 8, 456: 8,
436: 8, 430: 8, 429: 8, 428: 8, 427: 8, 412: 8, 393: 8, 375: 8, 366: 8, 362: 8, 353: 8, 342: 8,
335: 8, 292: 8, 274: 8, 254: 8, 1194: 7, 839: 7, 719: 7, 708: 7, 576: 7, 538: 7, 511: 7, 485: 7,
479: 7, 476: 7, 467: 7, 462: 7, 459: 7, 458: 7, 454: 7, 441: 7, 431: 7, 422: 7, 419: 7, 417: 7,
415: 7, 414: 7, 410: 7, 408: 7, 407: 7, 400: 7, 397: 7, 394: 7, 385: 7, 384: 7, 383: 7, 377: 7,
358: 7, 352: 7, 347: 7, 340: 7, 328: 7, 325: 7, 318: 7, 293: 7, 265: 7, 1386: 6, 1260: 6, 927: 6,
915: 6, 887: 6, 885: 6, 827: 6, 791: 6, 756: 6, 720: 6, 705: 6, 686: 6, 680: 6, 672: 6, 663: 6,
647: 6, 636: 6, 623: 6, 605: 6, 583: 6, 582: 6, 546: 6, 532: 6, 515: 6, 513: 6, 512: 6, 506: 6,
498: 6, 482: 6, 474: 6, 471: 6, 463: 6, 444: 6, 426: 6, 424: 6, 421: 6, 413: 6, 398: 6, 395: 6,
379: 6, 376: 6, 372: 6, 349: 6, 345: 6, 344: 6, 338: 6, 332: 6, 326: 6, 324: 6, 323: 6, 315: 6,
307: 6, 299: 6, 1813: 5, 1733: 5, 1709: 5, 1400: 5, 1365: 5, 1288: 5, 1051: 5, 976: 5, 949: 5, 843:
5, 822: 5, 788: 5, 781: 5, 770: 5, 752: 5, 692: 5, 678: 5, 665: 5, 651: 5, 641: 5, 639: 5, 635: 5,
632: 5, 630: 5, 608: 5, 602: 5, 596: 5, 589: 5, 581: 5, 577: 5, 568: 5, 564: 5, 561: 5, 548: 5,
545: 5, 543: 5, 541: 5, 535: 5, 527: 5, 523: 5, 517: 5, 514: 5, 508: 5, 500: 5, 489: 5, 488: 5,
487: 5, 484: 5, 478: 5, 470: 5, 465: 5, 460: 5, 451: 5, 446: 5, 442: 5, 437: 5, 434: 5, 425: 5,
423: 5, 420: 5, 411: 5, 405: 5, 403: 5, 402: 5, 399: 5, 396: 5, 392: 5, 389: 5, 388: 5, 365: 5,
364: 5, 363: 5, 263: 5, 2461: 4, 1893: 4, 1735: 4, 1413: 4, 1307: 4, 1270: 4, 1257: 4, 1254: 4, 122
4: 4, 1188: 4, 1156: 4, 1139: 4, 1102: 4, 1091: 4, 1085: 4, 1053: 4, 1011: 4, 997: 4, 986: 4, 972:
4, 964: 4, 951: 4, 950: 4, 929: 4, 889: 4, 870: 4, 852: 4, 847: 4, 840: 4, 832: 4, 830: 4, 814: 4,
804: 4, 798: 4, 789: 4, 775: 4, 772: 4, 771: 4, 767: 4, 741: 4, 738: 4, 734: 4, 724: 4, 715: 4,
698: 4, 696: 4, 688: 4, 681: 4, 679: 4, 677: 4, 671: 4, 669: 4, 667: 4, 662: 4, 660: 4, 657: 4,
655: 4, 654: 4, 653: 4, 642: 4, 633: 4, 629: 4, 628: 4, 627: 4, 626: 4, 624: 4, 618: 4, 616: 4,
611: 4, 604: 4, 601: 4, 598: 4, 595: 4, 594: 4, 593: 4, 591: 4, 590: 4, 587: 4, 573: 4, 570: 4,
567: 4, 562: 4, 555: 4, 552: 4, 551: 4, 547: 4, 544: 4, 540: 4, 534: 4, 531: 4, 528: 4, 521: 4,
520: 4, 519: 4, 518: 4, 507: 4, 504: 4, 503: 4, 502: 4, 499: 4, 497: 4, 496: 4, 494: 4, 493: 4,
491: 4, 490: 4, 483: 4, 469: 4, 464: 4, 461: 4, 453: 4, 452: 4, 450: 4, 449: 4, 445: 4, 390: 4,
387: 4, 374: 4, 369: 4, 346: 4, 317: 4, 6964: 3, 3412: 3, 3287: 3, 3244: 3, 2639: 3, 2500: 3, 2378:
3, 2242: 3, 2234: 3, 2077: 3, 2069: 3, 2061: 3, 1930: 3, 1857: 3, 1852: 3, 1833: 3, 1826: 3, 1809:
3, 1790: 3, 1776: 3, 1728: 3, 1713: 3, 1702: 3, 1699: 3, 1667: 3, 1660: 3, 1644: 3, 1639: 3, 1636:
3, 1576: 3, 1561: 3, 1556: 3, 1550: 3, 1531: 3, 1523: 3, 1414: 3, 1408: 3, 1376: 3, 1350: 3, 1347:
3, 1345: 3, 1298: 3, 1287: 3, 1275: 3, 1258: 3, 1256: 3, 1247: 3, 1238: 3, 1234: 3, 1233: 3, 1228:
3, 1217: 3, 1207: 3, 1199: 3, 1192: 3, 1146: 3, 1140: 3, 1129: 3, 1127: 3, 1100: 3, 1095: 3, 1094:
3, 1093: 3, 1064: 3, 1062: 3, 1056: 3, 1049: 3, 1047: 3, 1035: 3, 1027: 3, 1019: 3, 1018: 3, 1017:
3, 1015: 3, 1006: 3, 1002: 3, 998: 3, 995: 3, 990: 3, 981: 3, 978: 3, 977: 3, 966: 3, 963: 3, 958:
3, 947: 3, 942: 3, 941: 3, 940: 3, 939: 3, 936: 3, 934: 3, 930: 3, 922: 3, 917: 3, 905: 3, 886: 3,
880: 3, 871: 3, 869: 3, 860: 3, 854: 3, 845: 3, 838: 3, 837: 3, 825: 3, 818: 3, 816: 3, 815: 3,
806: 3, 805: 3, 797: 3, 796: 3, 794: 3, 792: 3, 777: 3, 765: 3, 764: 3, 763: 3, 761: 3, 755: 3,
751: 3, 749: 3, 748: 3, 747: 3, 746: 3, 740: 3, 736: 3, 733: 3, 730: 3, 728: 3, 726: 3, 714: 3,
712: 3, 711: 3, 704: 3, 693: 3, 691: 3, 689: 3, 687: 3, 684: 3, 683: 3, 682: 3, 674: 3, 668: 3,
664: 3, 661: 3, 659: 3, 646: 3, 644: 3, 640: 3, 631: 3, 625: 3, 622: 3, 617: 3, 612: 3, 607: 3,
597: 3, 592: 3, 588: 3, 586: 3, 585: 3, 584: 3, 580: 3, 578: 3, 572: 3, 566: 3, 565: 3, 563: 3,
560: 3, 559: 3, 557: 3, 554: 3, 553: 3, 542: 3, 539: 3, 537: 3, 530: 3, 529: 3, 516: 3, 510: 3,
486: 3, 480: 3, 468: 3, 466: 3, 455: 3, 447: 3, 432: 3, 409: 3, 406: 3, 371: 3, 367: 3, 361: 3,
357: 3, 15465: 2, 9845: 2, 8426: 2, 8133: 2, 7089: 2, 7073: 2, 6069: 2, 6033: 2, 5723: 2, 5331: 2,
5108: 2, 4936: 2, 4785: 2, 4776: 2, 4653: 2, 4214: 2, 4185: 2, 3998: 2, 3974: 2, 3896: 2, 3784: 2,
3747: 2, 3689: 2, 3628: 2, 3558: 2, 3521: 2, 3516: 2, 3509: 2, 3477: 2, 3461: 2, 3446: 2, 3437: 2,
3422: 2, 3232: 2, 3170: 2, 3149: 2, 3128: 2, 2991: 2, 2985: 2, 2933: 2, 2867: 2, 2826: 2, 2710: 2,
2703: 2, 2654: 2, 2643: 2, 2635: 2, 2628: 2, 2610: 2, 2599: 2, 2576: 2, 2573: 2, 2550: 2, 2536: 2,
2513: 2, 2490: 2, 2474: 2, 2470: 2, 2447: 2, 2446: 2, 2421: 2, 2415: 2, 2383: 2, 2347: 2, 2338: 2,
2322: 2, 2282: 2, 2256: 2, 2249: 2, 2245: 2, 2212: 2, 2209: 2, 2207: 2, 2185: 2, 2176: 2, 2168: 2,
2163: 2, 2145: 2, 2144: 2, 2135: 2, 2126: 2, 2113: 2, 2094: 2, 2086: 2, 2067: 2, 2056: 2, 2044: 2,
2043: 2, 2035: 2, 2028: 2, 2009: 2, 2007: 2, 2005: 2, 2003: 2, 1995: 2, 1984: 2, 1980: 2, 1979: 2,
[illegible]
```

```
1971: 2, 1950: 2, 1938: 2, 1932: 2, 1922: 2, 1916: 2, 1904: 2, 1897: 2, 1896: 2, 1877: 2, 1856: 2,
1846: 2, 1828: 2, 1817: 2, 1814: 2, 1804: 2, 1798: 2, 1792: 2, 1788: 2, 1771: 2, 1769: 2, 1759: 2,
1739: 2, 1732: 2, 1721: 2, 1714: 2, 1712: 2, 1708: 2, 1688: 2, 1682: 2, 1663: 2, 1662: 2, 1654: 2,
1650: 2, 1648: 2, 1643: 2, 1612: 2, 1610: 2, 1599: 2, 1596: 2, 1595: 2, 1581: 2, 1579: 2, 1575: 2,
1555: 2, 1537: 2, 1536: 2, 1528: 2, 1522: 2, 1516: 2, 1514: 2, 1513: 2, 1512: 2, 1511: 2, 1508: 2,
1506: 2, 1503: 2, 1501: 2, 1498: 2, 1496: 2, 1493: 2, 1491: 2, 1490: 2, 1484: 2, 1481: 2, 1478: 2,
1475: 2, 1467: 2, 1459: 2, 1453: 2, 1452: 2, 1444: 2, 1423: 2, 1417: 2, 1416: 2, 1415: 2, 1406: 2,
1394: 2, 1388: 2, 1384: 2, 1380: 2, 1371: 2, 1370: 2, 1363: 2, 1362: 2, 1355: 2, 1353: 2, 1341: 2,
1336: 2, 1326: 2, 1323: 2, 1320: 2, 1315: 2, 1314: 2, 1311: 2, 1305: 2, 1303: 2, 1302: 2, 1301: 2,
1297: 2, 1296: 2, 1295: 2, 1290: 2, 1278: 2, 1276: 2, 1272: 2, 1271: 2, 1269: 2, 1261: 2, 1259: 2,
1251: 2, 1245: 2, 1244: 2, 1243: 2, 1241: 2, 1236: 2, 1232: 2, 1221: 2, 1219: 2, 1218: 2, 1215: 2,
1208: 2, 1205: 2, 1201: 2, 1197: 2, 1190: 2, 1189: 2, 1178: 2, 1167: 2, 1164: 2, 1159: 2, 1153: 2,
1152: 2, 1151: 2, 1149: 2, 1143: 2, 1126: 2, 1125: 2, 1124: 2, 1121: 2, 1117: 2, 1111: 2, 1104: 2,
1101: 2, 1088: 2, 1084: 2, 1083: 2, 1080: 2, 1079: 2, 1078: 2, 1077: 2, 1072: 2, 1069: 2, 1068: 2,
1067: 2, 1066: 2, 1063: 2, 1050: 2, 1045: 2, 1044: 2, 1042: 2, 1041: 2, 1040: 2, 1039: 2, 1037: 2,
1032: 2, 1026: 2, 1023: 2, 1005: 2, 1000: 2, 993: 2, 987: 2, 984: 2, 980: 2, 974: 2, 971: 2, 962: 2
, 948: 2, 935: 2, 932: 2, 926: 2, 923: 2, 918: 2, 916: 2, 912: 2, 911: 2, 908: 2, 904: 2, 903: 2,
901: 2, 900: 2, 896: 2, 895: 2, 894: 2, 890: 2, 882: 2, 879: 2, 875: 2, 874: 2, 872: 2, 865: 2,
863: 2, 862: 2, 861: 2, 853: 2, 850: 2, 849: 2, 848: 2, 844: 2, 836: 2, 833: 2, 829: 2, 828: 2,
820: 2, 817: 2, 813: 2, 812: 2, 809: 2, 802: 2, 801: 2, 800: 2, 799: 2, 795: 2, 790: 2, 787: 2,
786: 2, 785: 2, 784: 2, 779: 2, 776: 2, 774: 2, 762: 2, 760: 2, 759: 2, 758: 2, 757: 2, 754: 2,
753: 2, 745: 2, 743: 2, 742: 2, 737: 2, 731: 2, 729: 2, 722: 2, 718: 2, 716: 2, 713: 2, 710: 2,
709: 2, 703: 2, 701: 2, 699: 2, 694: 2, 690: 2, 676: 2, 675: 2, 658: 2, 652: 2, 649: 2, 648: 2,
643: 2, 638: 2, 634: 2, 621: 2, 620: 2, 614: 2, 613: 2, 610: 2, 600: 2, 599: 2, 579: 2, 575: 2,
571: 2, 556: 2, 533: 2, 522: 2, 509: 2, 477: 2, 457: 2, 440: 2, 418: 2, 416: 2, 380: 2, 152454: 1,
118681: 1, 81146: 1, 68181: 1, 68141: 1, 68100: 1, 65828: 1, 64218: 1, 62858: 1, 54821: 1, 53242:
1, 48894: 1, 48544: 1, 46445: 1, 46114: 1, 43738: 1, 42781: 1, 42306: 1, 41907: 1, 40516: 1, 40402
: 1, 39815: 1, 38844: 1, 38346: 1, 37612: 1, 37304: 1, 36336: 1, 36112: 1, 36045: 1, 34808: 1, 341
02: 1, 33751: 1, 33450: 1, 33347: 1, 32818: 1, 31681: 1, 29190: 1, 28093: 1, 27843: 1, 25936: 1, 2
5919: 1, 25702: 1, 25691: 1, 25659: 1, 25286: 1, 24521: 1, 24426: 1, 24352: 1, 24086: 1, 24021: 1,
23509: 1, 23269: 1, 23193: 1, 22617: 1, 22205: 1, 22029: 1, 21480: 1, 21461: 1, 21204: 1, 20701: 1
, 20599: 1, 20320: 1, 20098: 1, 19556: 1, 19471: 1, 19400: 1, 19281: 1, 19238: 1, 19212: 1, 19128:
1, 19038: 1, 18864: 1, 18808: 1, 18717: 1, 18539: 1, 18464: 1, 18388: 1, 18118: 1, 18050: 1, 17950
: 1, 17947: 1, 17831: 1, 17636: 1, 17574: 1, 17565: 1, 17564: 1, 17457: 1, 17336: 1, 17319: 1, 172
74: 1, 17121: 1, 17084: 1, 16914: 1, 16843: 1, 16795: 1, 16625: 1, 16516: 1, 16252: 1, 15899: 1, 1
5634: 1, 15561: 1, 15489: 1, 15477: 1, 15379: 1, 15319: 1, 15136: 1, 14983: 1, 14956: 1, 14826: 1,
14701: 1, 14680: 1, 14663: 1, 14607: 1, 14599: 1, 14585: 1, 14480: 1, 14405: 1, 14215: 1, 14125: 1
, 13895: 1, 13856: 1, 13736: 1, 13700: 1, 13595: 1, 13466: 1, 13412: 1, 13395: 1, 13280: 1, 13214:
1, 13188: 1, 13152: 1, 13130: 1, 13026: 1, 12959: 1, 12937: 1, 12897: 1, 12888: 1, 12832: 1, 12688
: 1, 12586: 1, 12562: 1, 12551: 1, 12486: 1, 12462: 1, 12458: 1, 12383: 1, 12346: 1, 12316: 1, 123
10: 1, 12286: 1, 12230: 1, 12217: 1, 12216: 1, 12215: 1, 12175: 1, 12162: 1, 12156: 1, 12150: 1, 1
2122: 1, 12068: 1, 12048: 1, 12046: 1, 11984: 1, 11980: 1, 11955: 1, 11764: 1, 11654: 1, 11645: 1,
11607: 1, 11572: 1, 11529: 1, 11527: 1, 11452: 1, 11385: 1, 11338: 1, 11308: 1, 11241: 1, 11197: 1
, 11018: 1, 10975: 1, 10939: 1, 10807: 1, 10792: 1, 10596: 1, 10581: 1, 10503: 1, 10478: 1, 10467:
1, 10432: 1, 10348: 1, 10241: 1, 10240: 1, 10217: 1, 10185: 1, 10020: 1, 10016: 1, 9975: 1, 9965:
1, 9962: 1, 9926: 1, 9906: 1, 9865: 1, 9851: 1, 9824: 1, 9818: 1, 9800: 1, 9728: 1, 9703: 1, 9702:
1, 9655: 1, 9570: 1, 9530: 1, 9452: 1, 9409: 1, 9390: 1, 9327: 1, 9323: 1, 9253: 1, 9203: 1, 9157:
1, 9145: 1, 9110: 1, 9105: 1, 9088: 1, 9054: 1, 9051: 1, 9041: 1, 9028: 1, 9026: 1, 9018: 1, 8989:
1, 8984: 1, 8963: 1, 8894: 1, 8891: 1, 8836: 1, 8809: 1, 8799: 1, 8746: 1, 8710: 1, 8642: 1, 8641:
1, 8631: 1, 8583: 1, 8568: 1, 8393: 1, 8364: 1, 8361: 1, 8344: 1, 8282: 1, 8257: 1, 8244: 1, 8208:
1, 8185: 1, 8161: 1, 8143: 1, 8135: 1, 8090: 1, 8080: 1, 8074: 1, 8058: 1, 8036: 1, 8028: 1, 7950:
1, 7912: 1, 7882: 1, 7848: 1, 7842: 1, 7816: 1, 7805: 1, 7768: 1, 7749: 1, 7733: 1, 7723: 1, 7718:
1, 7710: 1, 7690: 1, 7647: 1, 7612: 1, 7602: 1, 7541: 1, 7528: 1, 7518: 1, 7494: 1, 7493: 1, 7484:
1, 7476: 1, 7472: 1, 7444: 1, 7429: 1, 7391: 1, 7382: 1, 7344: 1, 7339: 1, 7298: 1, 7288: 1, 7282:
1, 7264: 1, 7248: 1, 7235: 1, 7230: 1, 7223: 1, 7193: 1, 7192: 1, 7150: 1, 7136: 1, 7098: 1, 7071:
1, 7038: 1, 7024: 1, 7022: 1, 6975: 1, 6967: 1, 6953: 1, 6942: 1, 6887: 1, 6880: 1, 6879: 1, 6849:
1, 6842: 1, 6830: 1, 6827: 1, 6797: 1, 6795: 1, 6792: 1, 6791: 1, 6776: 1, 6760: 1, 6701: 1, 6692:
1, 6653: 1, 6647: 1, 6638: 1, 6629: 1, 6598: 1, 6590: 1, 6586: 1, 6570: 1, 6550: 1, 6539: 1, 6515:
1, 6512: 1, 6509: 1, 6481: 1, 6450: 1, 6438: 1, 6433: 1, 6431: 1, 6424: 1, 6353: 1, 6343: 1, 6336:
1, 6334: 1, 6319: 1, 6317: 1, 6315: 1, 6304: 1, 6288: 1, 6268: 1, 6226: 1, 6222: 1, 6207: 1, 6202:
1, 6188: 1, 6185: 1, 6172: 1, 6166: 1, 6159: 1, 6156: 1, 6143: 1, 6137: 1, 6134: 1, 6119: 1, 6080:
1, 6065: 1, 6030: 1, 6010: 1, 6005: 1, 6001: 1, 5982: 1, 5972: 1, 5952: 1, 5938: 1, 5931: 1, 5927:
1, 5911: 1, 5900: 1, 5884: 1, 5874: 1, 5871: 1, 5855: 1, 5841: 1, 5823: 1, 5818: 1, 5803: 1, 5797:
1, 5791: 1, 5787: 1, 5786: 1, 5785: 1, 5774: 1, 5773: 1, 5699: 1, 5680: 1, 5676: 1, 5673: 1, 5658:
1, 5643: 1, 5606: 1, 5603: 1, 5602: 1, 5596: 1, 5592: 1, 5586: 1, 5578: 1, 5573: 1, 5564: 1, 5557:
1, 5554: 1, 5495: 1, 5452: 1, 5451: 1, 5441: 1, 5420: 1, 5411: 1, 5401: 1, 5395: 1, 5380: 1, 5371:
1, 5362: 1, 5353: 1, 5348: 1, 5325: 1, 5320: 1, 5298: 1, 5296: 1, 5293: 1, 5284: 1, 5262: 1, 5261:
1, 5204: 1, 5186: 1, 5172: 1, 5169: 1, 5160: 1, 5149: 1, 5132: 1, 5130: 1, 5129: 1, 5124: 1, 5106:
1, 5102: 1, 5093: 1, 5086: 1, 5082: 1, 5041: 1, 5039: 1, 5037: 1, 5036: 1, 5034: 1, 5025: 1, 5018:
1, 5013: 1, 4992: 1, 4985: 1, 4967: 1, 4963: 1, 4946: 1, 4933: 1, 4928: 1, 4925: 1, 4909: 1, 4908:
1, 4896: 1, 4877: 1, 4866: 1, 4864: 1, 4849: 1, 4842: 1, 4841: 1, 4839: 1, 4811: 1, 4810: 1, 4809:
1, 4808: 1, 4787: 1, 4764: 1, 4753: 1, 4749: 1, 4743: 1, 4730: 1, 4721: 1, 4717: 1, 4713: 1, 4698:
1, 4678: 1, 4668: 1, 4649: 1, 4646: 1, 4640: 1, 4635: 1, 4608: 1, 4607: 1, 4592: 1, 4588: 1, 4586:
1, 4582: 1, 4580: 1, 4577: 1, 4544: 1, 4536: 1, 4535: 1, 4534: 1, 4519: 1, 4515: 1, 4489: 1, 4485:
1, 4484: 1, 4481: 1, 4480: 1, 4479: 1, 4469: 1, 4457: 1, 4454: 1, 4449: 1, 4436: 1, 4426: 1, 4414:
1, 4413: 1, 4412: 1, 4409: 1, 4400: 1, 4389: 1, 4388: 1, 4384: 1, 4379: 1, 4365: 1, 4354: 1, 4349:
1, 4345: 1, 4338: 1, 4331: 1, 4326: 1, 4323: 1, 4322: 1, 4317: 1, 4315: 1, 4311: 1, 4310: 1, 4309:
```

