Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [2]:
```
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
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```
In [3]: data = pd.read_csv('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[3]:

	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_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 [4]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID",
"TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[4]:

	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 [5]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [6]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    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, "sec
onds")
```

```
Time took for preprocessing the text : 127.83454 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

In [9]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Varia
tion']
```

In [10]:
```
result[result['ID']==1109]
```

Out[10]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	null

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 [11]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=
y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same dis
tribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y
_train, test_size=0.2)
```

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

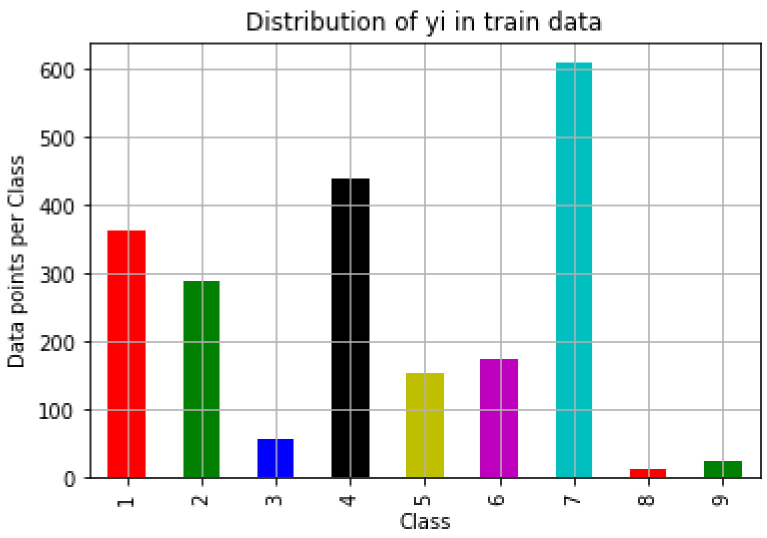
In [12]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

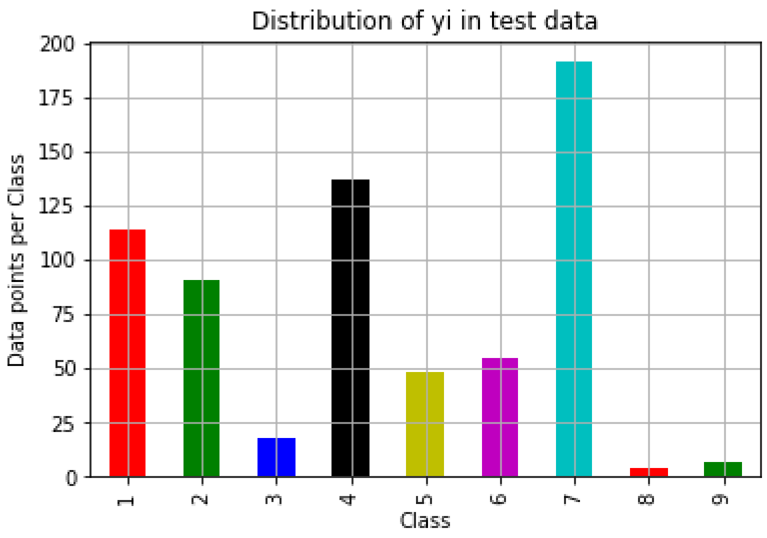
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [13]:
```
# it returns a dict, keys as class labels and values as the number of data poi
nts 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',color = my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.round((train_class_distribution.values[i]/train_df.shape[0]
*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.v
alues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*10
0), 3), '%)')

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

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

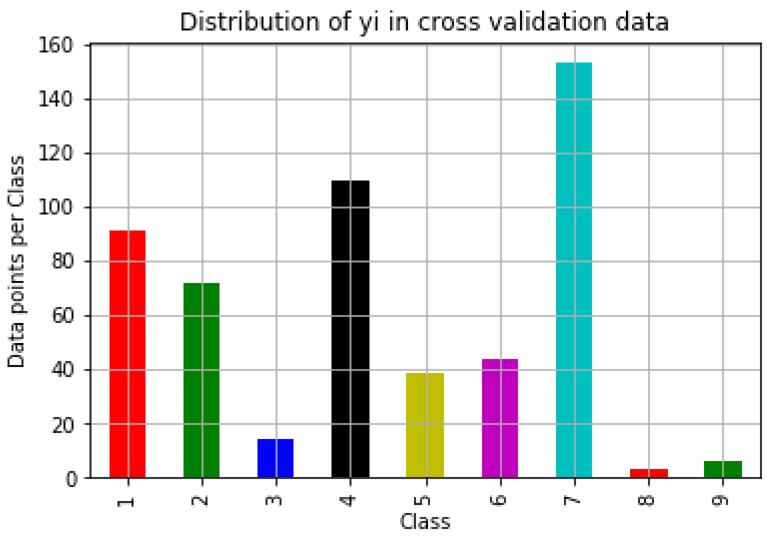
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```



```
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 %)
```

Observation

- here we can see that appox 28% point blong to class 7 and 20% point blong to class 4 these are major class
- and half % blong to class 3 and 1% point blong to class 9 these are minor class

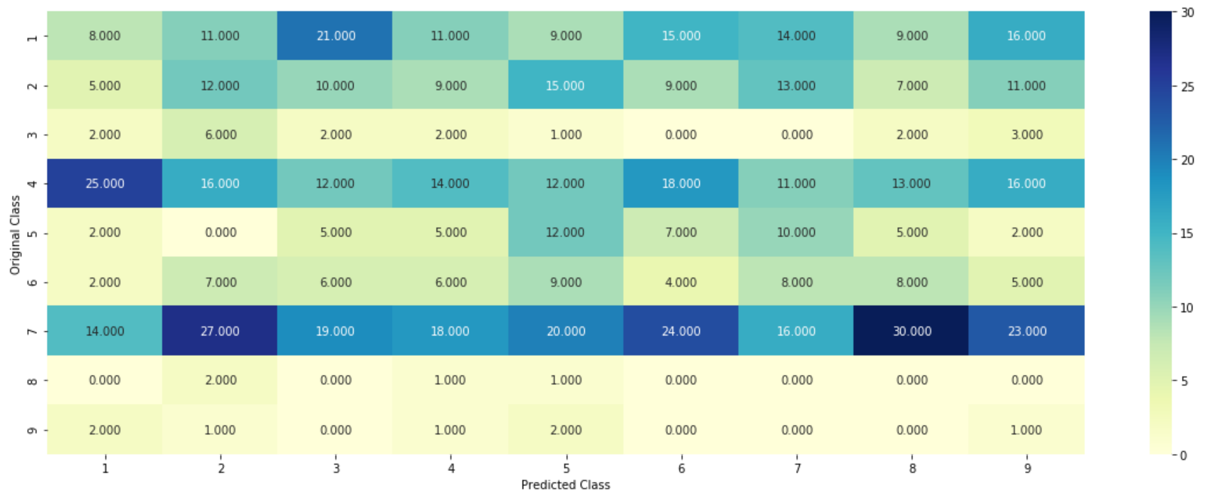
3.2 Prediction using a 'Random' Model

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

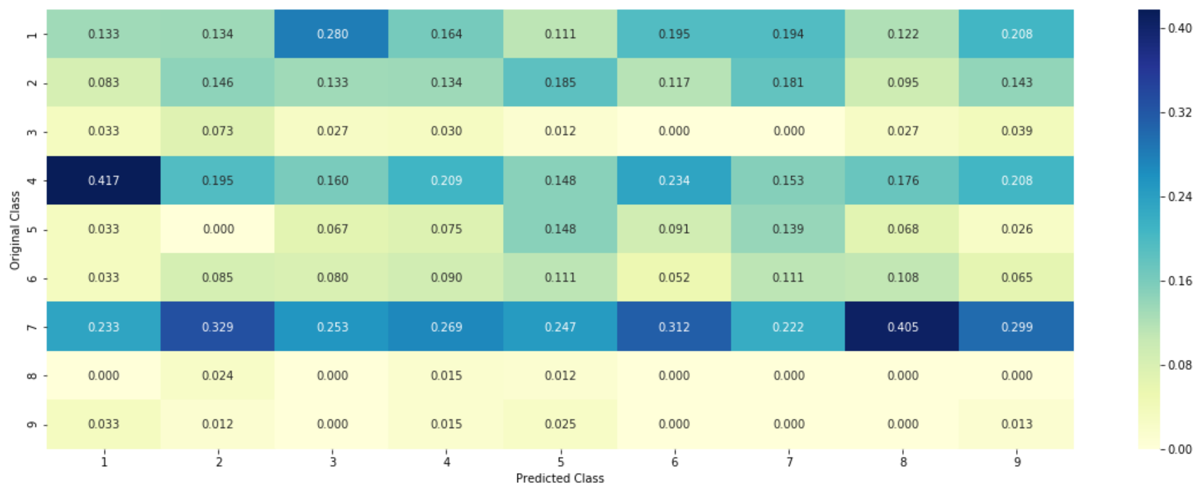
In [14]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at 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, y
ticklabels=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, y
ticklabels=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, y
ticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [15]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
r sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