```
1, 4301: 1, 4299: 1, 4289: 1, 4279: 1, 4273: 1, 4272: 1, 4254: 1, 4251: 1, 4241: 1, 4238: 1, 4237:
1, 4230: 1, 4205: 1, 4194: 1, 4182: 1, 4179: 1, 4157: 1, 4154: 1, 4144: 1, 4138: 1, 4128: 1, 4123:
1, 4121: 1, 4117: 1, 4115: 1, 4092: 1, 4091: 1, 4090: 1, 4085: 1, 4078: 1, 4075: 1, 4070: 1, 4062:
1, 4053: 1, 4052: 1, 4040: 1, 4034: 1, 4028: 1, 4022: 1, 4019: 1, 4018: 1, 4015: 1, 4013: 1, 4007:
1, 3997: 1, 3993: 1, 3991: 1, 3985: 1, 3982: 1, 3979: 1, 3970: 1, 3960: 1, 3956: 1, 3949: 1, 3945:
1, 3944: 1, 3942: 1, 3940: 1, 3927: 1, 3926: 1, 3922: 1, 3916: 1, 3909: 1, 3904: 1, 3893: 1, 3885:
1, 3874: 1, 3872: 1, 3866: 1, 3861: 1, 3853: 1, 3852: 1, 3844: 1, 3832: 1, 3828: 1, 3827: 1, 3816:
1, 3813: 1, 3808: 1, 3783: 1, 3781: 1, 3773: 1, 3768: 1, 3764: 1, 3756: 1, 3752: 1, 3742: 1, 3740:
1, 3739: 1, 3724: 1, 3715: 1, 3708: 1, 3706: 1, 3698: 1, 3697: 1, 3680: 1, 3675: 1, 3660: 1, 3659:
1, 3658: 1, 3654: 1, 3651: 1, 3647: 1, 3646: 1, 3644: 1, 3642: 1, 3636: 1, 3634: 1, 3631: 1, 3627:
1, 3620: 1, 3610: 1, 3609: 1, 3593: 1, 3592: 1, 3582: 1, 3577: 1, 3572: 1, 3567: 1, 3564: 1, 3560:
1, 3557: 1, 3554: 1, 3549: 1, 3548: 1, 3545: 1, 3544: 1, 3541: 1, 3540: 1, 3520: 1, 3515: 1, 3503:
1, 3495: 1, 3493: 1, 3492: 1, 3490: 1, 3487: 1, 3483: 1, 3478: 1, 3462: 1, 3457: 1, 3453: 1, 3444:
1, 3443: 1, 3440: 1, 3432: 1, 3428: 1, 3419: 1, 3417: 1, 3416: 1, 3410: 1, 3409: 1, 3399: 1, 3398:
1, 3397: 1, 3393: 1, 3392: 1, 3382: 1, 3377: 1, 3375: 1, 3374: 1, 3368: 1, 3364: 1, 3356: 1, 3342:
1, 3332: 1, 3331: 1, 3323: 1, 3322: 1, 3320: 1, 3318: 1, 3315: 1, 3310: 1, 3309: 1, 3307: 1, 3303:
1, 3301: 1, 3295: 1, 3291: 1, 3288: 1, 3284: 1, 3266: 1, 3261: 1, 3251: 1, 3245: 1, 3243: 1, 3240:
1, 3239: 1, 3230: 1, 3229: 1, 3227: 1, 3225: 1, 3224: 1, 3222: 1, 3221: 1, 3218: 1, 3215: 1, 3212:
1, 3209: 1, 3204: 1, 3201: 1, 3196: 1, 3193: 1, 3187: 1, 3186: 1, 3178: 1, 3168: 1, 3150: 1, 3144:
1, 3141: 1, 3140: 1, 3139: 1, 3137: 1, 3129: 1, 3126: 1, 3115: 1, 3111: 1, 3108: 1, 3105: 1, 3091:
1, 3089: 1, 3084: 1, 3079: 1, 3078: 1, 3067: 1, 3063: 1, 3060: 1, 3047: 1, 3045: 1, 3041: 1, 3038:
1, 3036: 1, 3034: 1, 3023: 1, 3022: 1, 3021: 1, 3020: 1, 3013: 1, 3012: 1, 3011: 1, 3010: 1, 3009:
1, 3008: 1, 2999: 1, 2995: 1, 2993: 1, 2992: 1, 2989: 1, 2979: 1, 2973: 1, 2971: 1, 2968: 1, 2967:
1, 2963: 1, 2959: 1, 2957: 1, 2956: 1, 2951: 1, 2950: 1, 2948: 1, 2947: 1, 2944: 1, 2940: 1, 2937:
1, 2932: 1, 2926: 1, 2911: 1, 2903: 1, 2900: 1, 2897: 1, 2893: 1, 2889: 1, 2887: 1, 2883: 1, 2851:
1, 2848: 1, 2847: 1, 2846: 1, 2839: 1, 2838: 1, 2836: 1, 2830: 1, 2828: 1, 2827: 1, 2812: 1, 2811:
1, 2800: 1, 2798: 1, 2792: 1, 2791: 1, 2788: 1, 2782: 1, 2775: 1, 2770: 1, 2767: 1, 2762: 1, 2756:
1, 2754: 1, 2748: 1, 2743: 1, 2733: 1, 2731: 1, 2726: 1, 2718: 1, 2714: 1, 2711: 1, 2709: 1, 2698:
1, 2693: 1, 2691: 1, 2689: 1, 2676: 1, 2675: 1, 2674: 1, 2668: 1, 2667: 1, 2666: 1, 2664: 1, 2660:
1, 2658: 1, 2657: 1, 2656: 1, 2653: 1, 2647: 1, 2646: 1, 2644: 1, 2636: 1, 2627: 1, 2614: 1, 2613:
1, 2612: 1, 2607: 1, 2605: 1, 2600: 1, 2596: 1, 2593: 1, 2587: 1, 2582: 1, 2578: 1, 2575: 1, 2570:
1, 2566: 1, 2563: 1, 2561: 1, 2554: 1, 2553: 1, 2542: 1, 2539: 1, 2535: 1, 2530: 1, 2529: 1, 2526:
1, 2524: 1, 2518: 1, 2516: 1, 2508: 1, 2499: 1, 2495: 1, 2493: 1, 2492: 1, 2488: 1, 2487: 1, 2485:
1, 2484: 1, 2482: 1, 2476: 1, 2469: 1, 2465: 1, 2458: 1, 2456: 1, 2442: 1, 2441: 1, 2438: 1, 2437:
1, 2434: 1, 2433: 1, 2430: 1, 2426: 1, 2424: 1, 2423: 1, 2412: 1, 2410: 1, 2408: 1, 2403: 1, 2395:
1, 2394: 1, 2393: 1, 2390: 1, 2387: 1, 2385: 1, 2384: 1, 2382: 1, 2381: 1, 2379: 1, 2375: 1, 2373:
1, 2371: 1, 2364: 1, 2361: 1, 2359: 1, 2356: 1, 2355: 1, 2353: 1, 2351: 1, 2340: 1, 2337: 1, 2334:
1, 2330: 1, 2327: 1, 2324: 1, 2314: 1, 2312: 1, 2310: 1, 2309: 1, 2307: 1, 2298: 1, 2292: 1, 2284:
1, 2276: 1, 2274: 1, 2273: 1, 2272: 1, 2269: 1, 2268: 1, 2265: 1, 2264: 1, 2262: 1, 2258: 1, 2257:
1, 2247: 1, 2246: 1, 2244: 1, 2238: 1, 2237: 1, 2230: 1, 2227: 1, 2223: 1, 2218: 1, 2216: 1, 2211:
1, 2201: 1, 2199: 1, 2191: 1, 2187: 1, 2186: 1, 2181: 1, 2180: 1, 2173: 1, 2172: 1, 2171: 1, 2170:
1, 2162: 1, 2150: 1, 2148: 1, 2147: 1, 2146: 1, 2142: 1, 2140: 1, 2132: 1, 2131: 1, 2129: 1, 2127:
1, 2123: 1, 2121: 1, 2118: 1, 2117: 1, 2115: 1, 2114: 1, 2112: 1, 2108: 1, 2104: 1, 2100: 1, 2099:
1, 2096: 1, 2093: 1, 2092: 1, 2091: 1, 2090: 1, 2089: 1, 2088: 1, 2087: 1, 2085: 1, 2081: 1, 2078:
1, 2071: 1, 2070: 1, 2060: 1, 2059: 1, 2053: 1, 2052: 1, 2049: 1, 2047: 1, 2045: 1, 2042: 1, 2041:
1, 2036: 1, 2034: 1, 2030: 1, 2020: 1, 2019: 1, 2015: 1, 2013: 1, 2012: 1, 2006: 1, 2001: 1, 1993:
1, 1991: 1, 1988: 1, 1986: 1, 1982: 1, 1975: 1, 1973: 1, 1970: 1, 1967: 1, 1959: 1, 1955: 1, 1954:
1, 1951: 1, 1946: 1, 1943: 1, 1942: 1, 1939: 1, 1936: 1, 1934: 1, 1929: 1, 1923: 1, 1918: 1, 1917:
1, 1914: 1, 1912: 1, 1910: 1, 1902: 1, 1895: 1, 1889: 1, 1882: 1, 1881: 1, 1880: 1, 1878: 1, 1873:
1, 1871: 1, 1870: 1, 1868: 1, 1865: 1, 1860: 1, 1848: 1, 1847: 1, 1840: 1, 1838: 1, 1836: 1, 1831:
1, 1829: 1, 1823: 1, 1822: 1, 1821: 1, 1819: 1, 1816: 1, 1807: 1, 1805: 1, 1803: 1, 1802: 1, 1800:
1, 1799: 1, 1797: 1, 1785: 1, 1783: 1, 1780: 1, 1779: 1, 1777: 1, 1774: 1, 1770: 1, 1768: 1, 1766:
1, 1763: 1, 1762: 1, 1758: 1, 1756: 1, 1754: 1, 1752: 1, 1750: 1, 1747: 1, 1746: 1, 1745: 1, 1744:
1, 1741: 1, 1740: 1, 1731: 1, 1730: 1, 1729: 1, 1727: 1, 1725: 1, 1718: 1, 1711: 1, 1710: 1, 1704:
1, 1703: 1, 1701: 1, 1695: 1, 1693: 1, 1689: 1, 1686: 1, 1683: 1, 1681: 1, 1680: 1, 1679: 1, 1677:
1, 1676: 1, 1675: 1, 1673: 1, 1672: 1, 1666: 1, 1665: 1, 1659: 1, 1655: 1, 1641: 1, 1640: 1, 1638:
1, 1637: 1, 1635: 1, 1634: 1, 1626: 1, 1623: 1, 1616: 1, 1615: 1, 1607: 1, 1606: 1, 1605: 1, 1603:
1, 1601: 1, 1597: 1, 1593: 1, 1589: 1, 1588: 1, 1586: 1, 1583: 1, 1580: 1, 1578: 1, 1572: 1, 1570:
1, 1569: 1, 1566: 1, 1565: 1, 1564: 1, 1563: 1, 1560: 1, 1558: 1, 1554: 1, 1553: 1, 1552: 1, 1551:
1, 1548: 1, 1547: 1, 1545: 1, 1544: 1, 1540: 1, 1534: 1, 1533: 1, 1532: 1, 1526: 1, 1525: 1, 1524:
1, 1520: 1, 1518: 1, 1517: 1, 1510: 1, 1509: 1, 1507: 1, 1504: 1, 1502: 1, 1500: 1, 1495: 1, 1492:
1, 1489: 1, 1486: 1, 1483: 1, 1480: 1, 1479: 1, 1477: 1, 1474: 1, 1472: 1, 1471: 1, 1464: 1, 1463:
1, 1461: 1, 1458: 1, 1457: 1, 1450: 1, 1449: 1, 1447: 1, 1445: 1, 1442: 1, 1438: 1, 1437: 1, 1434:
1, 1433: 1, 1431: 1, 1430: 1, 1429: 1, 1427: 1, 1426: 1, 1425: 1, 1424: 1, 1419: 1, 1418: 1, 1410:
1, 1407: 1, 1402: 1, 1401: 1, 1399: 1, 1397: 1, 1395: 1, 1390: 1, 1385: 1, 1383: 1, 1379: 1, 1377:
1, 1375: 1, 1374: 1, 1373: 1, 1369: 1, 1368: 1, 1367: 1, 1366: 1, 1364: 1, 1361: 1, 1360: 1, 1356:
1, 1354: 1, 1352: 1, 1351: 1, 1349: 1, 1348: 1, 1346: 1, 1343: 1, 1342: 1, 1338: 1, 1337: 1, 1335:
1, 1334: 1, 1332: 1, 1331: 1, 1329: 1, 1328: 1, 1319: 1, 1318: 1, 1316: 1, 1313: 1, 1310: 1, 1304:
1, 1300: 1, 1299: 1, 1289: 1, 1284: 1, 1282: 1, 1280: 1, 1273: 1, 1267: 1, 1266: 1, 1264: 1, 1255:
1, 1253: 1, 1249: 1, 1246: 1, 1240: 1, 1239: 1, 1231: 1, 1230: 1, 1229: 1, 1227: 1, 1223: 1, 1222:
1, 1214: 1, 1213: 1, 1212: 1, 1211: 1, 1209: 1, 1206: 1, 1204: 1, 1203: 1, 1202: 1, 1200: 1, 1198:
1, 1196: 1, 1195: 1, 1193: 1, 1187: 1, 1179: 1, 1176: 1, 1175: 1, 1171: 1, 1168: 1, 1165: 1, 1163:
1, 1161: 1, 1157: 1, 1154: 1, 1150: 1, 1148: 1, 1144: 1, 1138: 1, 1134: 1, 1133: 1, 1132: 1, 1128:
1, 1123: 1, 1122: 1, 1120: 1, 1119: 1, 1118: 1, 1116: 1, 1114: 1, 1113: 1, 1109: 1, 1108: 1, 1107:
1, 1103: 1, 1090: 1, 1089: 1, 1087: 1, 1081: 1, 1076: 1, 1075: 1, 1074: 1, 1071: 1, 1070: 1, 1061:
1, 1057: 1, 1055: 1, 1054: 1, 1048: 1, 1038: 1, 1036: 1, 1034: 1, 1033: 1, 1031: 1, 1025: 1, 1022:
1, 1021: 1, 1014: 1, 1013: 1, 1012: 1, 1009: 1, 1007: 1, 1004: 1, 1001: 1, 994: 1, 988: 1, 985: 1,
983: 1, 979: 1, 975: 1, 970: 1, 969: 1, 968: 1, 967: 1, 960: 1, 959: 1, 956: 1, 955: 1, 954: 1,
953: 1, 952: 1, 946: 1, 945: 1, 943: 1, 938: 1, 937: 1, 933: 1, 931: 1, 928: 1, 921: 1, 919: 1,
909: 1, 907: 1, 906: 1, 902: 1, 899: 1, 897: 1, 893: 1, 892: 1, 884: 1, 883: 1, 878: 1, 877: 1,
876: 1, 873: 1, 868: 1, 867: 1, 866: 1, 859: 1, 858: 1, 857: 1, 855: 1, 851: 1, 846: 1, 841: 1,
835: 1, 834: 1, 831: 1, 826: 1, 824: 1, 823: 1, 821: 1, 811: 1, 810: 1, 807: 1, 803: 1, 793: 1,
783: 1, 782: 1, 780: 1, 778: 1, 769: 1, 768: 1, 766: 1, 750: 1, 744: 1, 739: 1, 732: 1, 727: 1,
725: 1, 723: 1, 721: 1, 717: 1, 706: 1, 700: 1, 697: 1, 695: 1, 685: 1, 670: 1, 666: 1, 656: 1,
650: 1, 645: 1, 637: 1, 619: 1, 615: 1, 609: 1, 606: 1, 603: 1, 574: 1, 569: 1, 558: 1, 526: 1,
481: 1, 472: 1})
```

In [78]:

```python
# don't forget to normalize every feature
text_tfidf = TfidfVectorizer(max_features=1000)
train_text_feature_tfidf =  text_tfidf.fit_transform(train_df['TEXT'])
train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)

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

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

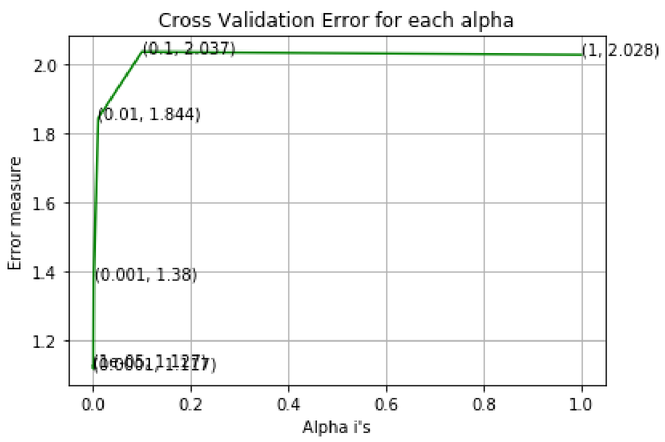
In [79]:

```python
# don't forget to normalize every feature
text_bi = CountVectorizer(ngram_range=(1,2),min_df=4)
train_text_feature_bi =  text_bi.fit_transform(train_df['TEXT'])
train_text_feature_bi = normalize(train_text_feature_bi, axis=0)

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

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

In [80]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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:
#------------------------------


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

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.1272516207901209
For values of alpha =  0.0001 The log loss is: 1.1171927860931992
For values of alpha =  0.001 The log loss is: 1.380369536686068
For values of alpha =  0.01 The log loss is: 1.8439942941546699
For values of alpha =  0.1 The log loss is: 2.036749612704292
For values of alpha =  1 The log loss is: 2.0279381002724803
```



```
For values of best alpha =  0.0001 The train log loss is: 0.8222455031052962
For values of best alpha =  0.0001 The cross validation log loss is: 1.1171927860931992
For values of best alpha =  0.0001 The test log loss is: 1.1536583085975054
```

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

Ans. Yes, it seems like!

In [81]:

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

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

In [82]:

```
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")
```

```
99.286 % of word of test data appeared in train data
99.797 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [83]:

```
#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 belongs 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 [84]:

```
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 [85]:

```
# 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 [{}]".format(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_r
o))
        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 [{}]".format(word,yes_no))

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

In [87]:

```
def get_impfeature_tfidf(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer(max_features=1000)
    var_count_vec = TfidfVectorizer(max_features=1000)
    text_count_vec = TfidfVectorizer(max_features=1000)

    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 [{}]".format(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_r
o))
        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 [{}]".format(word,yes_no))

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

Stacking the three types of features

In [88]:

```
# 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_variation_feature_onehotCoding))
```

```
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)
)

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
r()
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_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding =
np.hstack((train_gene_feature_responseCoding,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_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,
train_text_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_feature_responseCoding))
```

In [90]:

```
train_gene_var_tfidf = hstack((train_gene_feature_tfidf,train_variation_feature_tfidf))
test_gene_var_tfidf = hstack((test_gene_feature_tfidf,test_variation_feature_tfidf))
cv_gene_var_tfidf = hstack((cv_gene_feature_tfidf,cv_variation_feature_tfidf))

train_x_tfidf = hstack((train_gene_var_tfidf, train_text_feature_tfidf)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_tfidf = hstack((test_gene_var_tfidf, test_text_feature_tfidf)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_tfidf = hstack((cv_gene_var_tfidf, cv_text_feature_tfidf)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [92]:

```
train_x_tfidf_gent = hstack((train_gene_feature_tfidf, train_text_feature_tfidf)).tocsr()


test_x_tfidf_gent = hstack((test_gene_feature_tfidf, test_text_feature_tfidf)).tocsr()


cv_x_tfidf_gent = hstack((cv_gene_feature_tfidf, cv_text_feature_tfidf)).tocsr()
```

In [93]:

```
train_gene_var_onehotCoding_bi =
hstack((train_gene_feature_onehotCoding_bi,train_variation_feature_onehotCoding_bi))
test_gene_var_onehotCoding_bi =
hstack((test_gene_feature_onehotCoding_bi,test_variation_feature_onehotCoding_bi))
cv_gene_var_onehotCoding_bi =
hstack((cv_gene_feature_onehotCoding_bi,cv_variation_feature_onehotCoding_bi))

train_x_onehotCoding_bi = hstack((train_gene_var_onehotCoding, train_text_feature_bi)).tocsr()


test_x_onehotCoding_bi = hstack((test_gene_var_onehotCoding, test_text_feature_bi)).tocsr()


cv_x_onehotCoding_bi = hstack((cv_gene_var_onehotCoding, cv_text_feature_bi)).tocsr()
```

In [94]:

```
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, 128309)
(number of data points * number of features) in test data =  (665, 128309)
(number of data points * number of features) in cross validation data = (532, 128309)
```

In [95]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shap
e)
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)
```

In [96]:

```
print("Tfidf features :")
print("(number of data points * number of features) in train data = ", train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", test_x_tfidf.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_tfidf.shape)
```

```
Tfidf features :
(number of data points * number of features) in train data =  (2124, 2229)
(number of data points * number of features) in test data =  (665, 2229)
(number of data points * number of features) in cross validation data = (532, 2229)
```

In [97]:

```
print("bi gramsfeatures :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding_bi.sha
pe)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding_bi.shape
)
print("(number of data points * number of features) in cross validation data =",
cv_x_onehotCoding_bi.shape)
```

```
bi gramsfeatures :
(number of data points * number of features) in train data =  (2124, 610882)
(number of data points * number of features) in test data =  (665, 610882)
(number of data points * number of features) in cross validation data = (532, 610882)
```

4.1. Base Line Model

4.1.1. Naive Bayes

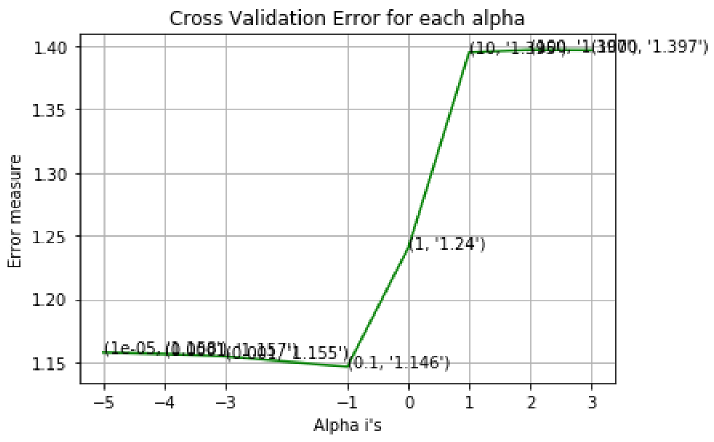
4.1.1.1. Hyper parameter tuning

In [98]:

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

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


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


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

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


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


predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.1578862987691054
for alpha = 0.0001
Log Loss : 1.1566259164435564
for alpha = 0.001
Log Loss : 1.1547515985865076
for alpha = 0.1
Log Loss : 1.14646804364171
for alpha = 1
Log Loss : 1.240315270907704
```

```
for alpha = 10
Log Loss : 1.3948732640619954
for alpha = 100
Log Loss : 1.3967964280017366
for alpha = 1000
Log Loss : 1.3965710515668486
```