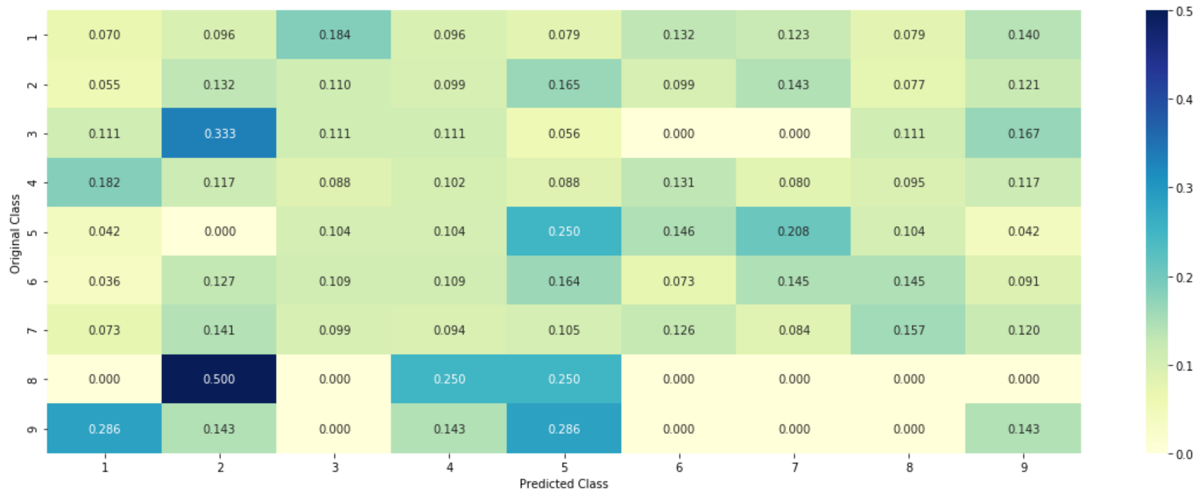
```
Log loss on Cross Validation Data using Random Model 2.46883811134
Log loss on Test Data using Random Model 2.53944478428
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [16]:
```
# 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 cl
ass1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representat
ion 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 occu
```

```
red 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 occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha
))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818
18177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.
03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897959
18366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.0510
20408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818
1818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.05
6818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060
6060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069
182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920
5295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.2715
2317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333
33334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.299
99999999999999, 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 fea
ture value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is
 there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

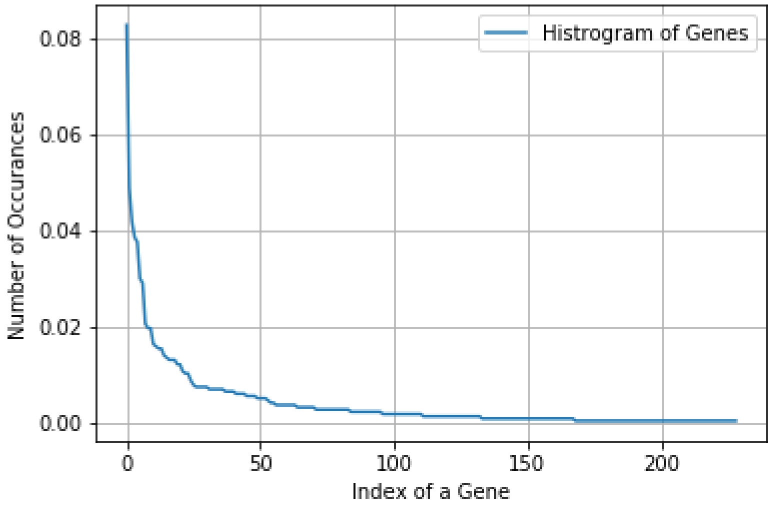
In [27]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 229
BRCA1     176
TP53      104
EGFR       89
PTEN       82
BRCA2      80
BRAF       64
KIT        62
ALK        44
ERBB2      42
PDGFRA     42
Name: Gene, dtype: int64
```

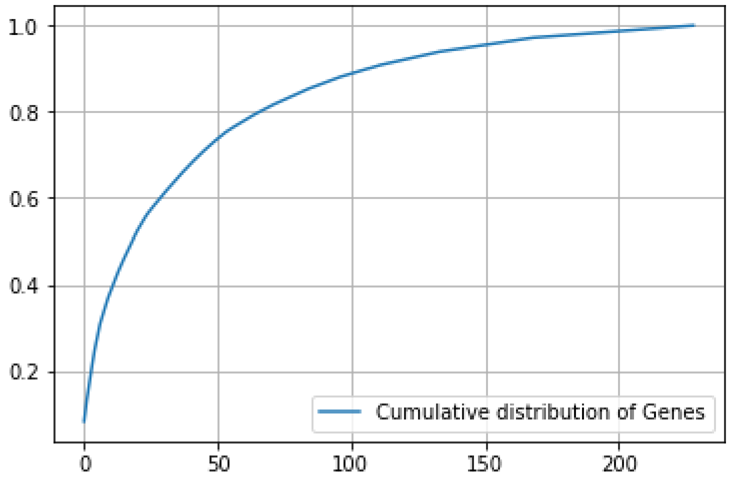
In [28]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

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

In [29]:
```
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 [30]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [31]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", tra
in_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test
_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df
))
```

In [32]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
```

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

In [33]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [34]:
```
train_df['Gene'].head()
```