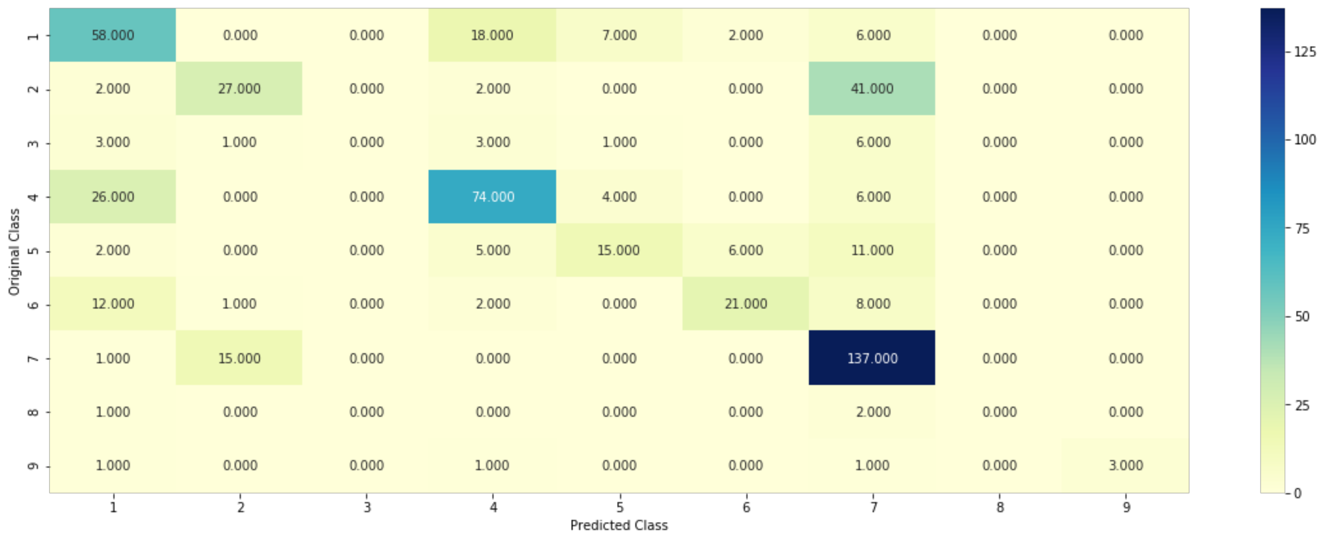
```
For values of best alpha =  0.1 The train log loss is: 0.7788468048793181
For values of best alpha =  0.1 The cross validation log loss is: 1.14646804364171
For values of best alpha =  0.1 The test log loss is: 1.20969260535979
```

4.1.1.2. Testing the model with best hyper paramters

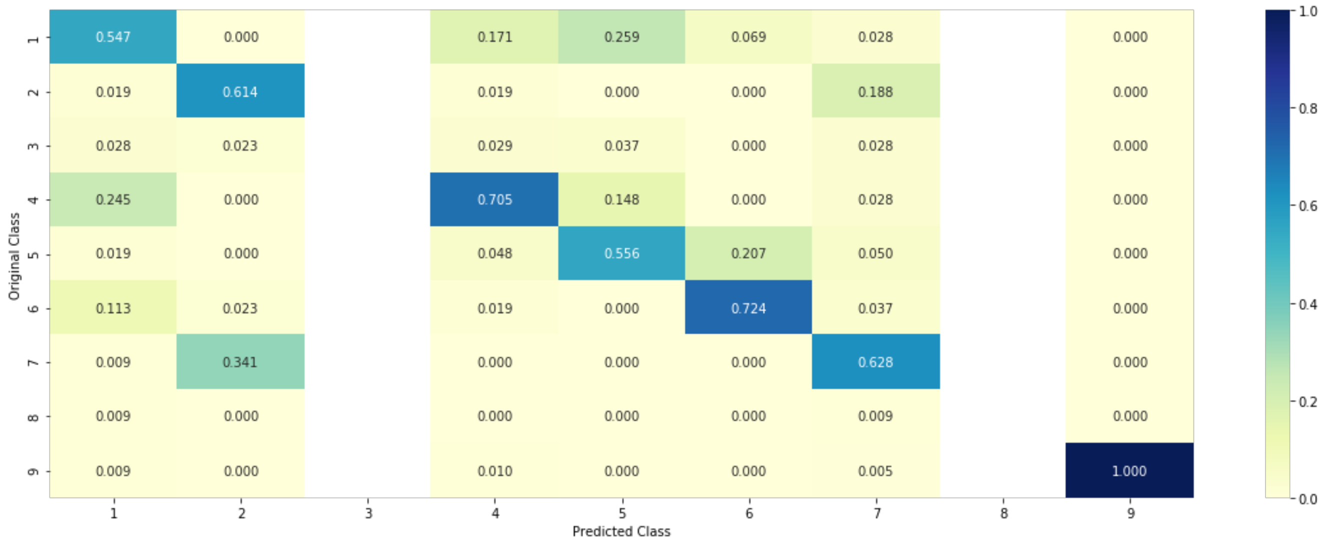
In [99]:

```
# 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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
# 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_tfidf)- cv_y))/cv
_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf.toarray()))
```

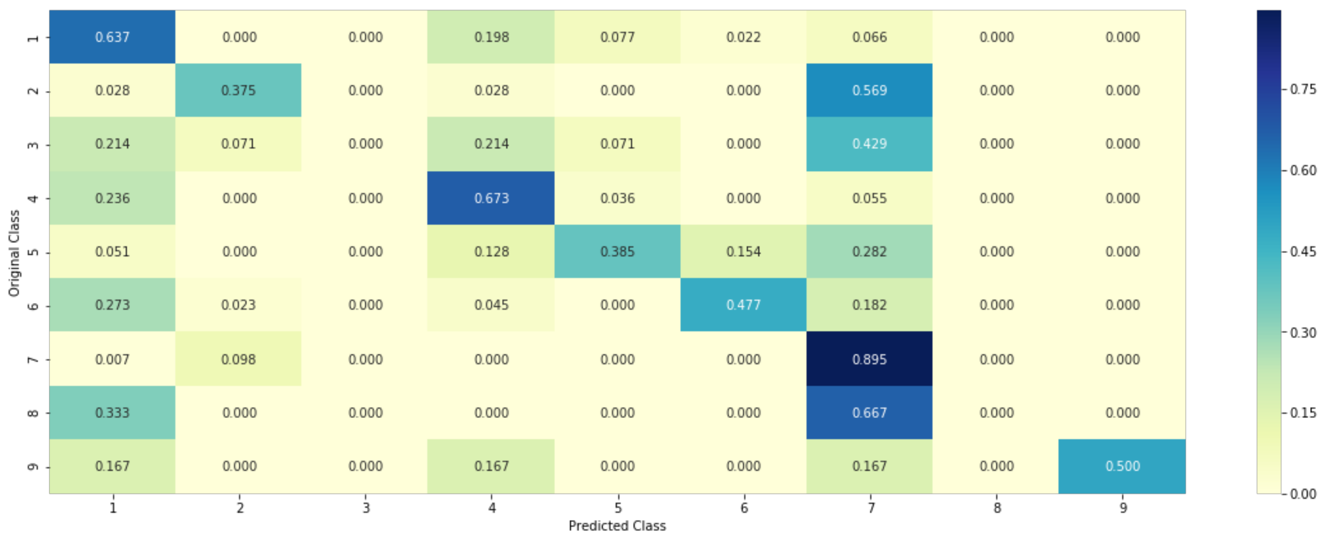
```
Log Loss : 1.14646804364171
Number of missclassified point : 0.37030075187969924
Number of misclassified point : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [100]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[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_tfidf(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3616 0.0491 0.0176 0.4267 0.0391 0.0374 0.0601 0.0047 0.0038]]
Actual Class : 4
--------------------------------------------------
12 Text feature [activity] present in test data point [True]
17 Text feature [protein] present in test data point [True]
18 Text feature [proteins] present in test data point [True]
19 Text feature [function] present in test data point [True]
22 Text feature [results] present in test data point [True]
23 Text feature [shown] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [important] present in test data point [True]
30 Text feature [type] present in test data point [True]
31 Text feature [whether] present in test data point [True]
32 Text feature [suppressor] present in test data point [True]
33 Text feature [two] present in test data point [True]
34 Text feature [functional] present in test data point [True]
35 Text feature [mutations] present in test data point [True]
36 Text feature [loss] present in test data point [True]
37 Text feature [determined] present in test data point [True]
38 Text feature [wild] present in test data point [True]
39 Text feature [described] present in test data point [True]
40 Text feature [reduced] present in test data point [True]
41 Text feature [may] present in test data point [True]
42 Text feature [either] present in test data point [True]
43 Text feature [although] present in test data point [True]
44 Text feature [indicate] present in test data point [True]
45 Text feature [therefore] present in test data point [True]
47 Text feature [catalytic] present in test data point [True]
48 Text feature [determine] present in test data point [True]
49 Text feature [show] present in test data point [True]
51 Text feature [suggesting] present in test data point [True]
52 Text feature [discussion] present in test data point [True]
53 Text feature [containing] present in test data point [True]
54 Text feature [three] present in test data point [True]
55 Text feature [related] present in test data point [True]
56 Text feature [thus] present in test data point [True]
58 Text feature [analysis] present in test data point [True]
59 Text feature [levels] present in test data point [True]
60 Text feature [lower] present in test data point [True]
61 Text feature [30] present in test data point [True]
62 Text feature [introduction] present in test data point [True]
63 Text feature [indicated] present in test data point [True]
64 Text feature [previously] present in test data point [True]
66 Text feature [fact] present in test data point [True]
68 Text feature [purified] present in test data point [True]
69 Text feature [could] present in test data point [True]
70 Text feature [similar] present in test data point [True]
71 Text feature [one] present in test data point [True]
72 Text feature [involved] present in test data point [True]
74 Text feature [contribute] present in test data point [True]
75 Text feature [however] present in test data point [True]
76 Text feature [using] present in test data point [True]
77 Text feature [suggest] present in test data point [True]
78 Text feature [several] present in test data point [True]
79 Text feature [effect] present in test data point [True]
80 Text feature [effects] present in test data point [True]
81 Text feature [affect] present in test data point [True]
82 Text feature [10] present in test data point [True]
83 Text feature [figure] present in test data point [True]
85 Text feature [site] present in test data point [True]
86 Text feature [found] present in test data point [True]
```

```
88 Text feature [result] present in test data point [True]
89 Text feature [phosphatase] present in test data point [True]
90 Text feature [used] present in test data point [True]
91 Text feature [associated] present in test data point [True]
92 Text feature [due] present in test data point [True]
94 Text feature [vivo] present in test data point [True]
95 Text feature [expressed] present in test data point [True]
96 Text feature [role] present in test data point [True]
97 Text feature [indicates] present in test data point [True]
99 Text feature [15] present in test data point [True]
Out of the top  100  features  68 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [101]:

```
test_point_index = 250
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[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_tfidf(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0488 0.089  0.0181 0.05   0.0452 0.0388 0.7015 0.0048 0.0038]]
Actual Class : 2
--------------------------------------------------
16 Text feature [activation] present in test data point [True]
18 Text feature [kinase] present in test data point [True]
19 Text feature [downstream] present in test data point [True]
20 Text feature [cells] present in test data point [True]
21 Text feature [expressing] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [also] present in test data point [True]
25 Text feature [independent] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [treatment] present in test data point [True]
34 Text feature [mutations] present in test data point [True]
35 Text feature [compared] present in test data point [True]
36 Text feature [shown] present in test data point [True]
38 Text feature [10] present in test data point [True]
39 Text feature [however] present in test data point [True]
40 Text feature [cell] present in test data point [True]
41 Text feature [addition] present in test data point [True]
44 Text feature [higher] present in test data point [True]
45 Text feature [similar] present in test data point [True]
46 Text feature [activating] present in test data point [True]
47 Text feature [well] present in test data point [True]
48 Text feature [previously] present in test data point [True]
50 Text feature [inhibitors] present in test data point [True]
51 Text feature [increased] present in test data point [True]
53 Text feature [activate] present in test data point [True]
54 Text feature [showed] present in test data point [True]
55 Text feature [may] present in test data point [True]
56 Text feature [mutant] present in test data point [True]
59 Text feature [found] present in test data point [True]
60 Text feature [presence] present in test data point [True]
63 Text feature [potential] present in test data point [True]
65 Text feature [enhanced] present in test data point [True]
68 Text feature [proliferation] present in test data point [True]
69 Text feature [although] present in test data point [True]
70 Text feature [survival] present in test data point [True]
73 Text feature [phosphorylation] present in test data point [True]
74 Text feature [observed] present in test data point [True]
76 Text feature [respectively] present in test data point [True]
77 Text feature [mutation] present in test data point [True]
```

```
77 Text feature [mutation] present in test data point [True]
78 Text feature [studies] present in test data point [True]
79 Text feature [results] present in test data point [True]
80 Text feature [total] present in test data point [True]
81 Text feature [consistent] present in test data point [True]
82 Text feature [interestingly] present in test data point [True]
83 Text feature [12] present in test data point [True]
84 Text feature [increase] present in test data point [True]
85 Text feature [previous] present in test data point [True]
88 Text feature [two] present in test data point [True]
91 Text feature [using] present in test data point [True]
92 Text feature [discussion] present in test data point [True]
93 Text feature [followed] present in test data point [True]
94 Text feature [serum] present in test data point [True]
95 Text feature [different] present in test data point [True]
96 Text feature [20] present in test data point [True]
97 Text feature [13] present in test data point [True]
98 Text feature [either] present in test data point [True]
Out of the top  100  features  58 are present in query point
```

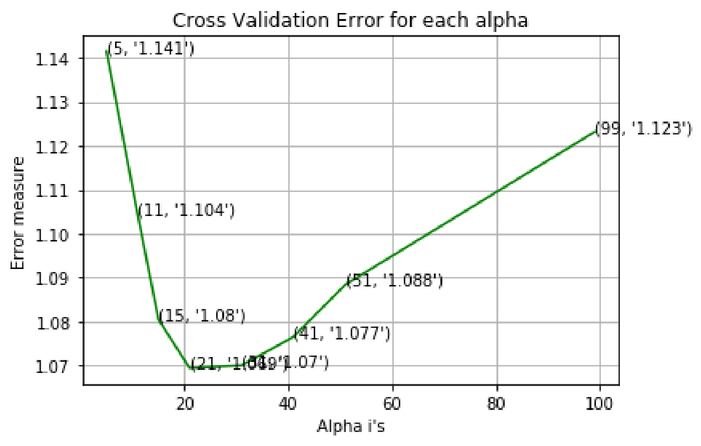
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [102]:

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


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

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

```
for alpha = 5
Log Loss : 1.141397703853408
for alpha = 11
Log Loss : 1.1044660154652124
for alpha = 15
Log Loss : 1.0804763650621667
for alpha = 21
Log Loss : 1.0694473583488184
for alpha = 31
Log Loss : 1.0700616030802657
for alpha = 41
Log Loss : 1.0765341696023512
for alpha = 51
Log Loss : 1.0884095894446488
for alpha = 99
Log Loss : 1.1231679658223603
```



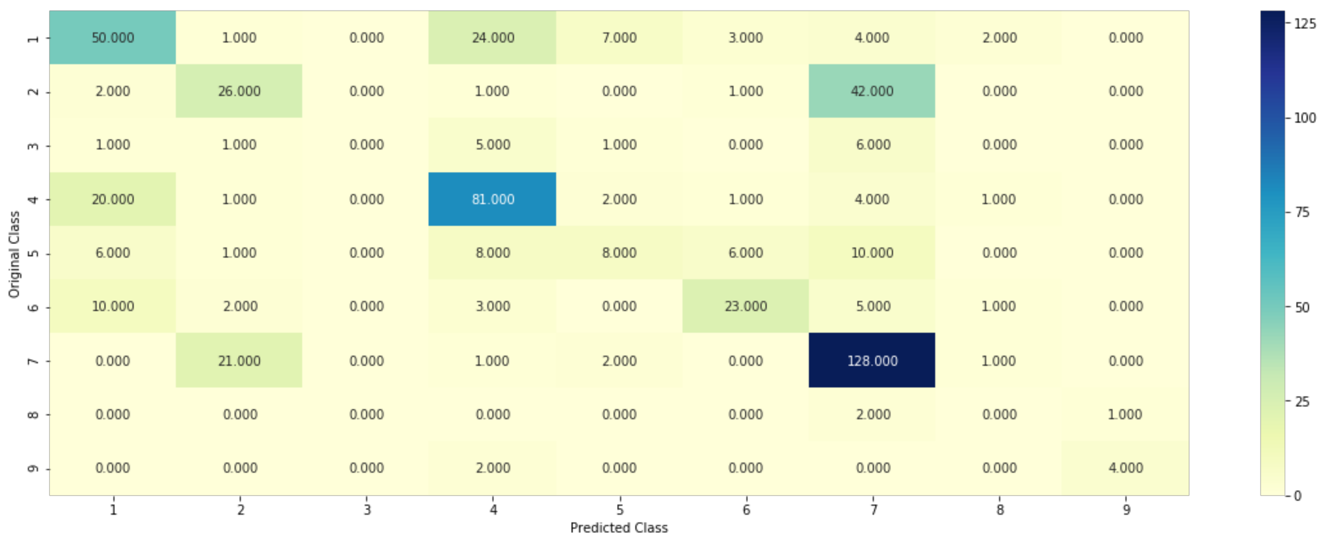
```
For values of best alpha =  21 The train log loss is: 0.7688475527906801
For values of best alpha =  21 The cross validation log loss is: 1.0694473583488184
For values of best alpha =  21 The test log loss is: 1.0733321155303905
```

4.2.2. Testing the model with best hyper paramters

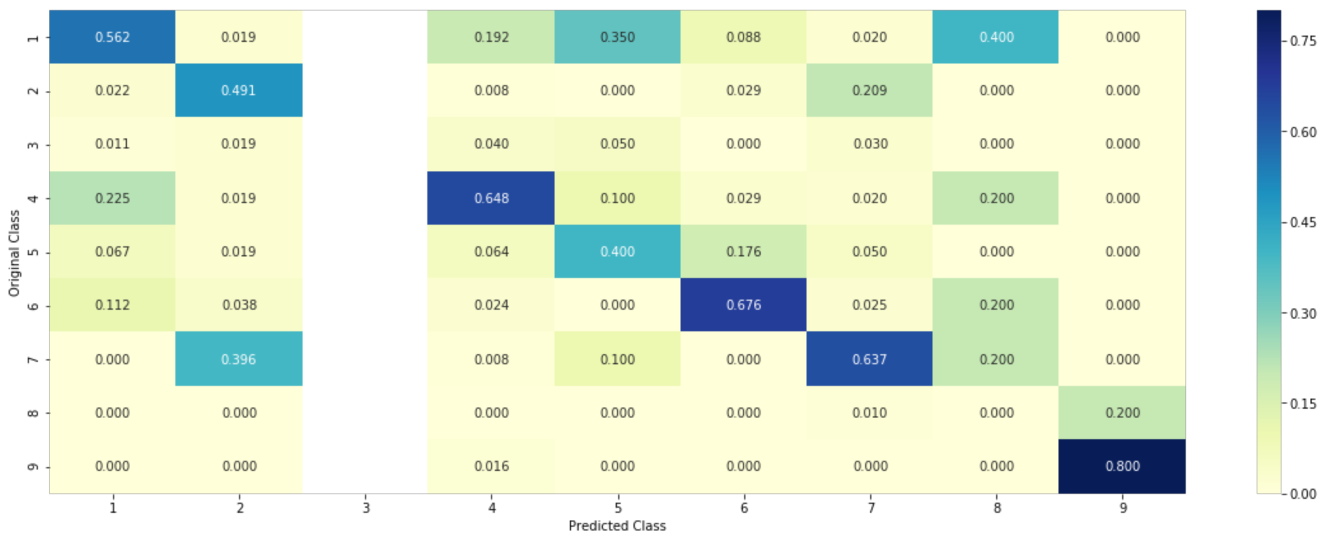
In [103]:

```
# 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-ne
ighbors-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)
```

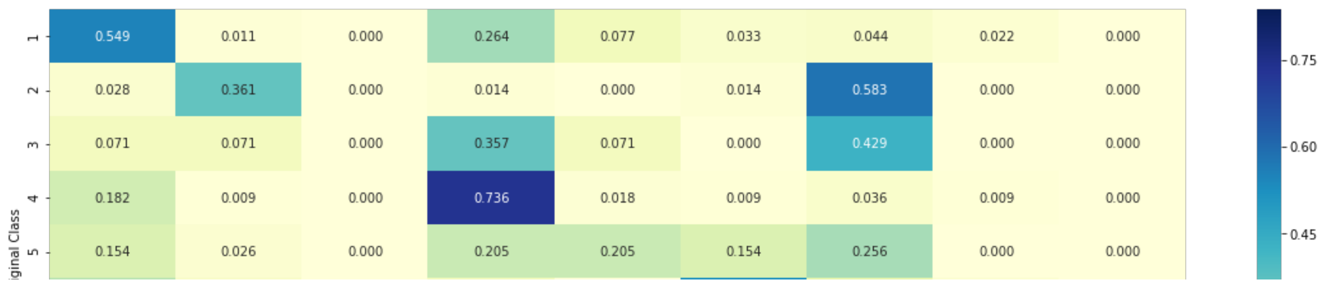
```
Log loss : 1.0694473583488184
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```

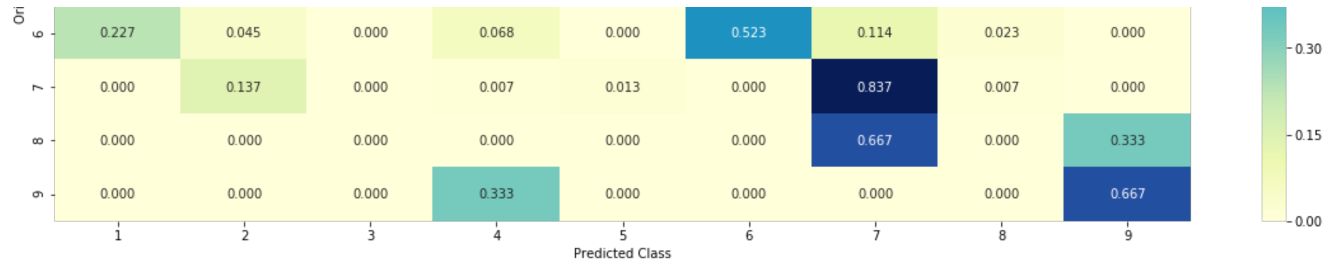


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.2.3.Sample Query point -1

In [104]:

```
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 : 4
The  21  nearest neighbours of the test points belongs to classes [1 4 4 6 4 4 4 4 1 1 1 1 4 1 4 4
1 1 1 4 1]
Fequency of nearest points : Counter({1: 10, 4: 10, 6: 1})
```

4.2.4. Sample Query Point-2

In [107]:

```
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 = 240

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

```
Predicted Class : 7
Actual Class : 3
the k value for knn is 21 and the nearest neighbours of the test points belongs to classes [3 7 7
7 7 7 7 7 7 7 5 7 7 5 7 7 7 7 7 7 2]
Fequency of nearest points : Counter({7: 17, 5: 2, 3: 1, 2: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