Out[34]:
```
1393    FGFR3
2994      KIT
507      TP53
558     SMAD3
1696     PMS2
Name: Gene, dtype: object
```

```
In [35]: gene_vectorizer.get_feature_names()
```

```
Out[35]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1a',
          'arid1b',
          'arid2',
          'asxl2',
          'atm',
          'aurka',
          'aurkb',
          'axin1',
          'b2m',
          'bap1',
          'bcl10',
          'bcl2',
          'bcl2l11',
          'bcor',
          'braf',
          'brca1',
          'brca2',
          'brip1',
          'btk',
          'card11',
          'carm1',
          'casp8',
          'cbl',
          'ccnd1',
          'ccnd3',
          'ccne1',
          'cdh1',
          'cdk12',
          'cdk4',
          'cdk6',
          'cdk8',
          'cdkn1a',
          'cdkn1b',
          'cdkn2a',
          'cdkn2b',
          'cdkn2c',
          'cebpa',
          'chek2',
          'cic',
          'crebbp',
          'ctcf',
          'ctnnb1',
          'ddr2',
          'dicer1',
          'dnmt3a',
          'egfr',
          'eif1ax',
          'elf3',
          'ep300',
          'epas1',
          'erbb2',
          'erbb3',
          'erbb4',
          'ercc2',
          'ercc3',
          'ercc4',
          'erg',
          'errfi1',
          'esr1',
          'etv1',
          'etv6',
          'ewsr1',
          'ezh2',
          'fam58a',
          'fanca',
          'fancc',
          'fat1',
          'fbxw7',
          'fgf19',
          'fgf4',
          'fgfr1',
          'fgfr2',
          'fgfr3',
          'fgfr4',
          'flt3',
          'foxa1',
          'foxl2',
          'foxo1',
          'foxp1',
          'fubp1',
          'gata3',
          'gli1',
          'gnaq',
          'gnas',
          'h3f3a',
          'hist1h1c',
          'hla',
          'hnf1a',
          'hras',
          'idh1',
          'idh2',
          'igf1r',
          'ikzf1',
          'il7r',
          'jak1',
          'jak2',
          'kdm5a',
          'kdm5c',
          'kdm6a',
          'kdr',
          'keap1',
          'kit',
          'kmt2a',
          'kmt2b',
          'kmt2c',
          'kmt2d',
          'knstrn',
          'kras',
          'lats1',
          'lats2',
          'map2k1',
          'map2k2',
          'map2k4',
          'map3k1',
          'mapk1',
          'mdm2',
          'med12',
          'mef2b',
          'men1',
          'met',
          'mga',
          'mlh1',
          'mpl',
          'msh2',
          'msh6',
          'mtor',
          'myc',
          'mycn',
          'myd88',
          'myod1',
          'nf1',
          'nf2',
          'nfe2l2',
          'nfkbia',
          'nkx2',
          'notch1',
          'notch2',
          'npm1',
          'nras',
          'nsd1',
          'ntrk1',
          'ntrk2',
          'ntrk3',
          'nup93',
          'pak1',
          'pax8',
          'pbrm1',
          'pdgfra',
          'pdgfrb',
          'pik3ca',
          'pik3cb',
          'pik3cd',
          'pik3r1',
          'pik3r2',
          'pik3r3',
          'pim1',
          'pms2',
          'pole',
          'ppm1d',
          'ppp2r1a',
          'ppp6c',
          'prdm1',
          'ptch1',
          'pten',
          'ptpn11',
          'ptprd',
          'ptprt',
          'rab35',
          'rac1',
          'rad50',
          'rad51c',
          'rad54l',
          'raf1',
          'rara',
          'rasa1',
          'rb1',
          'rbm10',
          'ret',
          'rheb',
          'rhoa',
          'rit1',
          'rnf43',
          'ros1',
          'runx1',
          'rxra',
          'sdhb',
          'sf3b1',
          'shoc2',
          'shq1',
          'smad2',
          'smad3',
          'smad4',
          'smarca4',
          'smarcb1',
          'smo',
          'sos1',
          'sox9',
          'spop',
          'src',
          'stag2',
          'stat3',
          'stk11',
          'tcf3',
          'tcf7l2',
          'tert',
          'tet1',
          'tet2',
          'tgfbr1',
          'tmprss2',
          'tp53',
          'tp53bp1',
          'tsc1',
          'tsc2',
          'u2af1',
          'vhl',
          'whsc1l1',
          'xpo1',
          'xrcc2']
```

In [36]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature:", train_gene_feature_onehotCoding.shap
e)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding m
ethod. The shape of gene feature: (2124, 228)
```

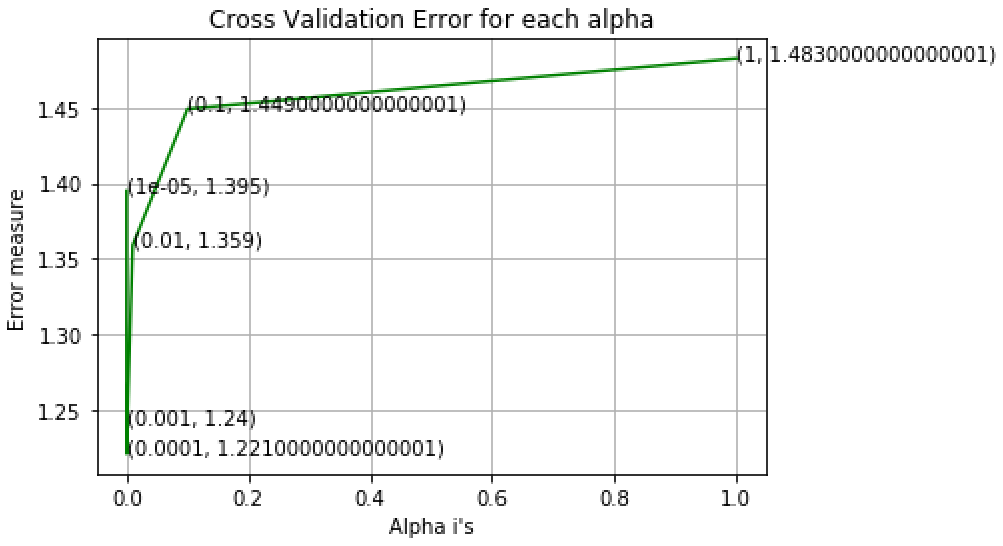
Q4. How good is this gene feature in predicting y_i?

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 [37]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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



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

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


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.39487935693
For values of alpha =  0.0001 The log loss is: 1.22059281005
For values of alpha =  0.001 The log loss is: 1.24048252045
For values of alpha =  0.01 The log loss is: 1.35876658836
For values of alpha =  0.1 The log loss is: 1.44920868283
For values of alpha =  1 The log loss is: 1.48263430934
```



```
For values of best alpha =  0.0001 The train log loss is: 1.02638066963
For values of best alpha =  0.0001 The cross validation log loss is: 1.220592
81005
For values of best alpha =  0.0001 The test log loss is: 1.23980664632
```