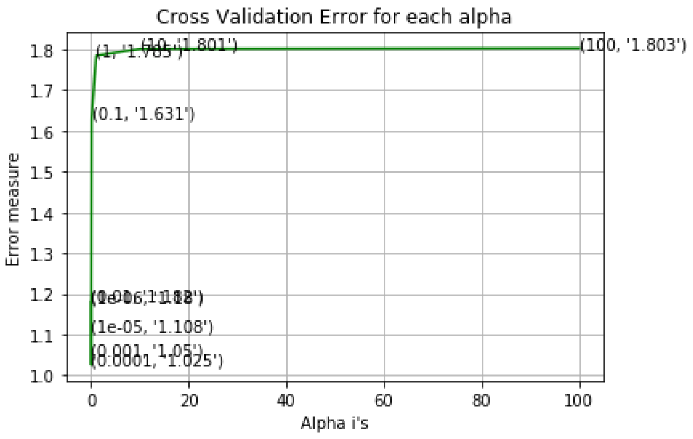
4.3.1.1. Hyper paramter tuning

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_i
ter=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-in
tuition-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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
```

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

```
for alpha = 1e-06
Log Loss : 1.1803450479593718
for alpha = 1e-05
Log Loss : 1.1084754165237434
for alpha = 0.0001
Log Loss : 1.0252700565486053
for alpha = 0.001
Log Loss : 1.0503796535307397
for alpha = 0.01
Log Loss : 1.1817493747361103
for alpha = 0.1
Log Loss : 1.6307349261509914
for alpha = 1
Log Loss : 1.7848675195217225
for alpha = 10
Log Loss : 1.8013364842811541
for alpha = 100
Log Loss : 1.8030696316287114
```



```
For values of best alpha =  0.0001 The train log loss is: 0.5856416566287704
For values of best alpha =  0.0001 The cross validation log loss is: 1.0252700565486053
For values of best alpha =  0.0001 The test log loss is: 1.011939929011602
```

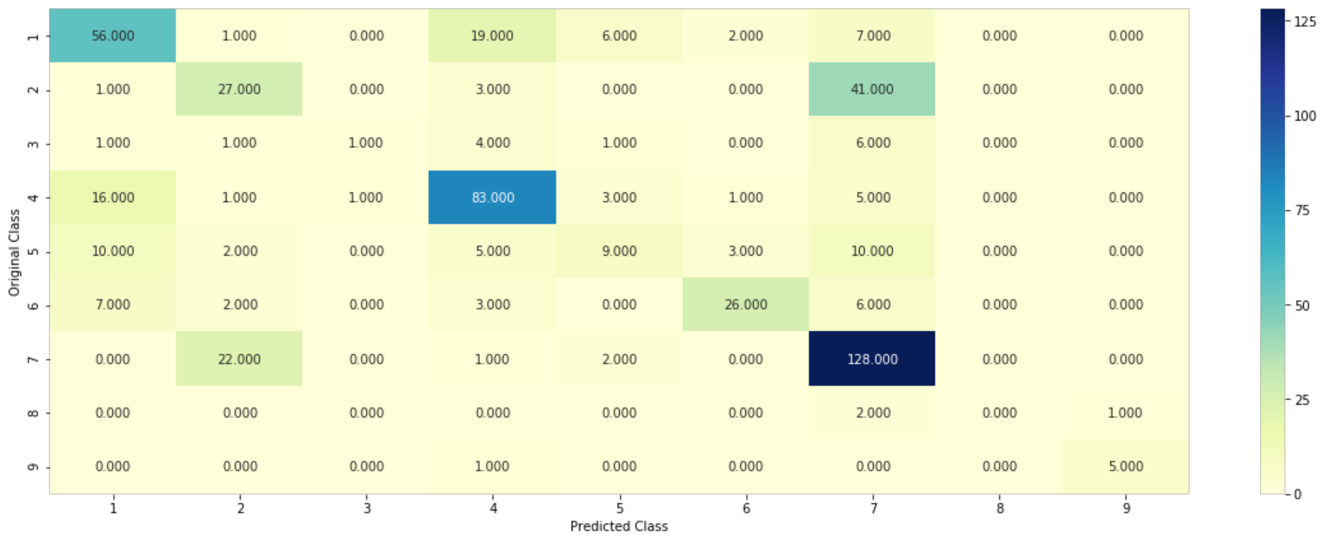
4.3.1.2. Testing the model with best hyper paramters

In [109]:

```
# 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_i
ter=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-in
tuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

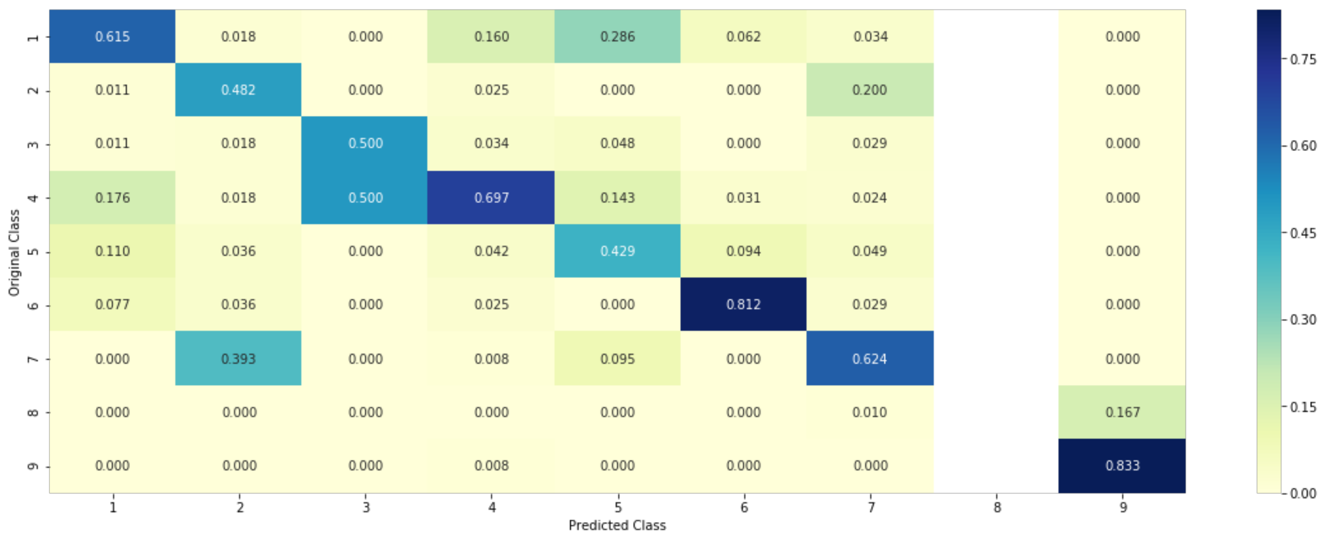
```
Log loss : 1.0252700565486053
Number of mis-classified points : 0.37030075187969924
```

```
-------------------- Confusion matrix --------------------
```



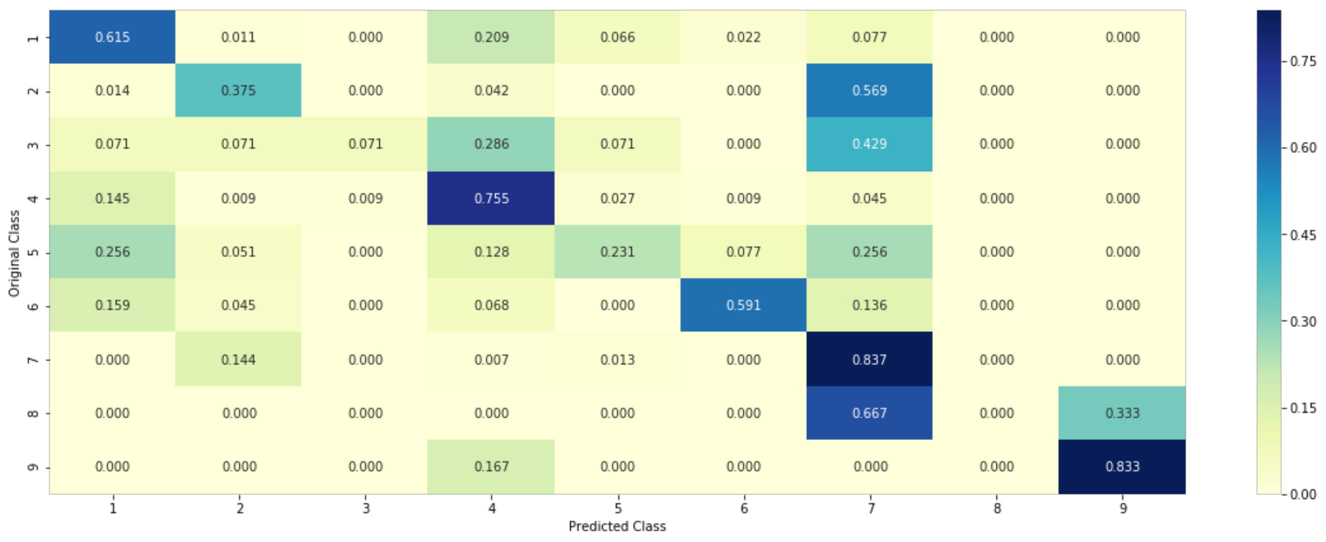
	1	2	3	4	5	6	7	8	9
1	56.000	1.000	0.000	19.000	6.000	2.000	7.000	0.000	0.000
2	1.000	27.000	0.000	3.000	0.000	0.000	41.000	0.000	0.000
3	1.000	1.000	1.000	4.000	1.000	0.000	6.000	0.000	0.000
4	16.000	1.000	1.000	83.000	3.000	1.000	5.000	0.000	0.000
5	10.000	2.000	0.000	5.000	9.000	3.000	10.000	0.000	0.000
6	7.000	2.000	0.000	3.000	0.000	26.000	6.000	0.000	0.000
7	0.000	22.000	0.000	1.000	2.000	0.000	128.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000
9	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.615	0.018	0.000	0.160	0.286	0.062	0.034		0.000
2	0.011	0.482	0.000	0.025	0.000	0.000	0.200		0.000
3	0.011	0.018	0.500	0.034	0.048	0.000	0.029		0.000
4	0.176	0.018	0.500	0.697	0.143	0.031	0.024		0.000
5	0.110	0.036	0.000	0.042	0.429	0.094	0.049		0.000
6	0.077	0.036	0.000	0.025	0.000	0.812	0.029		0.000
7	0.000	0.393	0.000	0.008	0.095	0.000	0.624		0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.010		0.167
9	0.000	0.000	0.000	0.008	0.000	0.000	0.000		0.833

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



	1	2	3	4	5	6	7	8	9
1	0.615	0.011	0.000	0.209	0.066	0.022	0.077	0.000	0.000
2	0.014	0.375	0.000	0.042	0.000	0.000	0.569	0.000	0.000
3	0.071	0.071	0.071	0.286	0.071	0.000	0.429	0.000	0.000
4	0.145	0.009	0.009	0.755	0.027	0.009	0.045	0.000	0.000
5	0.256	0.051	0.000	0.128	0.231	0.077	0.256	0.000	0.000
6	0.159	0.045	0.000	0.068	0.000	0.591	0.136	0.000	0.000
7	0.000	0.144	0.000	0.007	0.013	0.000	0.837	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.667	0.000	0.333
9	0.000	0.000	0.000	0.167	0.000	0.000	0.000	0.000	0.833

4.3.1.3. Feature Importance

```
In [110]:
```

```
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_tfidf.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [111]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3334 0.0818 0.0031 0.4995 0.02   0.0199 0.0215 0.0155 0.0052]]
Actual Class : 4
--------------------------------------------------
315 Text feature [00001] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [115]:

```
test_point_index = 250
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0249 0.2125 0.0045 0.0086 0.0533 0.0203 0.6726 0.0023 0.001 ]]
Actual Class : 2
--------------------------------------------------
```

```
423 Text feature [0001] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2. Without Class balancing

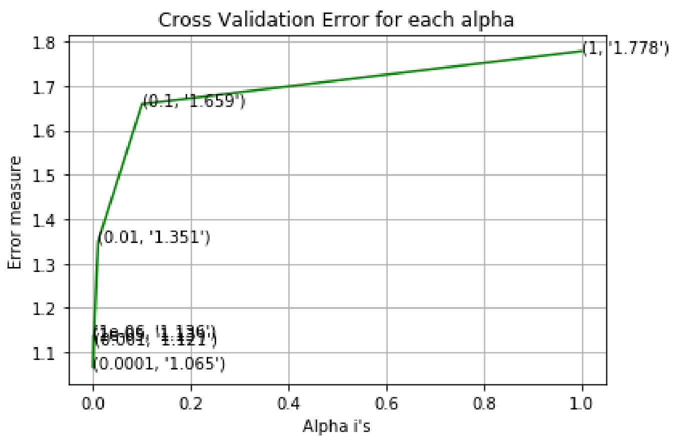
4.3.2.1. Hyper paramter tuning

In [116]:

```
# 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_i
ter=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-in
tuition-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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

```
predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.136079025153073
for alpha = 1e-05
Log Loss : 1.1296592413758024
for alpha = 0.0001
Log Loss : 1.0647513697775808
for alpha = 0.001
Log Loss : 1.121206076011605
for alpha = 0.01
Log Loss : 1.351217364405122
for alpha = 0.1
Log Loss : 1.6592992443180818
for alpha = 1
Log Loss : 1.7782550054817006
```



```
For values of best alpha =  0.0001 The train log loss is: 0.5695515574861976
For values of best alpha =  0.0001 The cross validation log loss is: 1.0647513697775808
For values of best alpha =  0.0001 The test log loss is: 1.0457711946823909
```

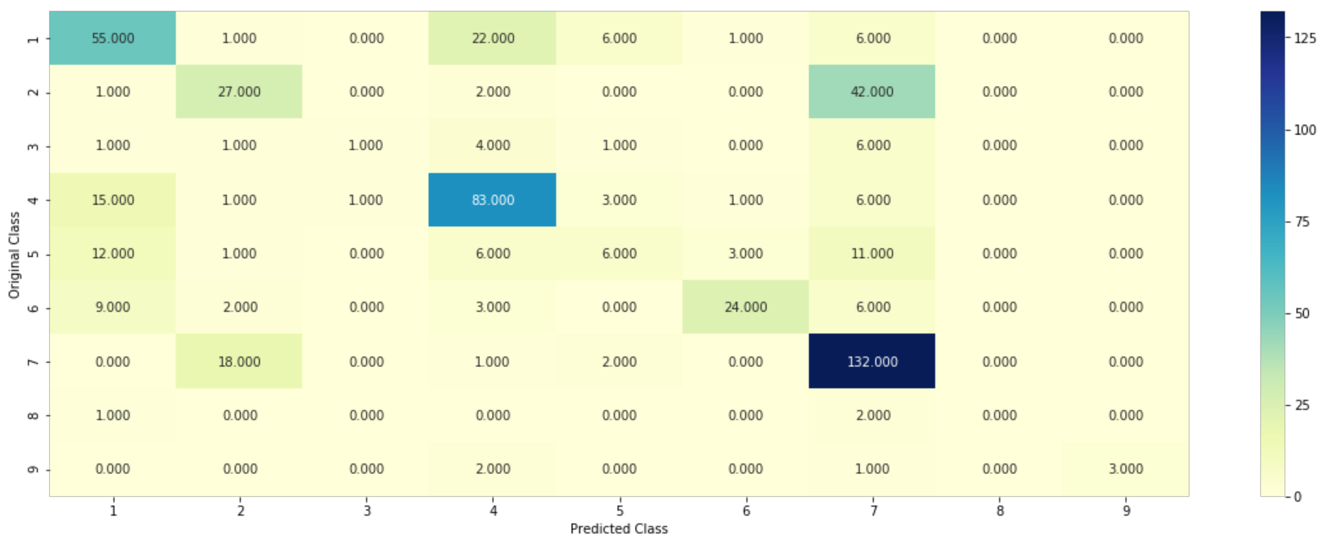
4.3.2.2. Testing model with best hyper parameters

In [117]:

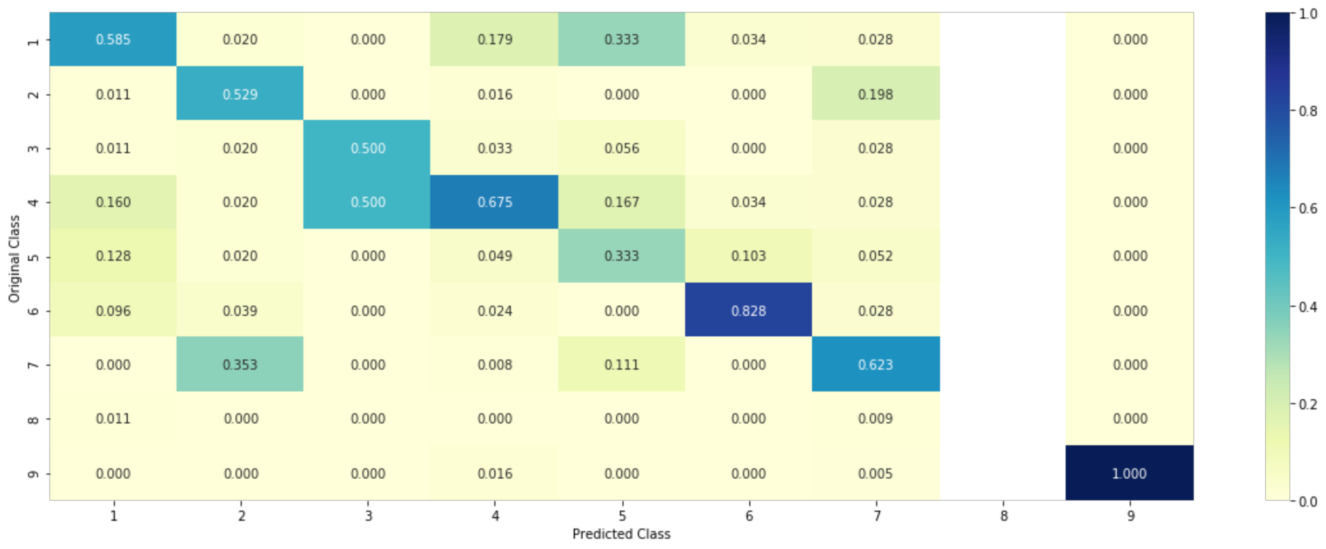
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

```
Log loss : 1.0647513697775808
```

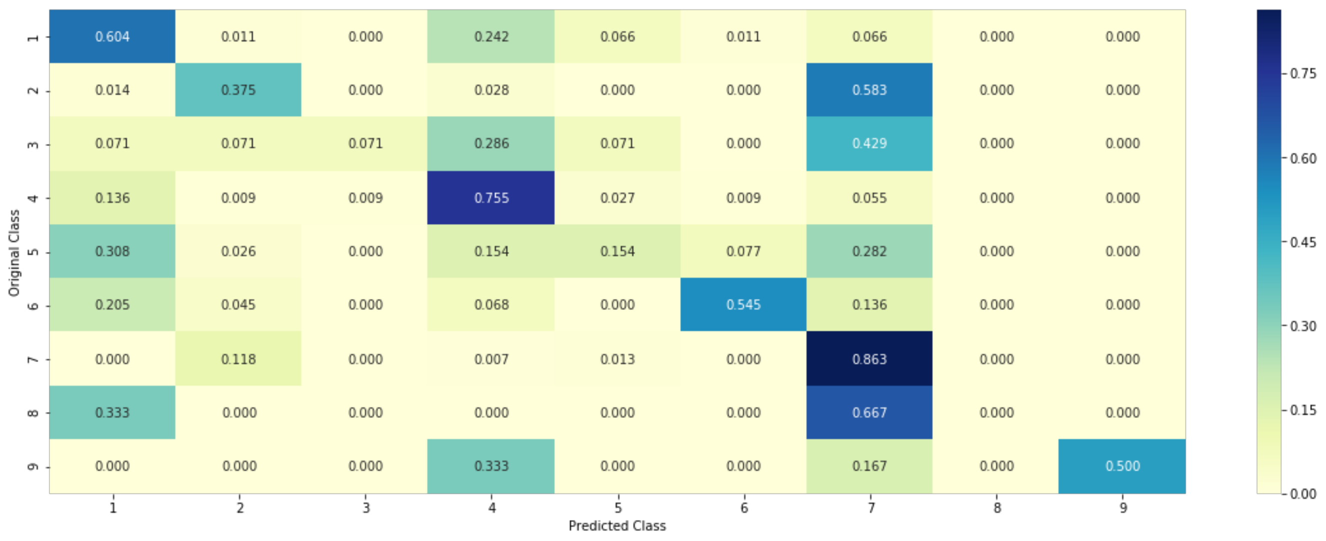
```
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [118]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3147 0.0861 0.0015 0.5312 0.0194 0.0189 0.0187 0.0065 0.0029]]
Actual Class : 4
--------------------------------------------------
319 Text feature [00001] 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 [120]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 250
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.390e-02 1.898e-01 1.300e-03 7.300e-03 4.750e-02 1.560e-02 7.131
e-01
  1.000e-03 4.000e-04]]
Actual Class : 2
--------------------------------------------------
318 Text feature [0001] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.4. Linear Support Vector Machines

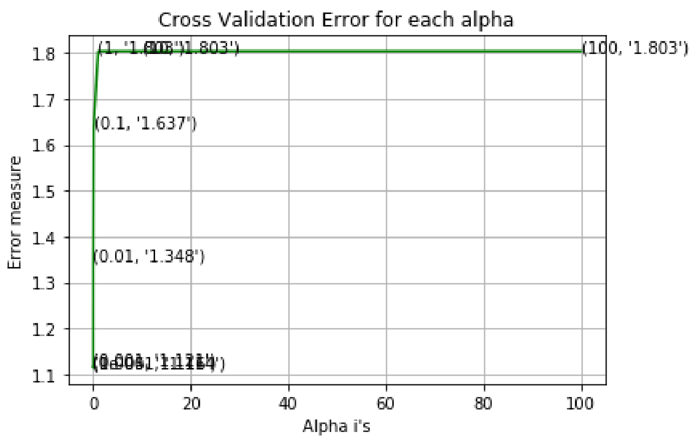
4.4.1. Hyper paramter tuning

In [121]:

```
# 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, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', r
andom_state=42)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.1160770924047878
for C = 0.0001
Log Loss : 1.113782642651394
for C = 0.001
Log Loss : 1.121473670538308
for C = 0.01
Log Loss : 1.3484141091513078
for C = 0.1
```

```
Log Loss : 1.6369083271152276
for C = 1
Log Loss : 1.8032947065111697
for C = 10
Log Loss : 1.8032802477690308
for C = 100
Log Loss : 1.8032803802552055
```



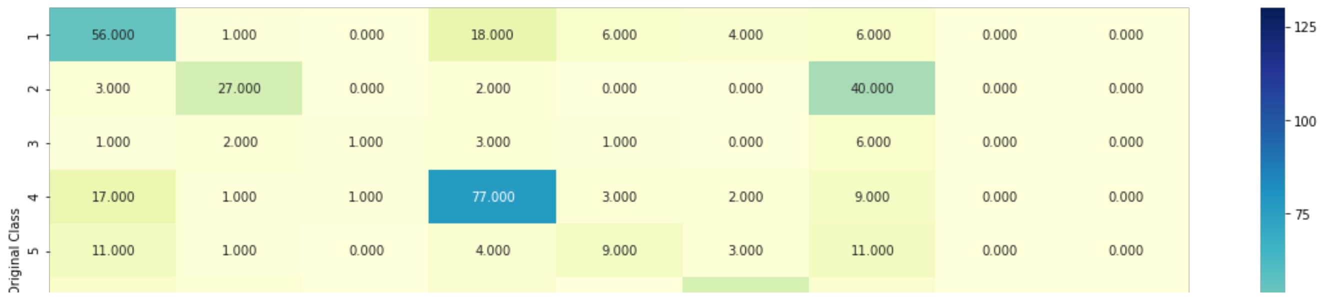
```
For values of best alpha =  0.0001 The train log loss is: 0.670019854350629
For values of best alpha =  0.0001 The cross validation log loss is: 1.113782642651394
For values of best alpha =  0.0001 The test log loss is: 1.1115828496149405
```