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 [38]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",
unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  229  gen
es in train dataset?
Ans
1. In test data 640 out of 665 : 96.2406015037594
2. In cross validation data 509 out of  532 : 95.67669172932331
```

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?

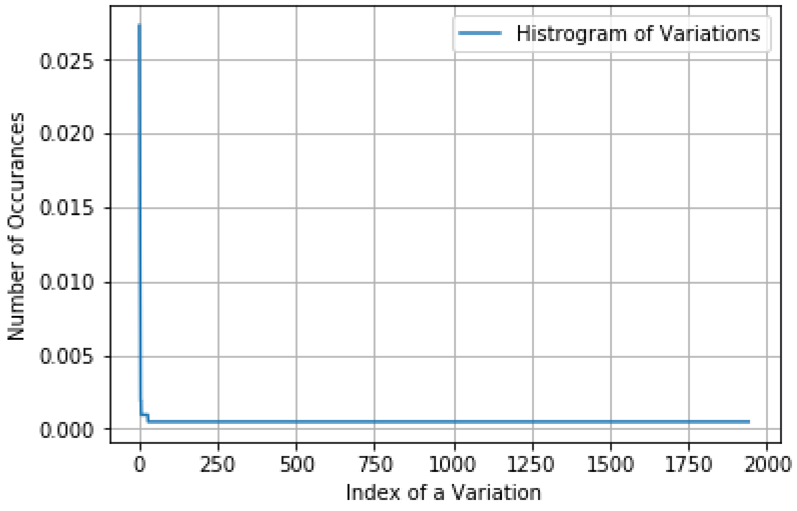
In [39]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1943
Truncating_Mutations    58
Deletion                48
Amplification           32
Fusions                 19
G12V                     4
Overexpression           4
G12C                     2
K117N                    2
T167A                    2
E542K                    2
Name: Variation, dtype: int64
```

In [40]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

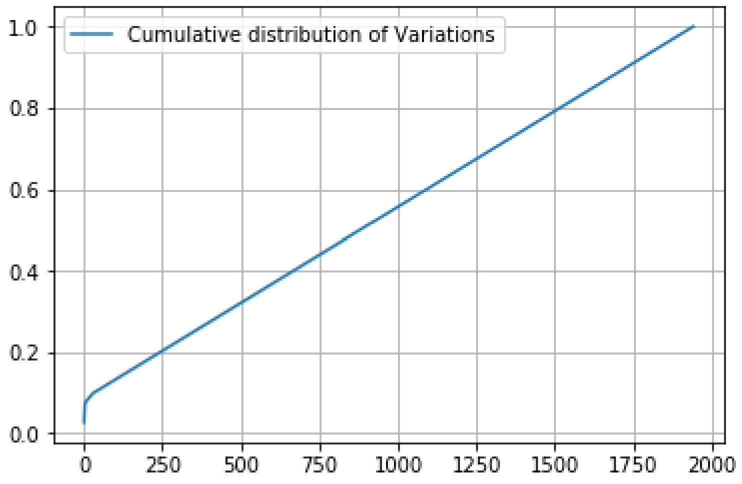
```
Ans: There are 1943 different categories of variations in the train data, and
they are distibuted as follows
```