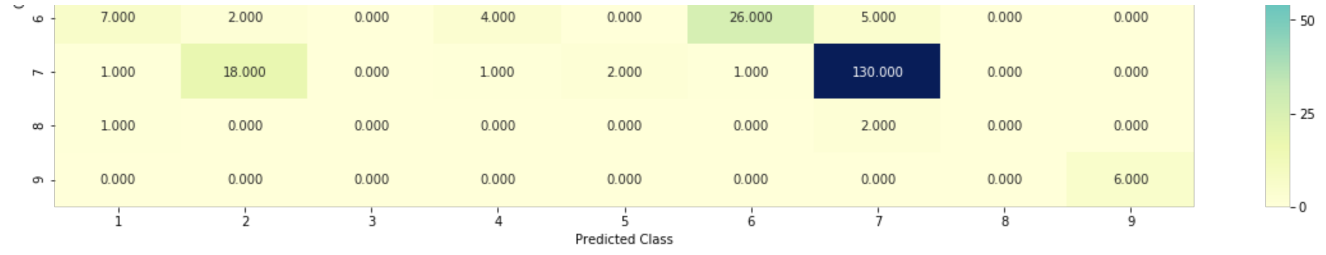
4.4.2. Testing model with best hyper parameters

In [122]:

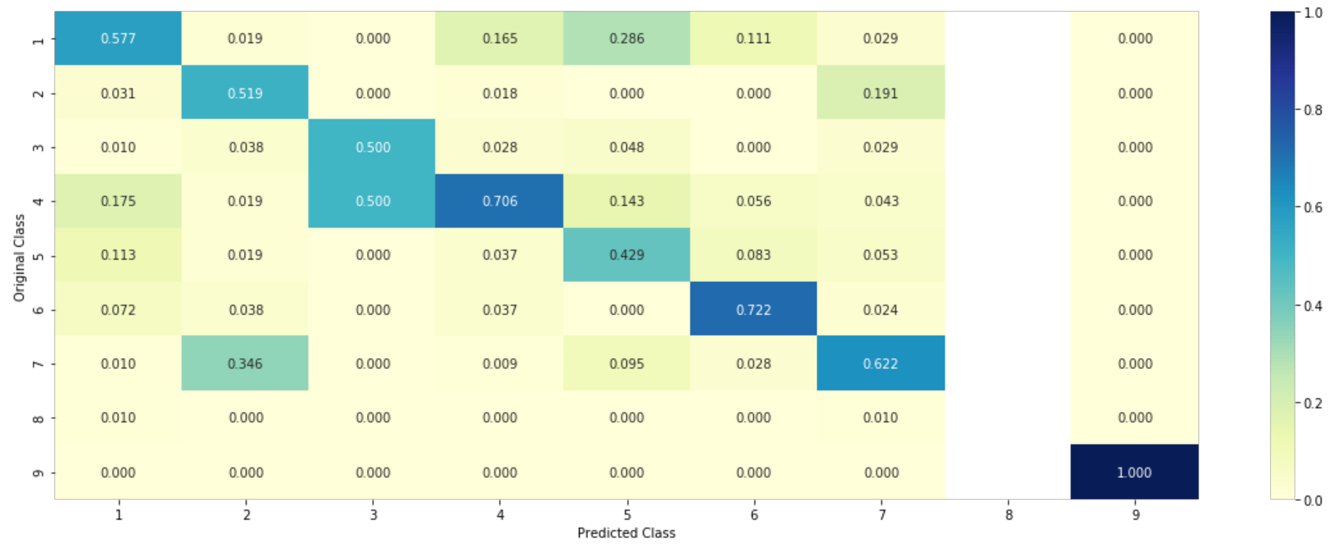
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_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/
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y, clf)
```

```
Log loss : 1.113782642651394
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

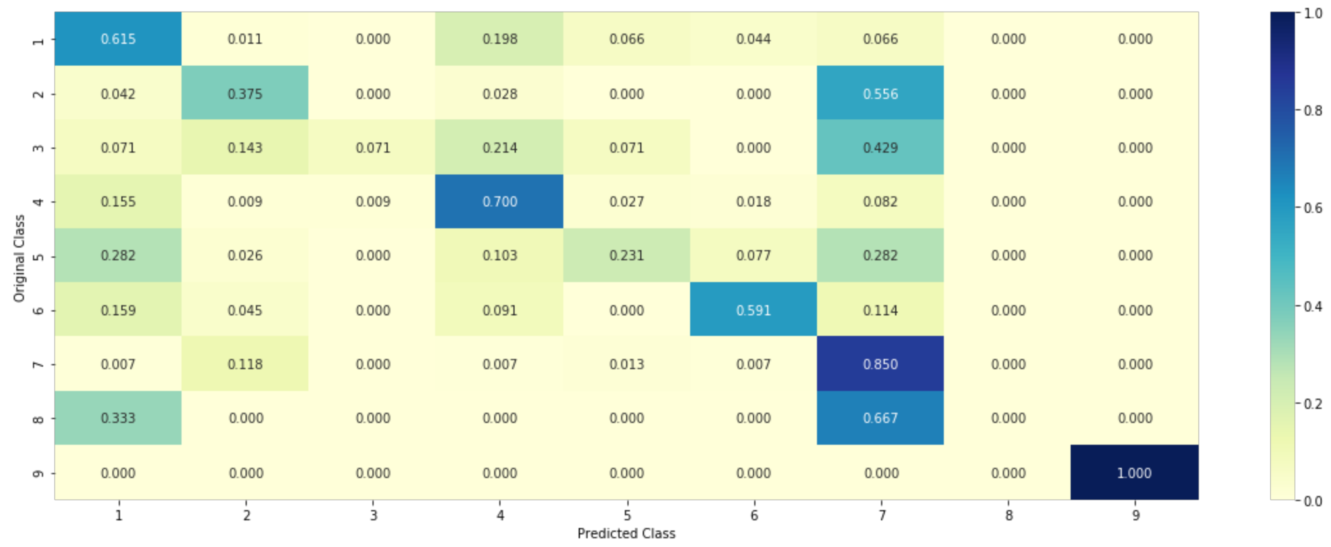




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [123]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]),4))
```

```
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1985 0.1787 0.004  0.5065 0.0508 0.0226 0.0205 0.0137 0.0047]]
Actual Class : 4
--------------------------------------------------
425 Text feature [00001] 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 [124]:

```
test_point_index = 250
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0418 0.1354 0.0089 0.017  0.0449 0.0164 0.7322 0.0021 0.0012]]
Actual Class : 2
--------------------------------------------------
299 Text feature [0001] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [125]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_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-fores
t-and-their-construction-2/
# --------------------------------
```

```python
# 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 = [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_tfidf, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidf, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
        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)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "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_tfidf)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "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_tfidf)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

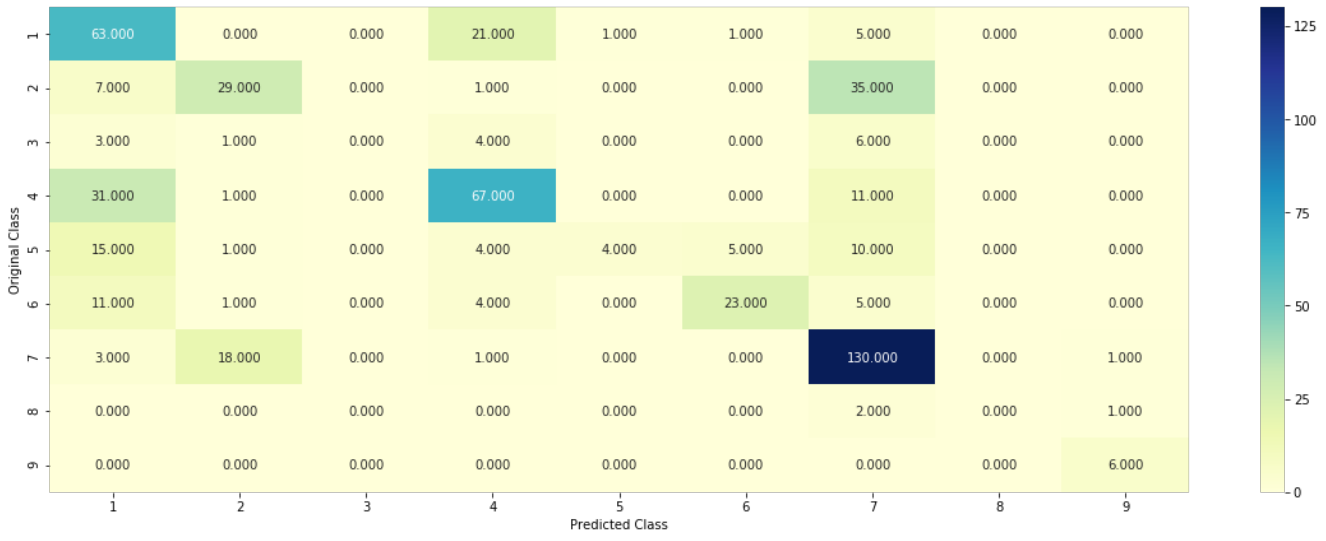
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.173702465111509
for n_estimators = 100 and max depth =  10
Log Loss : 1.2207390843977843
for n_estimators = 200 and max depth =  5
Log Loss : 1.1603654773888652
for n_estimators = 200 and max depth =  10
Log Loss : 1.2150782196195966
for n_estimators = 500 and max depth =  5
Log Loss : 1.1528381490395339
for n_estimators = 500 and max depth =  10
Log Loss : 1.2129972225729317
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1537001420116748
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2080377023471511
Log Loss : 1.2080377022471511
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1524862659313755
for n_estimators = 2000 and max depth =  10
Log Loss : 1.206036613249754
For values of best estimator =  2000 The train log loss is: 0.8545256188406025
For values of best estimator =  2000 The cross validation log loss is: 1.152486265931376
For values of best estimator =  2000 The test log loss is: 1.2222948395420883
```

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

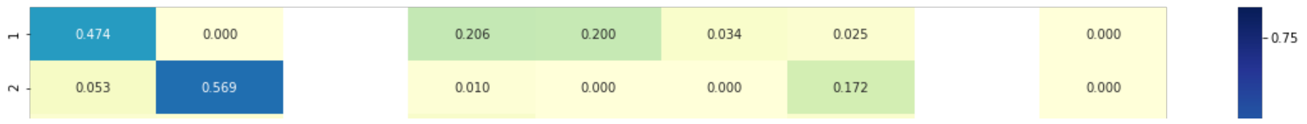
In [126]:

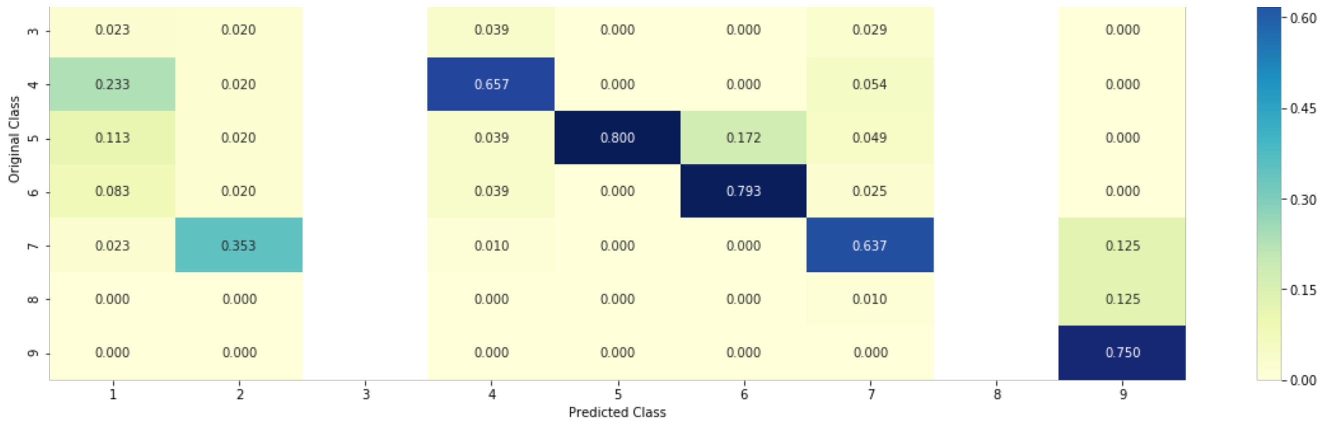
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, 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-fores
t-and-their-construction-2/
# --------------------------------

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)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y, clf)
```

```
Log loss : 1.152486265931376
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```

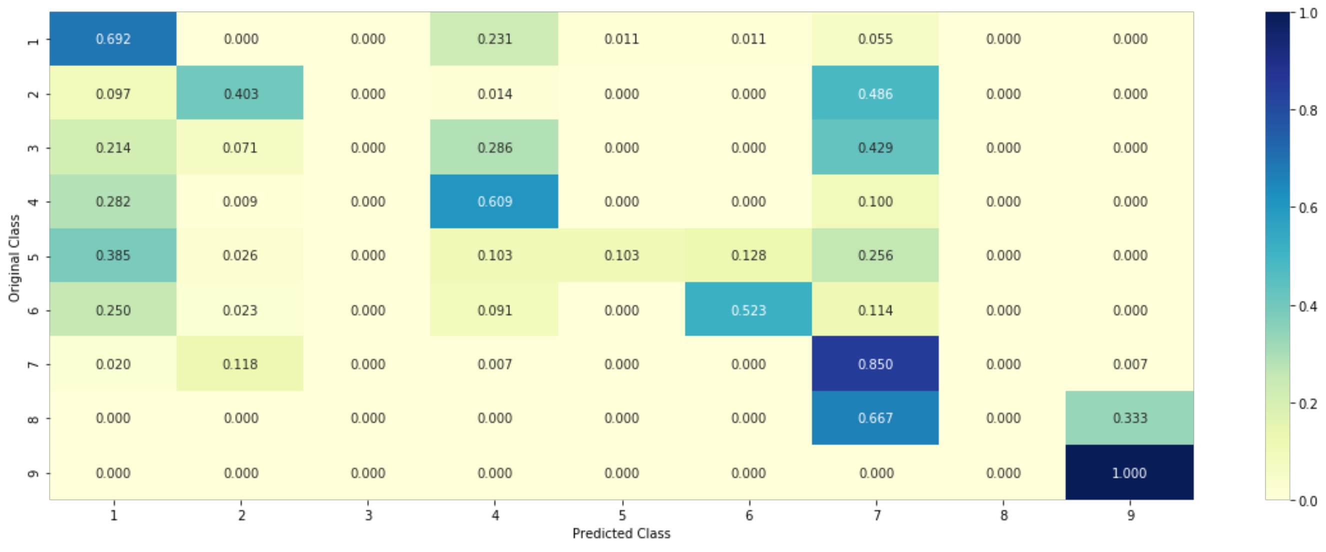


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [127]:

```
# 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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0965 0.1279 0.0262 0.1457 0.0686 0.0653 0.4613 0.0052 0.0031]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
5 Text feature [activated] present in test data point [True]
```

```
6 Text feature [activation] present in test data point [True]
7 Text feature [tyrosine] present in test data point [True]
8 Text feature [function] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
14 Text feature [oncogenic] present in test data point [True]
32 Text feature [cells] present in test data point [True]
33 Text feature [constitutively] present in test data point [True]
37 Text feature [receptor] present in test data point [True]
42 Text feature [cell] present in test data point [True]
48 Text feature [ba] present in test data point [True]
49 Text feature [proteins] present in test data point [True]
51 Text feature [f3] present in test data point [True]
58 Text feature [proliferation] present in test data point [True]
61 Text feature [expression] present in test data point [True]
63 Text feature [extracellular] present in test data point [True]
71 Text feature [oncogene] present in test data point [True]
81 Text feature [assays] present in test data point [True]
85 Text feature [dna] present in test data point [True]
89 Text feature [lines] present in test data point [True]
94 Text feature [type] present in test data point [True]
Out of the top  100  features  23 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [128]:

```
test_point_index = 250
no_feature = 200
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_tfidf(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.096  0.2257 0.0297 0.0704 0.0622 0.0563 0.4498 0.0063 0.0037]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitors] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
6 Text feature [activation] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [loss] present in test data point [True]
11 Text feature [treatment] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
17 Text feature [therapy] present in test data point [True]
18 Text feature [akt] present in test data point [True]
22 Text feature [trials] present in test data point [True]
25 Text feature [variants] present in test data point [True]
27 Text feature [protein] present in test data point [True]
31 Text feature [activate] present in test data point [True]
32 Text feature [cells] present in test data point [True]
34 Text feature [functional] present in test data point [True]
35 Text feature [pten] present in test data point [True]
39 Text feature [signaling] present in test data point [True]
40 Text feature [growth] present in test data point [True]
42 Text feature [cell] present in test data point [True]
44 Text feature [therapeutic] present in test data point [True]
49 Text feature [proteins] present in test data point [True]
50 Text feature [phosphatase] present in test data point [True]
52 Text feature [downstream] present in test data point [True]
55 Text feature [patients] present in test data point [True]
58 Text feature [proliferation] present in test data point [True]
59 Text feature [clinical] present in test data point [True]
60 Text feature [efficacy] present in test data point [True]
61 Text feature [expression] present in test data point [True]
64 Text feature [survival] present in test data point [True]
68 Text feature [advanced] present in test data point [True]
```

```
72 Text feature [activity] present in test data point [True]
74 Text feature [ras] present in test data point [True]
78 Text feature [ovarian] present in test data point [True]
79 Text feature [tagged] present in test data point [True]
81 Text feature [assays] present in test data point [True]
85 Text feature [dna] present in test data point [True]
86 Text feature [expressing] present in test data point [True]
89 Text feature [lines] present in test data point [True]
94 Text feature [type] present in test data point [True]
95 Text feature [conserved] present in test data point [True]
99 Text feature [pi3k] present in test data point [True]
100 Text feature [information] present in test data point [True]
104 Text feature [phospho] present in test data point [True]
106 Text feature [affect] present in test data point [True]
107 Text feature [21] present in test data point [True]
108 Text feature [variant] present in test data point [True]
109 Text feature [binding] present in test data point [True]
110 Text feature [classification] present in test data point [True]
112 Text feature [phosphorylated] present in test data point [True]
114 Text feature [testing] present in test data point [True]
115 Text feature [patient] present in test data point [True]
119 Text feature [sequence] present in test data point [True]
122 Text feature [mammalian] present in test data point [True]
123 Text feature [ability] present in test data point [True]
124 Text feature [one] present in test data point [True]
127 Text feature [potential] present in test data point [True]
129 Text feature [serum] present in test data point [True]
130 Text feature [genes] present in test data point [True]
131 Text feature [terminal] present in test data point [True]
133 Text feature [cancer] present in test data point [True]
134 Text feature [sequencing] present in test data point [True]
136 Text feature [based] present in test data point [True]
141 Text feature [vector] present in test data point [True]
142 Text feature [american] present in test data point [True]
143 Text feature [effective] present in test data point [True]
144 Text feature [interaction] present in test data point [True]
145 Text feature [evidence] present in test data point [True]
146 Text feature [transfected] present in test data point [True]
147 Text feature [effect] present in test data point [True]
149 Text feature [wild] present in test data point [True]
155 Text feature [26] present in test data point [True]
159 Text feature [known] present in test data point [True]
160 Text feature [results] present in test data point [True]
161 Text feature [breast] present in test data point [True]
162 Text feature [database] present in test data point [True]
163 Text feature [independent] present in test data point [True]
165 Text feature [multiple] present in test data point [True]
167 Text feature [mutants] present in test data point [True]
168 Text feature [well] present in test data point [True]
169 Text feature [gene] present in test data point [True]
170 Text feature [mutant] present in test data point [True]
173 Text feature [published] present in test data point [True]
174 Text feature [likely] present in test data point [True]
175 Text feature [increased] present in test data point [True]
180 Text feature [controls] present in test data point [True]
184 Text feature [pathway] present in test data point [True]
185 Text feature [deletion] present in test data point [True]
186 Text feature [presence] present in test data point [True]
189 Text feature [tumors] present in test data point [True]
192 Text feature [specific] present in test data point [True]
194 Text feature [31] present in test data point [True]
197 Text feature [defined] present in test data point [True]
198 Text feature [given] present in test data point [True]
Out of the top  200  features  95 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [129]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_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-fores
t-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='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 = [10,50,100,200,500,1000]
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)], 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)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "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[int(best_alpha/4)], "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_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y
test, predict_y, labels=clf.classes_, eps=1e-15))
```

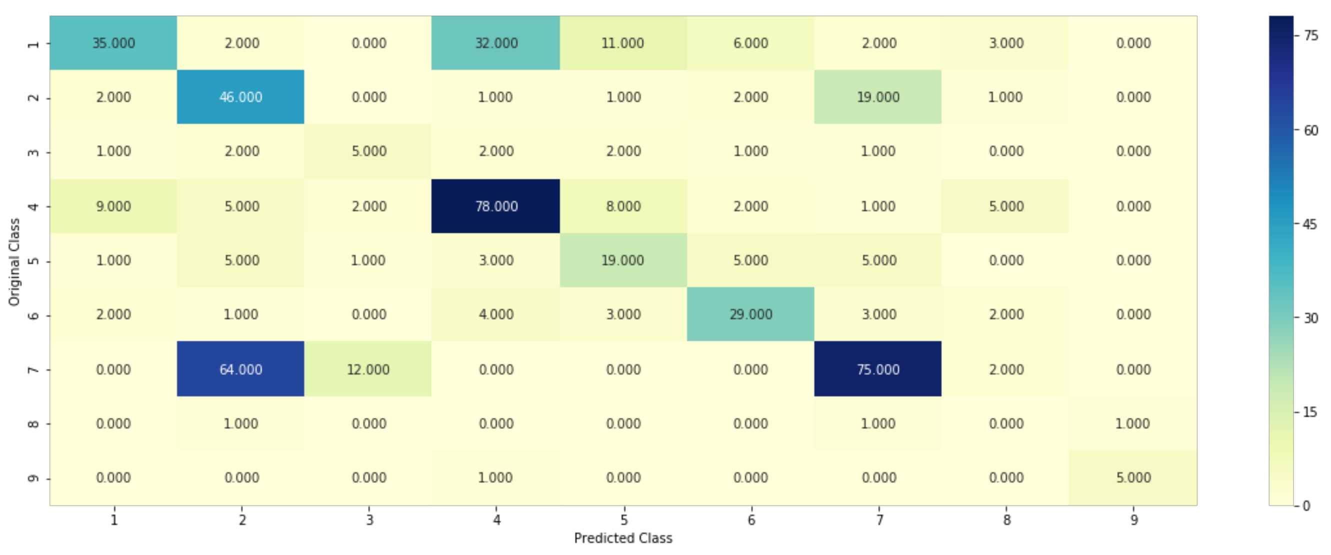
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.046276617356074
for n_estimators = 10 and max depth =  3
Log Loss : 1.6753862003879836
for n_estimators = 10 and max depth =  5
Log Loss : 1.484141566108487
for n_estimators = 10 and max depth =  10
Log Loss : 2.0150156888348416
for n_estimators = 50 and max depth =  2
Log Loss : 1.6672917738558573
for n_estimators = 50 and max depth =  3
Log Loss : 1.4182034365977667
for n_estimators = 50 and max depth =  5
Log Loss : 1.3765719603474647
for n_estimators = 50 and max depth =  10
Log Loss : 1.5924099796639501
for n_estimators = 100 and max depth =  2
Log Loss : 1.5118565494938596
for n_estimators = 100 and max depth =  3
Log Loss : 1.4469701943195254
for n_estimators = 100 and max depth =  5
Log Loss : 1.2746349522925766
for n_estimators = 100 and max depth =  10
Log Loss : 1.674618178170568
for n_estimators = 200 and max depth =  2
Log Loss : 1.5483265850663843
for n_estimators = 200 and max depth =  3
Log Loss : 1.4325320453883807
for n_estimators = 200 and max depth =  5
Log Loss : 1.3380735708141775
for n_estimators = 200 and max depth =  10
Log Loss : 1.6627669079768008
for n_estimators = 500 and max depth =  2
Log Loss : 1.6043069982986211
for n_estimators = 500 and max depth =  3
Log Loss : 1.4704250454373198
for n_estimators = 500 and max depth =  5
Log Loss : 1.310790829039582
for n_estimators = 500 and max depth =  10
Log Loss : 1.6782920421300112
for n_estimators = 1000 and max depth =  2
Log Loss : 1.596014358283692
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4858258792571533
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2961812732564235
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6693862813618774
For values of best alpha =  100 The train log loss is: 0.05555314204551388
For values of best alpha =  100 The cross validation log loss is: 1.2746349522925773
For values of best alpha =  100 The test log loss is: 1.3120815832748898
```

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

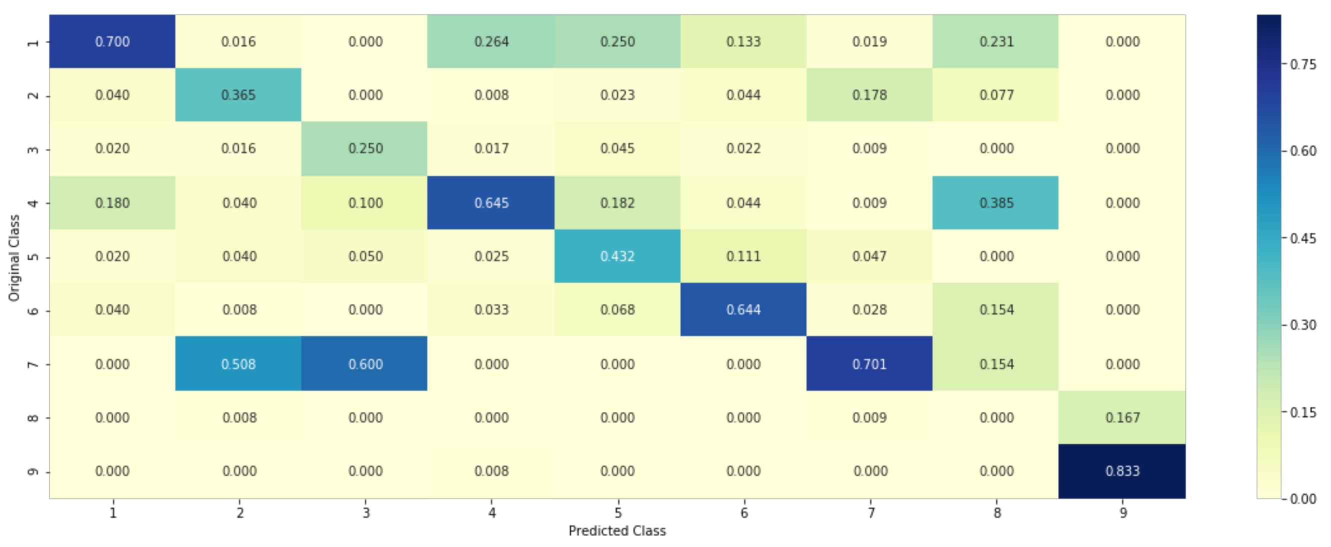
In [130]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_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()
# 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-fores
t-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)
```

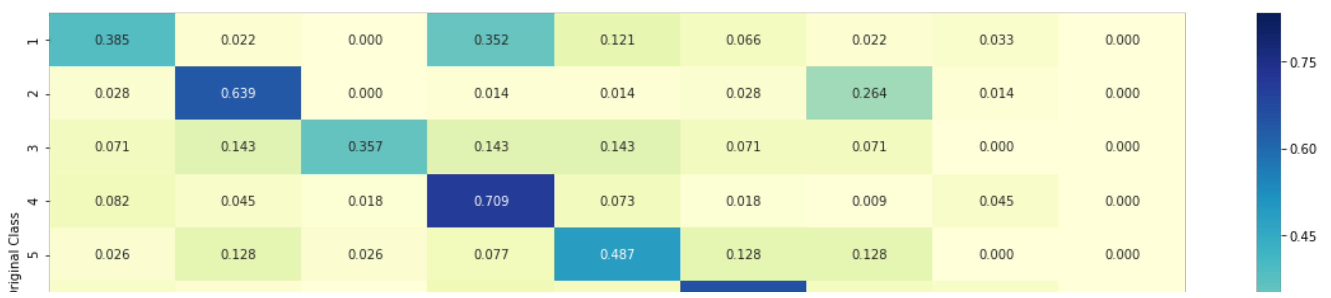
```
Log loss : 1.274634952292577
Number of mis-classified points : 0.45112781954887216
-------------------- Confusion matrix --------------------
```

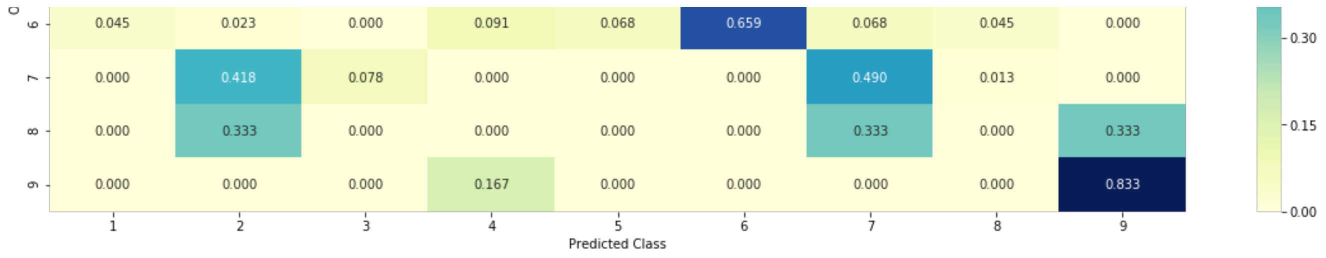


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [131]:

```
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 = 100
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_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: [[0.0123 0.5783 0.0734 0.0181 0.0224 0.0361 0.2318 0.0202 0.0074]]
Actual Class : 7
--------------------------------------------------
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
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene 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
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [132]:

```
test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_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: [[0.0082 0.0034 0.0022 0.9673 0.0022 0.0034 0.0025 0.0062 0.0046]]
Actual Class : 4
--------------------------------------------------
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
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene 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
Text is important feature
Variation is important feature
Gene 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 [133]:

```
# 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_i
ter=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-in
tuition-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, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_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_s
amples_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-fores
t-and-their-construction-2/
# --------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0
)
clf1.fit(train_x_tfidf, 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_tfidf, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

sig_clf1.fit(train_x_tfidf, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_tfidf)
)))
sig_clf2.fit(train_x_tfidf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_tfidf))))
sig_clf3.fit(train_x_tfidf, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tfidf))))
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_p
robas=True)
    sclf.fit(train_x_tfidf, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_tfidf))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.80
Naive Bayes : Log Loss: 1.15
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.506
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.123
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.138
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.269
```

In [134]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_tfidf, train_y)
print("Stacking Classifer : Test data Log Loss: %0.3f" % ( log_loss(test_y,
sclf.predict_proba(test_x_tfidf))))
```

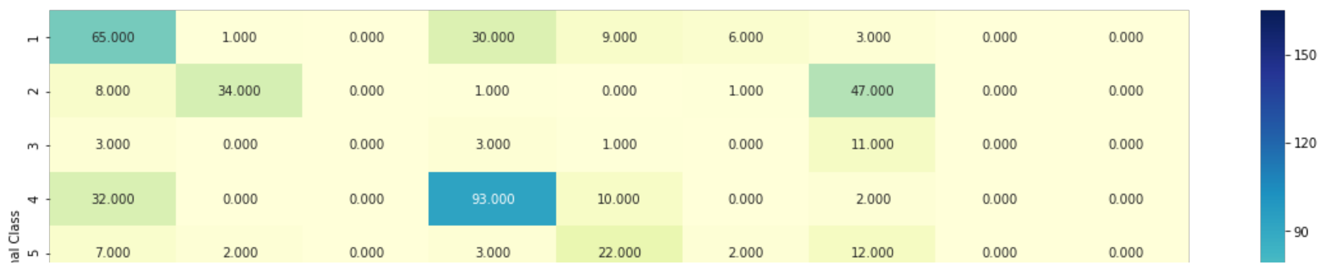
```
Stacking Classifer : Test data Log Loss: 1.153
```

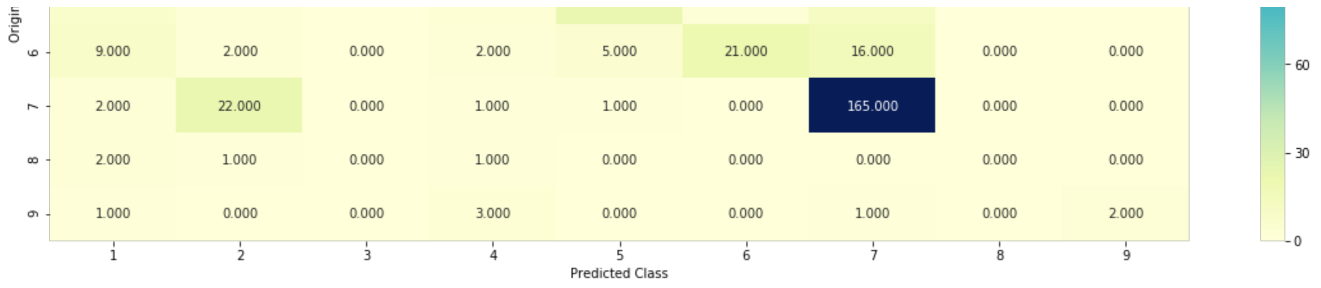
4.7.2 testing the model with the best hyper parameters

In [135]:

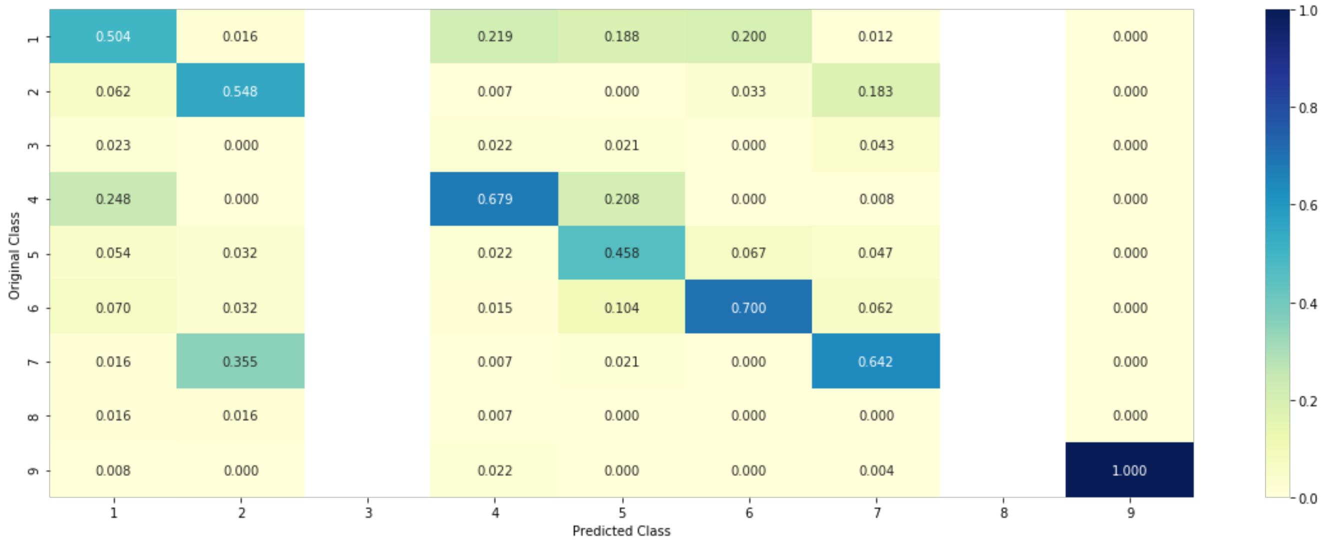
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_tfidf, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.7935596820461525
Log loss (CV) on the stacking classifier : 1.1228910641158187
Log loss (test) on the stacking classifier : 1.1532263318750275
Number of missclassified point : 0.3954887218045113
-------------------- Confusion matrix --------------------
```

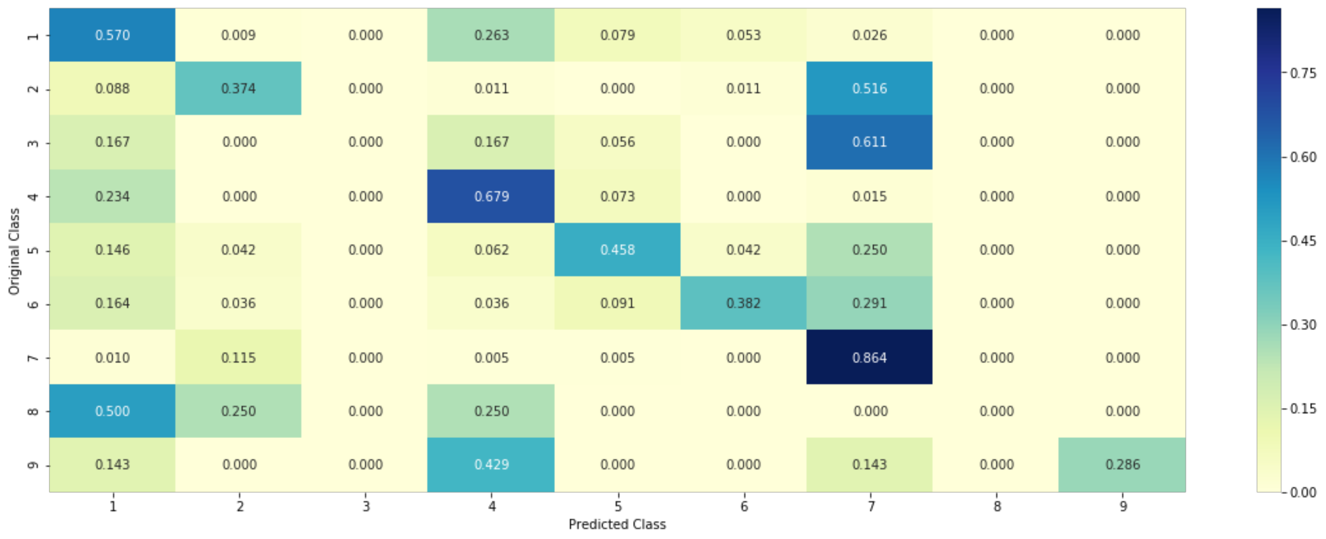




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

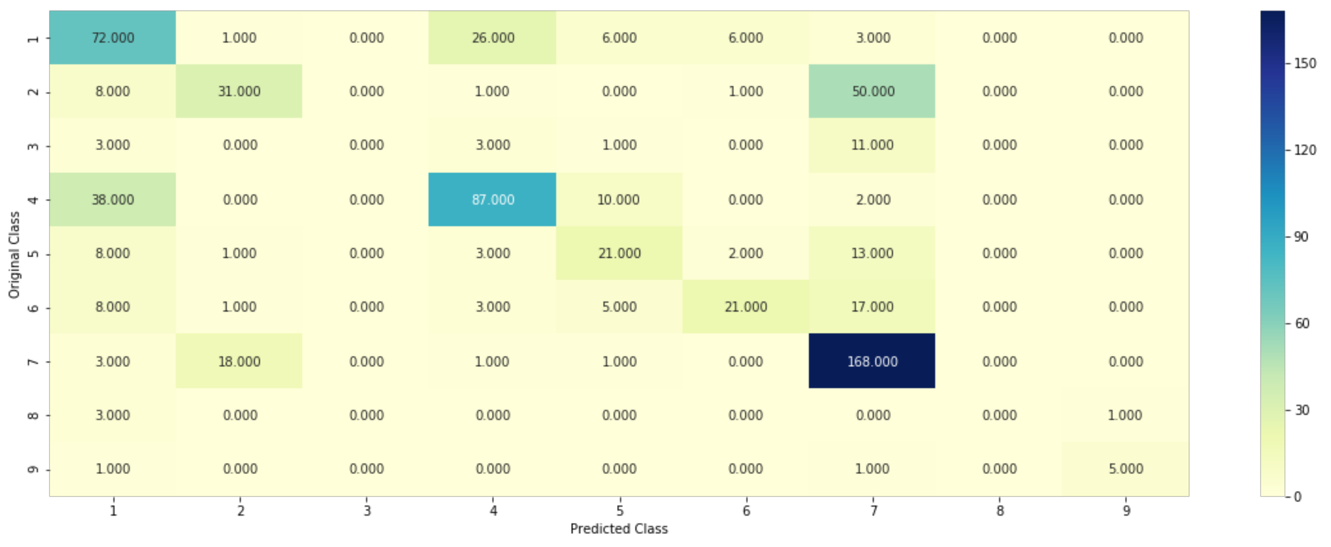


4.7.3 Maximum Voting classifier

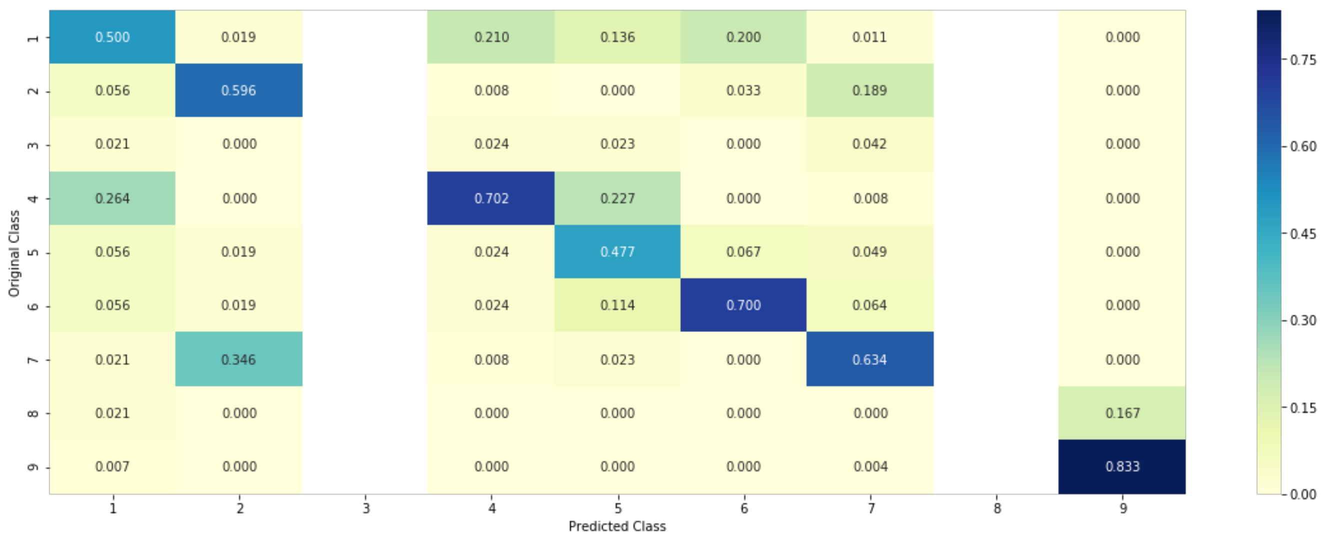
In [136]:

```
#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_tfidf, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_tfidf)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_tfidf)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
vclf.predict_proba(test_x_tfidf)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfidf)- test_y))/t
est_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidf))
```

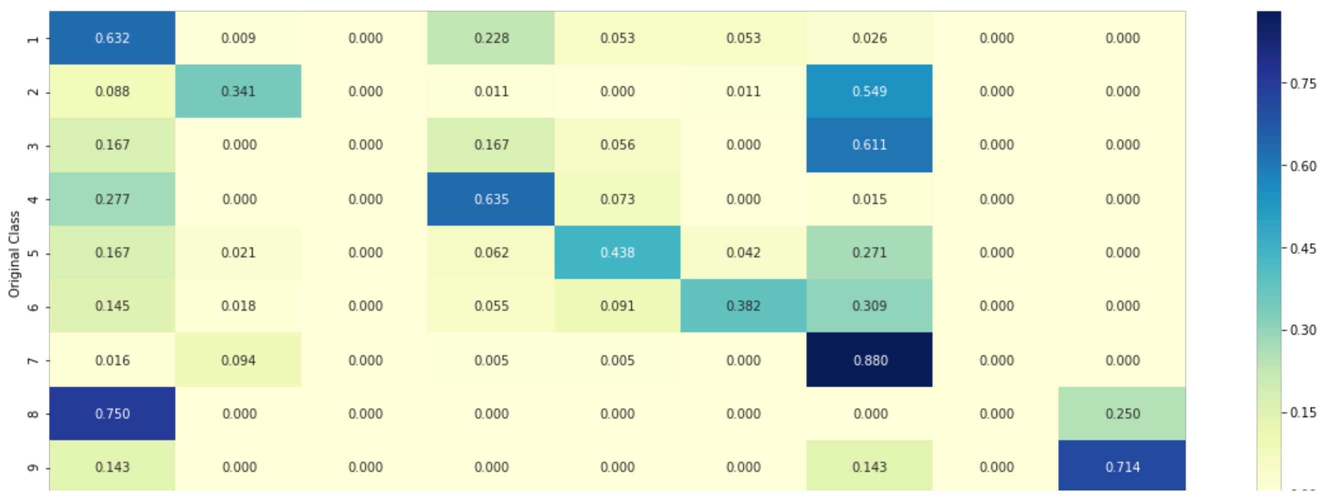
```
Log loss (train) on the VotingClassifier : 0.9393402542648616
Log loss (CV) on the VotingClassifier : 1.1728395528489513
Log loss (test) on the VotingClassifier : 1.192971534994921
Number of missclassified point : 0.39097744360902253
-------------------- Confusion matrix --------------------
```

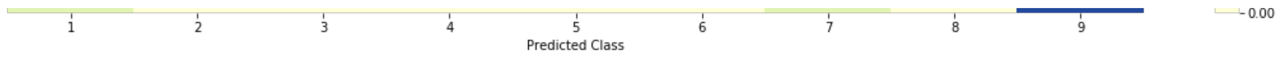


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





5. Assignments

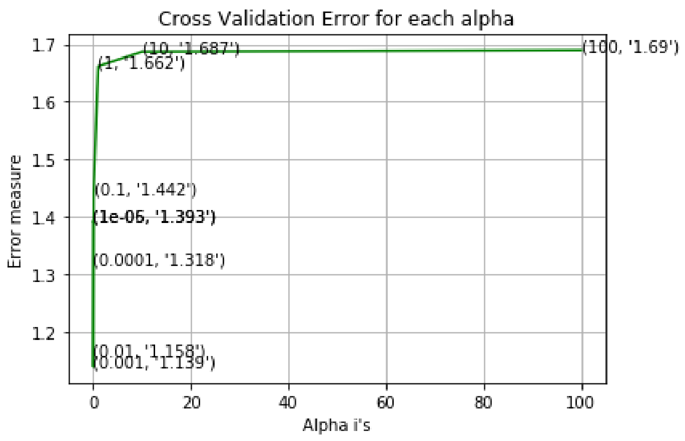
5.1 Logistic Regression with uni grams

5.1.1 With class balancing

In [137]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3926004079512482
for alpha = 1e-05
Log Loss : 1.3929321310774372
for alpha = 0.0001
Log Loss : 1.3175983709656844
for alpha = 0.001
Log Loss : 1.138519272360989
for alpha = 0.01
Log Loss : 1.1576912775992727
for alpha = 0.1
Log Loss : 1.4415403214297742
for alpha = 1
Log Loss : 1.6620282695410549
for alpha = 10
Log Loss : 1.6873691765272463
for alpha = 100
Log Loss : 1.6899457238688105
```