In [41]:
```
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 [42]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[ 0.02730697  0.04990584  0.06497175 ...,  0.99905838  0.99952919  1.
]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [43]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Varia
tion", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variat
ion", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
n", cv_df))
```

In [44]:
```
print("train_variation_feature_responseCoding is a converted feature using the
 response coding method. The shape of Variation feature:", train_variation_fea
ture_responseCoding.shape)
```

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

In [45]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(trai
n_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df[
'Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Vari
ation'])
```

In [46]:
```
print("train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature:", train_variation_feat
ure_onehotCoding.shape)
```

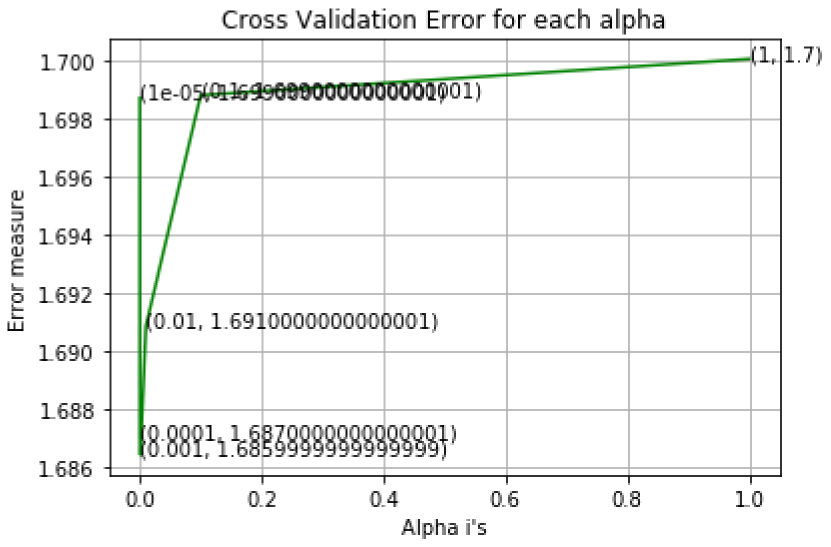
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1972)
```

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

Let's build a model just like the earlier!

In [47]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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



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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.69874012834
For values of alpha =  0.0001 The log loss is: 1.68702899702
For values of alpha =  0.001 The log loss is: 1.68643344068
For values of alpha =  0.01 The log loss is: 1.69088252194
For values of alpha =  0.1 The log loss is: 1.69883405125
For values of alpha =  1 The log loss is: 1.70008440187
```



```
For values of best alpha =  0.001 The train log loss is: 0.991658491091
For values of best alpha =  0.001 The cross validation log loss is: 1.6864334
4068
For values of best alpha =  0.001 The test log loss is: 1.70470118213
```

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 [48]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'
]))))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1943  genes in test and cross
validation data sets?
Ans
1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 70 out of  532 : 13.157894736842104
```