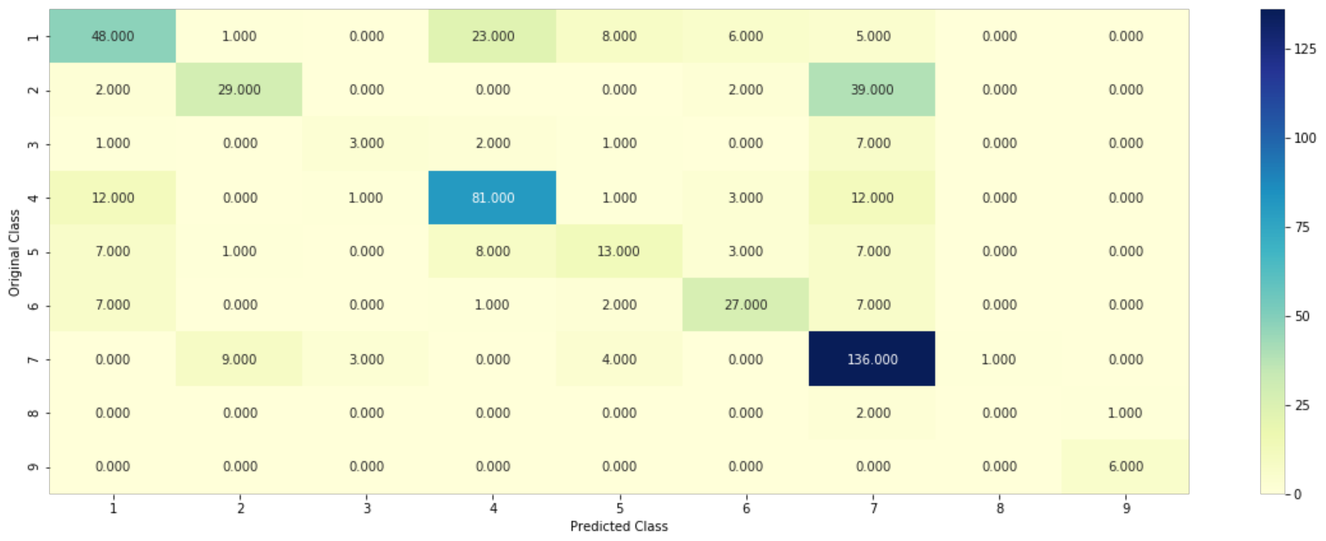


```
For values of best alpha =  0.001 The train log loss is: 0.596262281582009
For values of best alpha =  0.001 The cross validation log loss is: 1.138519272360989
For values of best alpha =  0.001 The test log loss is: 1.1424523388585621
```

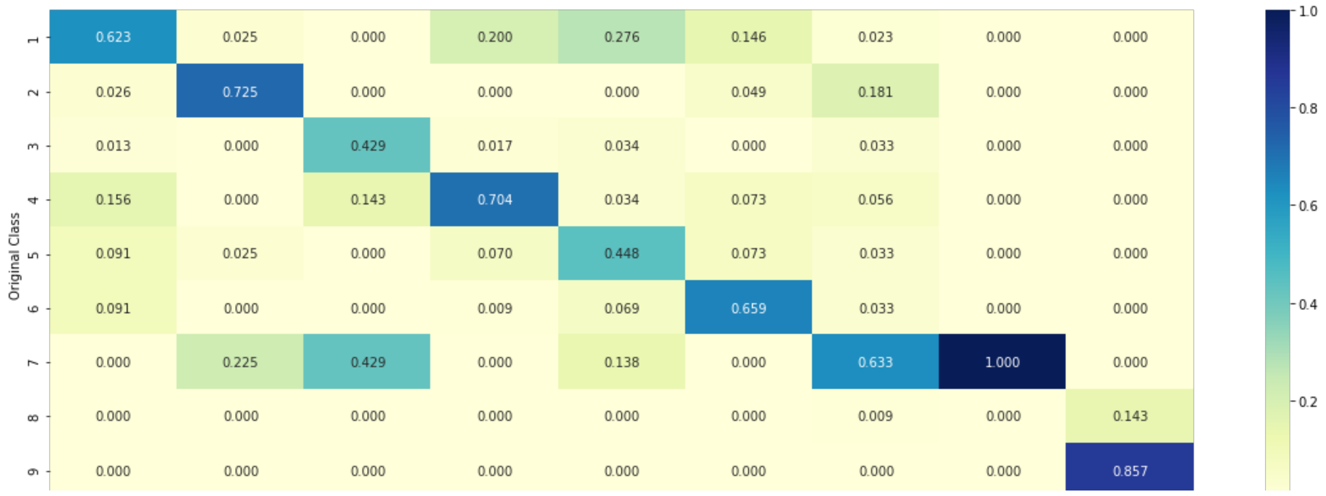
In [138]:

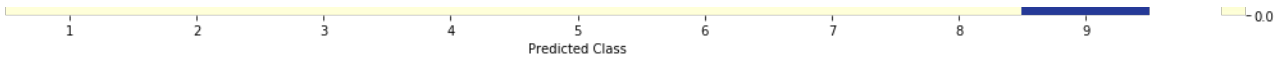
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.138519272360989
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```

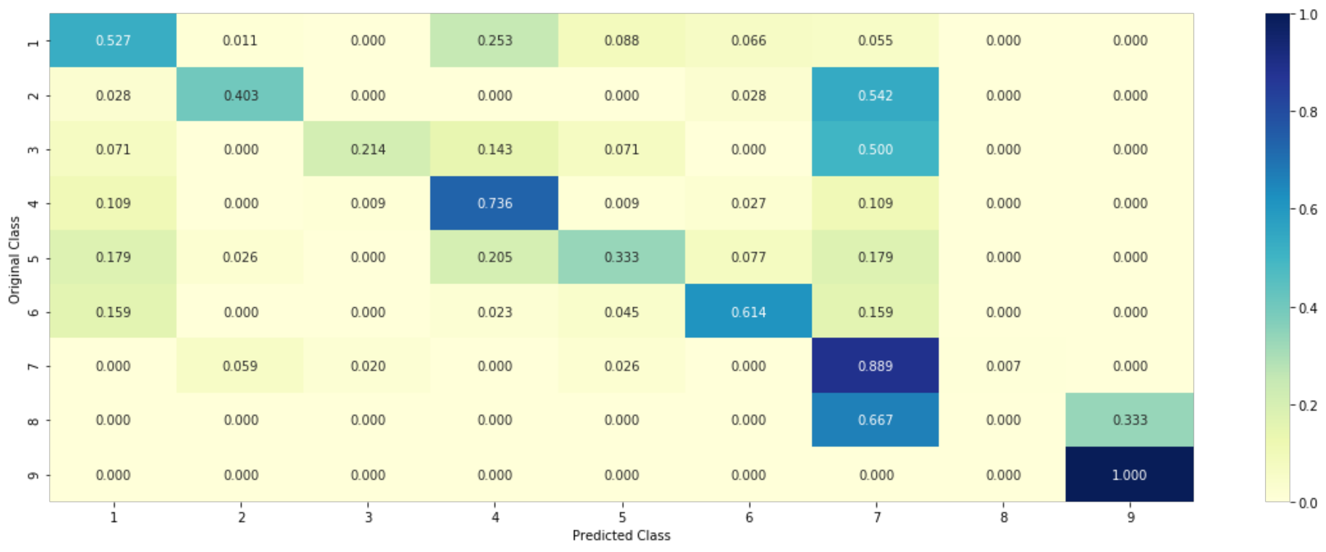


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



In [139]:

```python
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

In [141]:

```python
# 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()

    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 [{}]".format(word,yes_no))
                print(i, "Gene feature [{}] present in test data point [{}]".format(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_r
o))
        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 [{}]".format(word,yes_no))

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

In [142]:

```
# from tabulate import tabulate
#feature importance
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_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_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3137 0.1944 0.0172 0.1271 0.0436 0.0155 0.2536 0.0157 0.0192]]
Actual Class : 4
--------------------------------------------------
214 Text feature [ptprm] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

In [143]:

```
test_point_index = 300
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_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0202 0.0142 0.0046 0.0073 0.0076 0.0025 0.9351 0.0039 0.0046]]
Actual Class : 7
--------------------------------------------------
247 Text feature [rbd] present in test data point [True]
251 Text feature [constitutive] present in test data point [True]
256 Text feature [mitogen] present in test data point [True]
263 Text feature [upstate] present in test data point [True]
280 Text feature [hras] present in test data point [True]
344 Text feature [transforming] present in test data point [True]
354 Text feature [ligand] present in test data point [True]
368 Text feature [phospho] present in test data point [True]
412 Text feature [activated] present in test data point [True]
418 Text feature [downstream] present in test data point [True]
461 Text feature [oncoproteins] present in test data point [True]
```

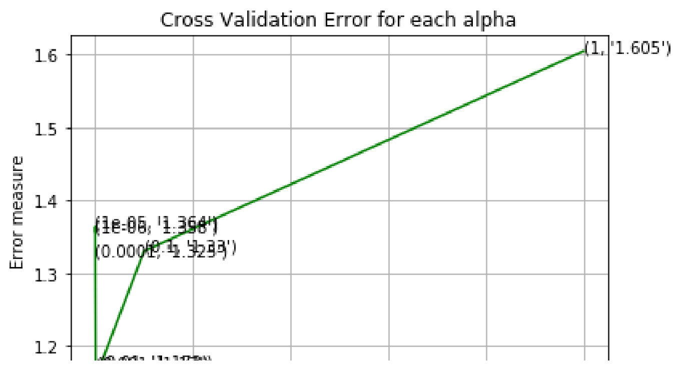
```
498 Text feature [tyr204] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

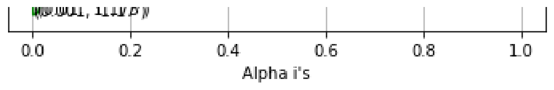
5.1.2 without balancing

In [145]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3576694526902746
for alpha = 1e-05
Log Loss : 1.3639598382364768
for alpha = 0.0001
Log Loss : 1.3254494403930361
for alpha = 0.001
Log Loss : 1.1700854093920483
for alpha = 0.01
Log Loss : 1.1730066266391692
for alpha = 0.1
Log Loss : 1.3304897839805476
for alpha = 1
Log Loss : 1.6049633127983167
```



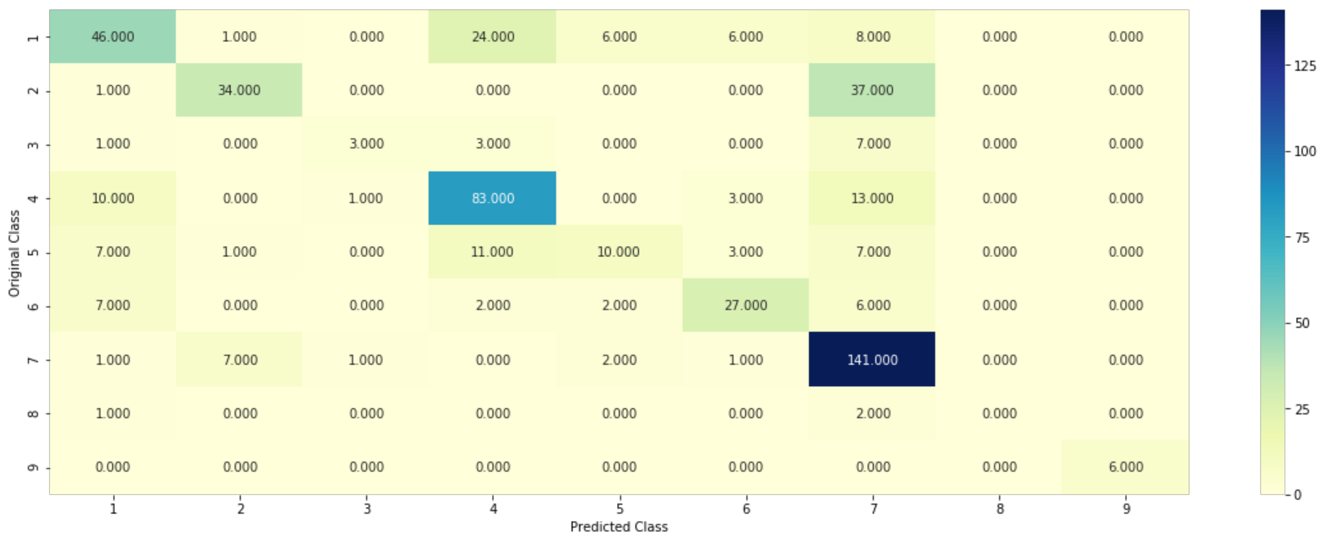


```
For values of best alpha =  0.001 The train log loss is: 0.5884637830402645
For values of best alpha =  0.001 The cross validation log loss is: 1.1700854093920483
For values of best alpha =  0.001 The test log loss is: 1.1536315505014978
```

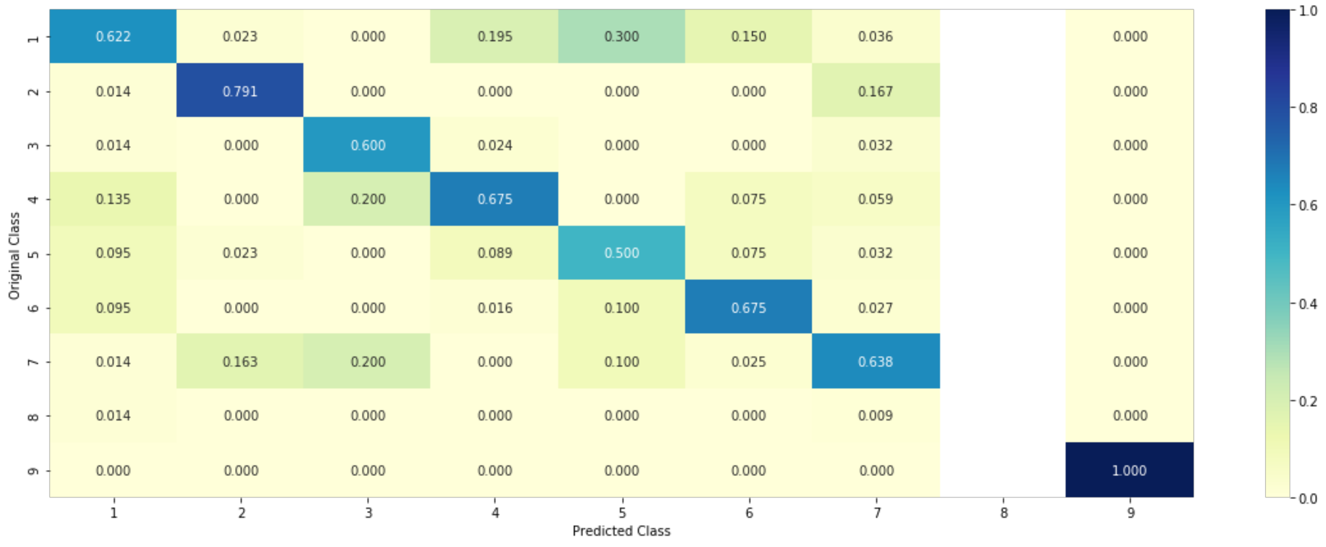
In [146]:

```
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)
```

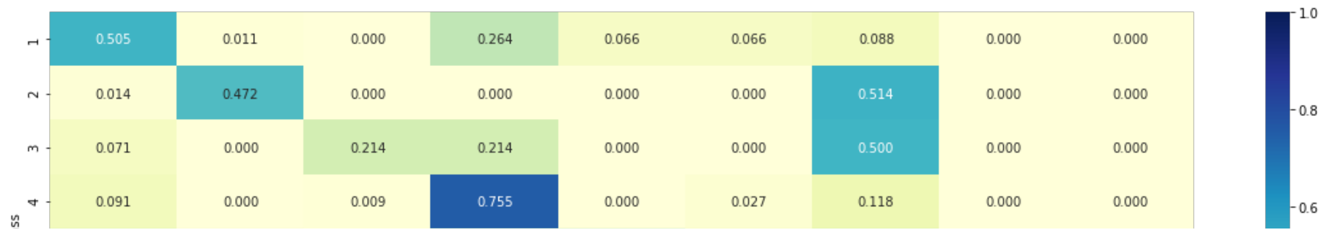
```
Log loss : 1.1700854093920483
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```

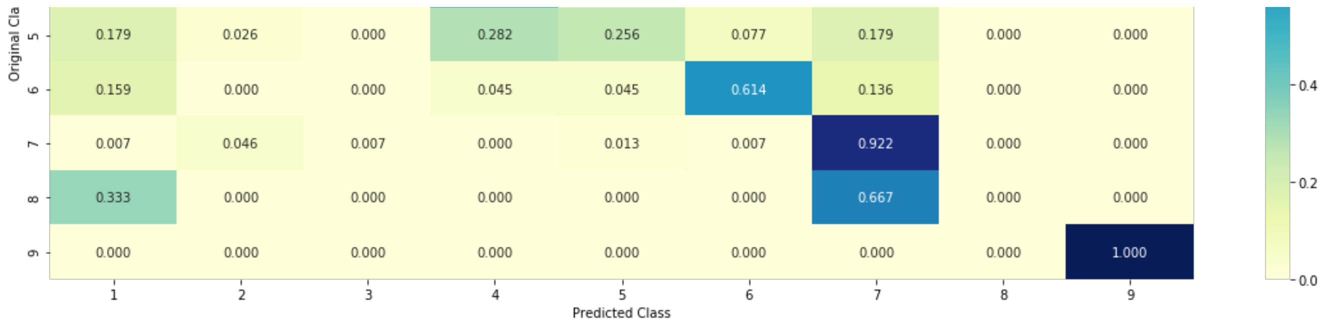


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





In [147]:

```
#feature ijmportance
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_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3227 0.2001 0.0049 0.1341 0.0384 0.0127 0.2788 0.0067 0.0016]]
Actual Class : 4
--------------------------------------------------
261 Text feature [ptprm] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

In [148]:

```
test_point_index = 300
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_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

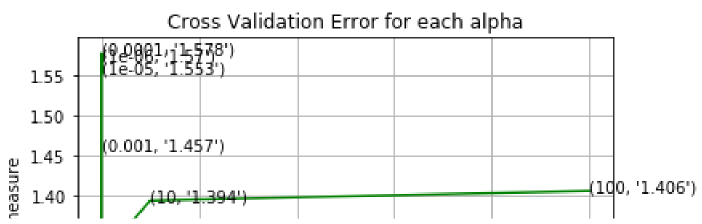
```
Predicted Class : 7
Predicted Class Probabilities: [[2.380e-02 1.460e-02 9.000e-04 8.400e-03 6.100e-03 2.200e-03 9.425
e-01
  1.300e-03 3.000e-04]]
Actual Class : 7
--------------------------------------------------
128 Text feature [hras] present in test data point [True]
266 Text feature [constitutive] present in test data point [True]
289 Text feature [ligand] present in test data point [True]
297 Text feature [phospho] present in test data point [True]
306 Text feature [transforming] present in test data point [True]
335 Text feature [cylinders] present in test data point [True]
340 Text feature [extracellular] present in test data point [True]
344 Text feature [reportedly] present in test data point [True]
351 Text feature [mitogen] present in test data point [True]
355 Text feature [downstream] present in test data point [True]
372 Text feature [activated] present in test data point [True]
380 Text feature [expressing] present in test data point [True]
419 Text feature [rbd] present in test data point [True]
493 Text feature [oncogenes] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

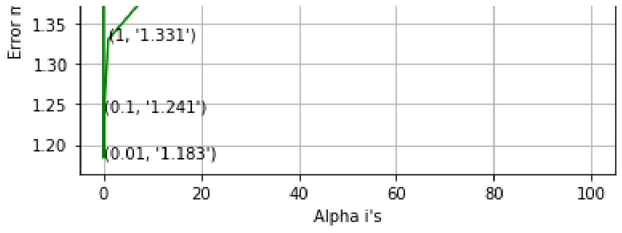
5.2 Losistic Regression with Bigrams

In [149]:

```
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_bi, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_bi, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_bi)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding_bi, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_bi, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_bi)
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_bi)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_bi)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.5701790906006337
for alpha = 1e-05
Log Loss : 1.55278370236505
for alpha = 0.0001
Log Loss : 1.5782313720610683
for alpha = 0.001
Log Loss : 1.4572061778889762
for alpha = 0.01
Log Loss : 1.1832101751298263
for alpha = 0.1
Log Loss : 1.241364147850324
for alpha = 1
Log Loss : 1.3311595400196026
for alpha = 10
Log Loss : 1.3943883275958244
for alpha = 100
Log Loss : 1.4064142874412064
```



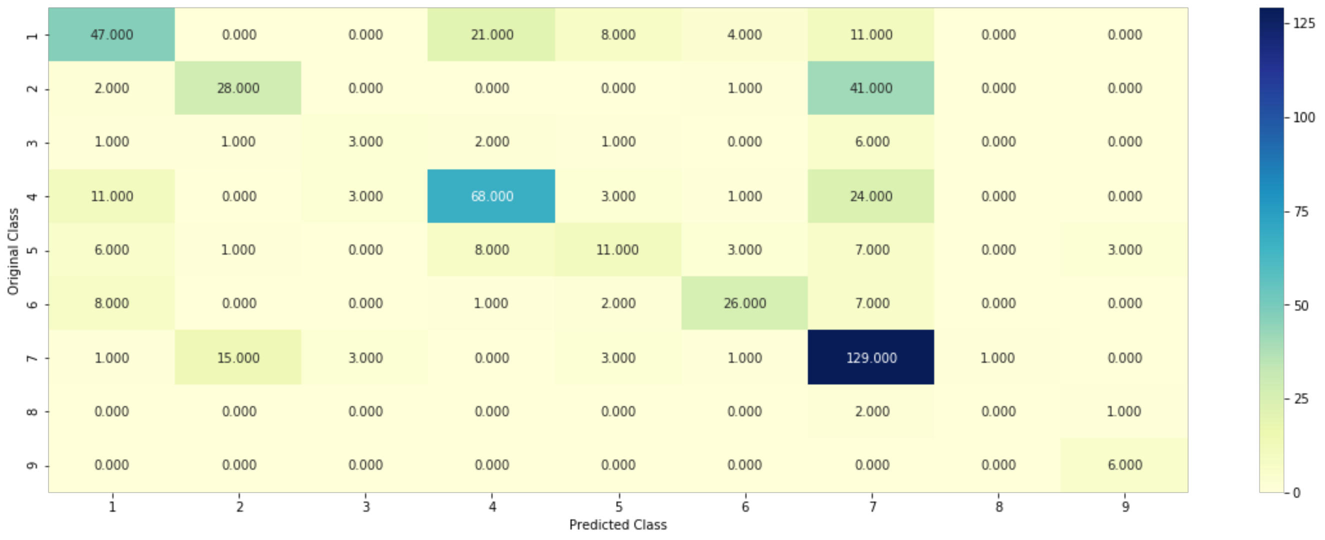


```
For values of best alpha =  0.01 The train log loss is: 0.8134810846808337
For values of best alpha =  0.01 The cross validation log loss is: 1.1832101751298263
For values of best alpha =  0.01 The test log loss is: 1.1962126707439618
```

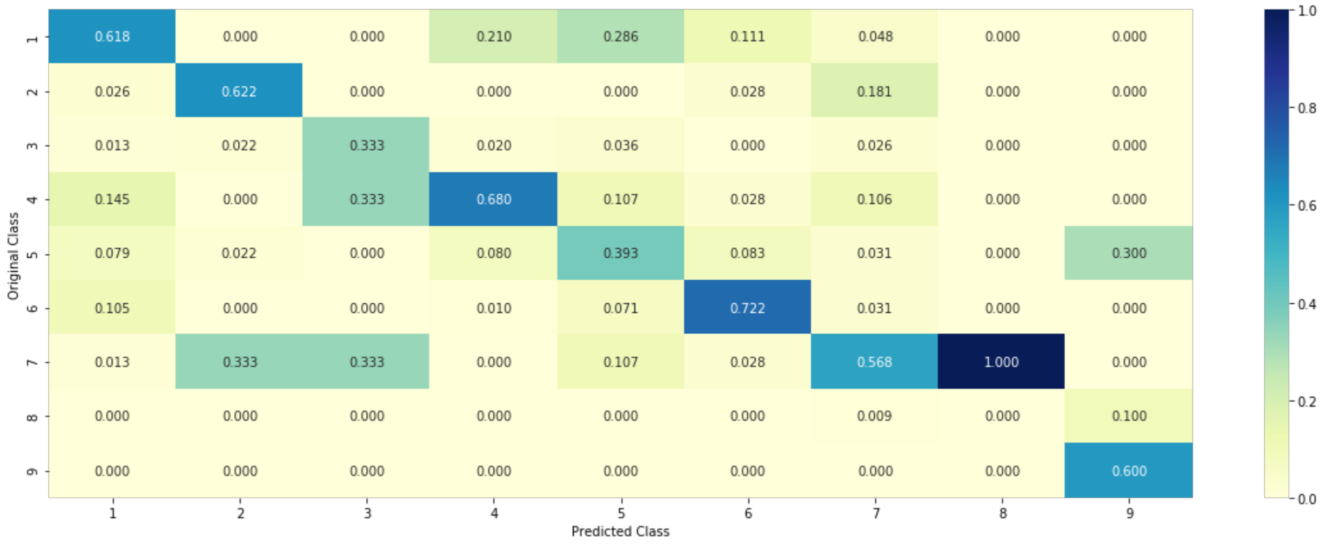
In [150]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_bi, train_y, cv_x_onehotCoding_bi, cv_y, cl
f)
```

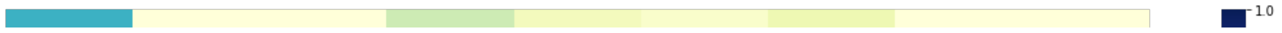
```
Log loss : 1.1832101751298263
Number of mis-classified points : 0.40225563909774437
-------------------- Confusion matrix --------------------
```

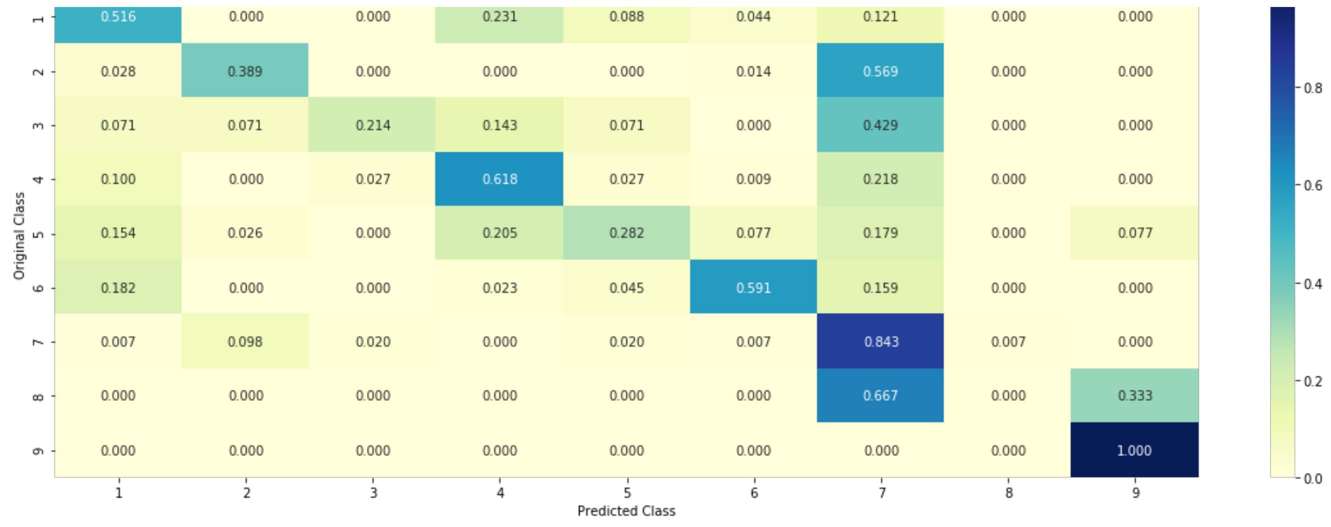


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





In [151]:

```
# 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(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer(ngram_range=(1,2))
    var_count_vec = CountVectorizer(ngram_range=(1,2))
    text_count_vec = CountVectorizer(ngram_range=(1,2),min_df=4)

    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 [{}]".format(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_r
o))
        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 [{}]".format(word,yes_no))

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

In [152]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding_bi,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_bi[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding_bi[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(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.2567 0.1845 0.0205 0.1506 0.0591 0.0301 0.2775 0.0081 0.0128]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

In [153]:

```
test_point_index = 200
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_bi[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding_bi[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(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3894 0.1341 0.0179 0.1314 0.0496 0.0244 0.2397 0.0058 0.0076]]
Actual Class : 1
--------------------------------------------------
45 Text feature [frame] present in test data point [True]
125 Text feature [methylcellulose] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

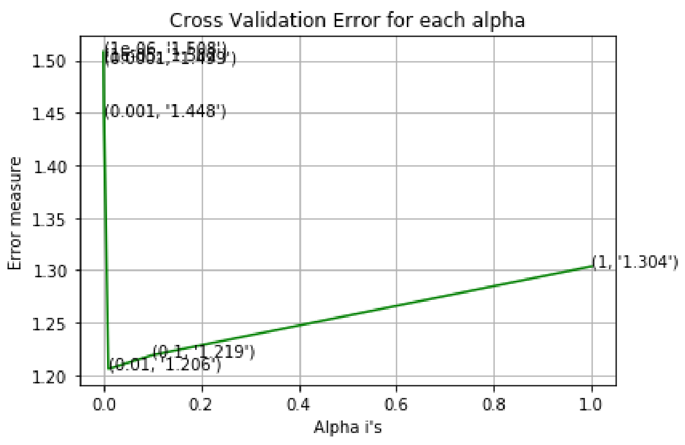
5.2.2 Bi GRams without balancong

In [154]:

```
# 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_i
ter=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-in
tuition-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:
#-------------------------------------
```

```python
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_bi, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_bi, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_bi)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding_bi)
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_bi)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_bi)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.5081324006888144
for alpha = 1e-05
Log Loss : 1.5021120138246367
for alpha = 0.0001
Log Loss : 1.4986833848181305
for alpha = 0.001
Log Loss : 1.4481572723090619
for alpha = 0.01
Log Loss : 1.2059063972077098
for alpha = 0.1
Log Loss : 1.2190406554643751
for alpha = 1
Log Loss : 1.3035185635118407
```

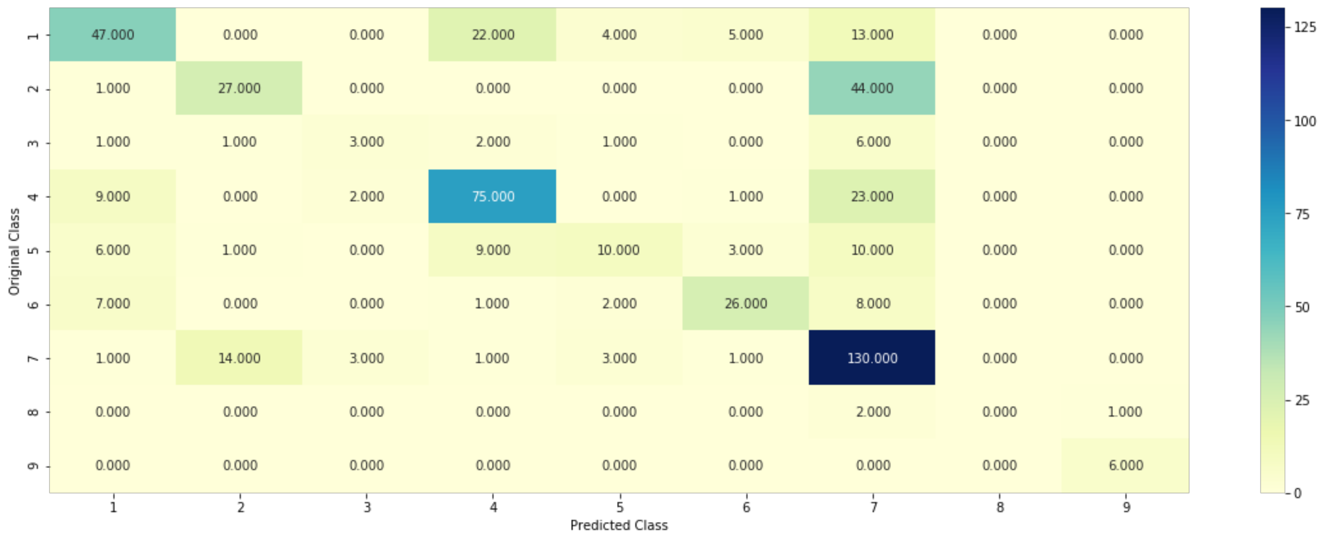


```
For values of best alpha =  0.01 The train log loss is: 0.8184187332427846
For values of best alpha =  0.01 The cross validation log loss is: 1.2059063972077098
For values of best alpha =  0.01 The test log loss is: 1.2129253577032963
For values of best alpha = 0.01 The best log loss is: [illegible]
```

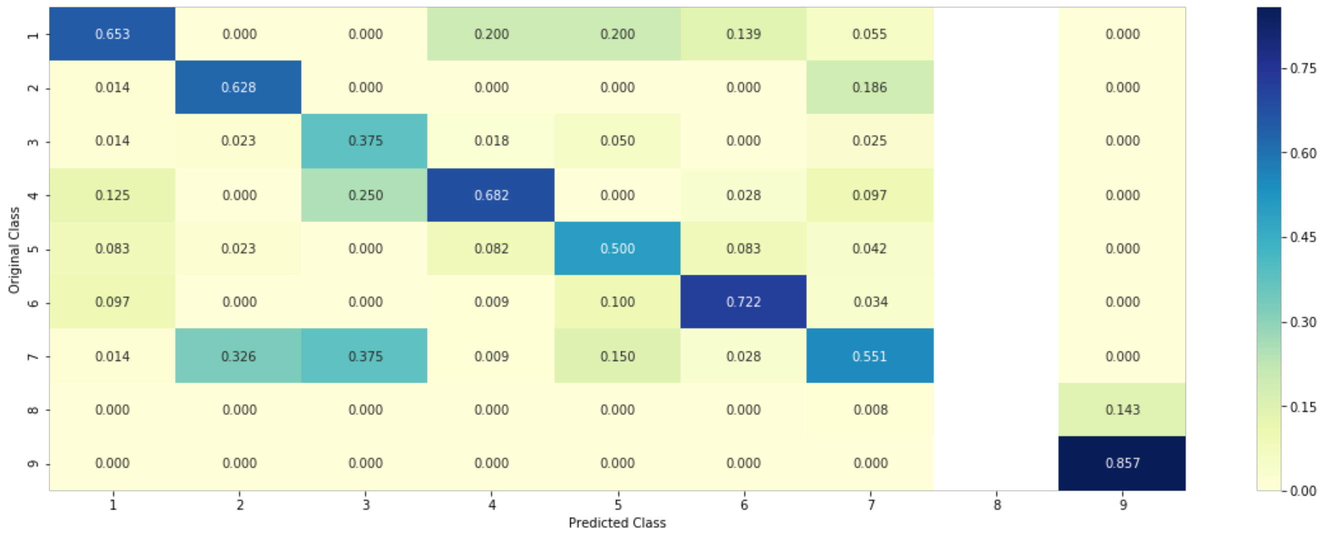
In [155]:

```python
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_bi, train_y, cv_x_onehotCoding_bi, cv_y, cl
f)
```

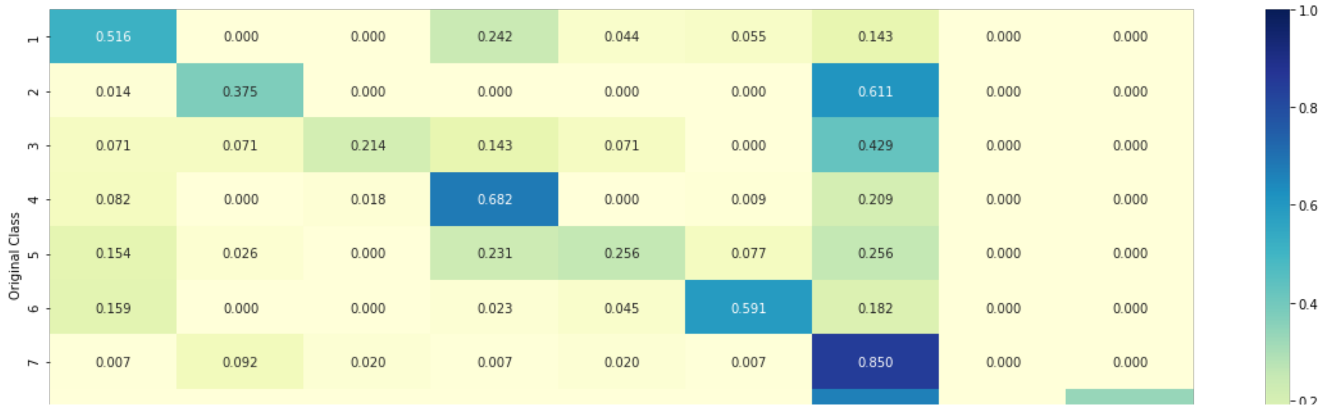
```
Log loss : 1.2059063972077098
Number of mis-classified points : 0.39097744360902253
-------------------- Confusion matrix --------------------
```

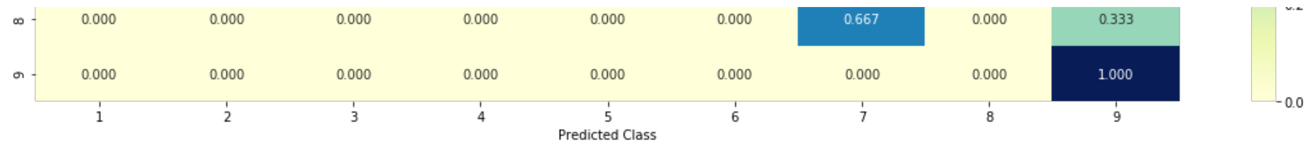


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





In [156]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_bi,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_bi[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding_bi[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(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.2582 0.1906 0.0104 0.1566 0.0587 0.0277 0.2864 0.0079 0.0034]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

In [ ]:

```
test_point_index = 200
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_bi[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding_bi[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(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3822 0.1393 0.0104 0.135  0.0497 0.0251 0.2482 0.0066 0.0037]]
Actual Class : 1
--------------------------------------------------
```

Lets do some feature enginerring and try to reduce test log loss to less than 1

Key Idea: AS many text features which are null earlier has been replaced by gene and varation features. Lets try to build text vocabulary using gene and varaition features.

In [12]:

```
# Collecting all the genes and variations data into a single list
gene_variation = []
for gene in data['Gene'].values:
     gene_variation.append(gene)
for variation in data['Variation'].values:
     gene_variation.append(variation)
```

In [19]:

```
tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()
train_text = tfidfVectorizer.transform(train_df['TEXT'])
test_text = tfidfVectorizer.transform(test_df['TEXT'])
cv_text = tfidfVectorizer.transform(cv_df['TEXT'])
cv_text = tfidfVectorizer.transform(cv_df['TEXT'])
```

In [27]:

```
train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_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_onehotCoding)).tocsr()
train_y = np.array(list(train_df['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_onehotCoding)).tocsr()
test_y = np.array(list(test_df['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(cv_df['Class']))
```

In [28]:

```
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, 129309)
(number of data points * number of features) in test data =  (665, 129309)
(number of data points * number of features) in cross validation data = (532, 129309)
```

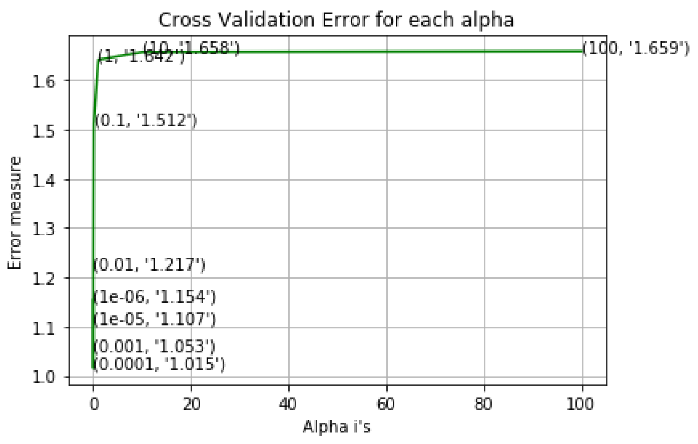
In [ ]:

```
Applying Logitic regression without balancing
```

In [30]:

```
alpha = [10 ** x for x in range(-6, 3)]
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))
 # 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( alpha=alpha[best_alpha], 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)
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, pr
edict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1539711248814608
for alpha = 1e-05
Log Loss : 1.1066221010215331
for alpha = 0.0001
Log Loss : 1.0150392996751356
for alpha = 0.001
Log Loss : 1.0525592428910804
for alpha = 0.01
Log Loss : 1.2166293387220264
for alpha = 0.1
Log Loss : 1.5121795499605706
for alpha = 1
Log Loss : 1.6419981837928346
for alpha = 10
Log Loss : 1.6576593693590316
for alpha = 100
Log Loss : 1.6593894346843812
```



```
For values of best alpha =  0.0001 The train log loss is: 0.43251572480677586
For values of best alpha =  0.0001 The cross validation log loss is: 1.0150392996751356
For values of best alpha =  0.0001 The test log loss is: 0.9918582603813854
```

we are getting test Log loss less than 1

In [ ]:

```
Performance Table:
```

In [4]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Hyperparameter", 'Vectorizer','Train logloss','Test logloss']
x.add_row(["Naive Bayees", 'alpha=0.1','tfidf','0.778','1.209'])
x.add_row(["K NN", 'k=21','Response coding','0.768','1.073'])
x.add_row(["LogisiticRegression(Balanced)", 'alpha=0.0001','tfidf','0.585','1.011'])
x.add_row(["LogisiticRegression( without Balance)", 'alpha=0.0001','tfidf','0.5695','1.045'])
x.add_row(['LinearSVM', 'alpha=0.0001','tfidf','0.67','1.11'])
x.add_row(['RandomForestclasifier', 'estimators=2000,depth=5','tfidf','0.854','1.222'])
x.add_row(['RandomForestclasifier', 'estimators=2000,depth=5','Responsecoding','0.55','1.31'])
x.add_row(['Stackingclassifer(LR+SVM+NB)', 'alpha=0.1','tfidf','1.12','1.15'])
x.add_row(['MaximumVotingclassifer(LR,SVC,RF)', '','tfidf','0.93','1.19'])
x.add_row(["LogisiticRegression(Balanced)", 'alpha=0.01','UniGrams','0.596','1.14'])
x.add_row(["LogisiticRegression(withoutBalance)", 'alpha=0.001','UniGrams','0.588','1.115'])
x.add_row(["LogisiticRegression(Balanced)", 'alpha=0.01','BiGrams','0.813','1.196'])
x.add_row(["LogisiticRegression(without Balance)", 'alpha=0.001','BiGrams','0.818','1.212'])
x.add_row(["LogisiticRegression(without Balance)",
```

```python
'alpha=0.0001','FeatureEngineering','0.43','0.99'])
print(x)
```

```
+----------------------------------------+-------------------------+---------------------+-----------
+---------------+
|                 Model                  |      Hyperparameter     |      Vectorizer     | Train logl
s | Test logloss |
+----------------------------------------+-------------------------+---------------------+-----------
+---------------+
|              Naive Bayees              |        alpha=0.1        |        tfidf        |     0.778
|     1.209     |
|                  K NN                  |           k=21          |   Response coding   |     0.768
|     1.073     |
|      LogisiticRegression(Balanced)     |       alpha=0.0001      |        tfidf        |     0.585
|     1.011     |
| LogisiticRegression( without Balance)  |       alpha=0.0001      |        tfidf        |     0.5695
|     1.045     |
|               LinearSVM                |       alpha=0.0001      |        tfidf        |      0.67
|      1.11     |
|          RandomForestclasifier         | estimators=2000,depth=5 |        tfidf        |     0.854
|     1.222     |
|          RandomForestclasifier         | estimators=2000,depth=5 |    Responsecoding   |      0.55
|      1.31     |
|       Stackingclassifer(LR+SVM+NB)     |        alpha=0.1        |        tfidf        |      1.12
|      1.15     |
|    MaximumVotingclassifer(LR,SVC,RF)   |                         |        tfidf        |      0.93
|      1.19     |
|      LogisiticRegression(Balanced)     |        alpha=0.01       |       UniGrams      |     0.596
|      1.14     |
|   LogisiticRegression(withoutBalance)  |       alpha=0.001       |       UniGrams      |     0.588
|     1.115     |
|      LogisiticRegression(Balanced)     |        alpha=0.01       |       BiGrams       |     0.813
|     1.196     |
|   LogisiticRegression(without Balance) |       alpha=0.001       |       BiGrams       |     0.818
|     1.212     |
|   LogisiticRegression(without Balance) |       alpha=0.0001      | FeatureEngineering  |      0.43
|      0.99     |
+----------------------------------------+-------------------------+---------------------+-----------
+---------------+
```