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 [49]:
```
# 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 [50]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_di
ct.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(
row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [73]:
```
# building a  TfidfVectorizer with all the words that occured minimum 3 times
 in train data
text_vectorizer =  TfidfVectorizer(max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 2000
```

In [74]:
```
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 [75]:
```
#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 [76]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train
_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [77]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=0)

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

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

In [78]:
```
#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 [79]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({212.50616612369888: 1, 143.07482141455321: 1, 127.69923788723162: 1,
109.94665188166135: 1, 102.48144687369434: 1, 99.3080705906183: 1, 98.6340096
88095887: 1, 96.802501161225109: 1, 96.706728618854797: 1, 92.16536421841317
4: 1, 87.70999511594917: 1, 77.020531144017994: 1, 74.491928374626156: 1, 73.
971585662295738: 1, 72.827026764338669: 1, 71.971963620595048: 1, 66.76933055
6640867: 1, 66.412786175199855: 1, 66.270935809735334: 1, 65.087842171399501:
1, 63.516561981774366: 1, 62.605368471718052: 1, 58.268155121673544: 1, 57.03
1860181432293: 1, 56.454517584589183: 1, 56.359513285836471: 1, 55.4962243823
50547: 1, 54.247725071977278: 1, 53.891142267868453: 1, 53.773811932426682:
1, 53.455536111378422: 1, 53.04022316289322: 1, 52.907620265449211: 1, 49.987
901842445197: 1, 49.074526703987232: 1, 48.555998831368882: 1, 46.97456904153
8352: 1, 46.758820492188832: 1, 45.904985990532182: 1, 45.449720027910608: 1,
45.135004851693722: 1, 44.052660282723693: 1, 43.430170288095503: 1, 43.38661
2087812161: 1, 42.949601586806807: 1, 42.705679659864387: 1, 40.9664486397843
89: 1, 39.1836768910918: 1, 39.106356821183205: 1, 38.617557944069219: 1, 38.
097258383864691: 1, 38.045865002922476: 1, 36.849284379146049: 1, 36.70935178
4133638: 1, 36.666349380086594: 1, 36.436396996686341: 1, 36.051268803295081:
1, 36.041670166710887: 1, 35.630053686103132: 1, 35.629503610559844: 1, 35.44
4330675165801: 1, 35.318138830548527: 1, 34.646134968189429: 1, 34.5656506302
03152: 1, 34.558541208504181: 1, 34.146169079689741: 1, 34.117547272418548:
1, 34.057631822789389: 1, 34.014748050789926: 1, 33.477853542444912: 1, 33.44
9639189603261: 1, 32.911421464613454: 1, 32.813788075502309: 1, 32.6003207031
183: 1, 32.26247637346674: 1, 31.33740294950157: 1, 31.153147503345963: 1, 3
1.137834048560542: 1, 30.658999629966427: 1, 30.253120639646841: 1, 30.133197
534534567: 1, 29.912628253354672: 1, 29.850338370531819: 1, 29.7499314165003
6: 1, 29.644573433217801: 1, 29.323732958408694: 1, 29.315809234205474: 1, 2
8.922452024437892: 1, 28.835091654395075: 1, 28.602341478104226: 1, 28.327593
261607031: 1, 27.854209012158822: 1, 27.671613539085079: 1, 27.63772848782025
1: 1, 27.505795871769553: 1, 27.480500363034096: 1, 27.421443306358775: 1, 2
7.249355359443932: 1, 27.0302569519305: 1, 27.010294696253727: 1, 26.99768632
6985139: 1, 26.987019756089321: 1, 26.737487976938819: 1, 26.549345527251802:
1, 26.473407460698937: 1, 26.450463634453662: 1, 26.342641557496076: 1, 26.27
879953488938: 1, 26.238171009984374: 1, 26.190603038840951: 1, 26.13166370412
2101: 1, 26.006484222005547: 1, 25.636855549703238: 1, 25.556531001477666: 1,
25.549886066990005: 1, 25.54800665035744: 1, 25.517842449006821: 1, 25.416505
402692305: 1, 25.188911116905984: 1, 25.123719517472868: 1, 25.09619062450586
6: 1, 25.079133090045541: 1, 25.037750079369438: 1, 25.012328565300436: 1, 2
4.812793831841958: 1, 24.657462212179244: 1, 24.449978969284722: 1, 24.338060
928857878: 1, 24.337012503608982: 1, 24.220269189390692: 1, 24.04169048693921
7: 1, 24.036672913039567: 1, 24.001031398668005: 1, 23.946161998111275: 1, 2
3.859241112363144: 1, 23.848358585005052: 1, 23.404832354585718: 1, 23.338336
21192419: 1, 23.2933113586617: 1, 23.155479486511734: 1, 23.147696944875225:
1, 23.142001301612918: 1, 22.99179334859857: 1, 22.533257759450077: 1, 22.451
084472951315: 1, 22.361364670438338: 1, 22.16788742909684: 1, 22.145560797255
079: 1, 21.909896433205546: 1, 21.832375391152922: 1, 21.687944542966402: 1,
21.641643660566054: 1, 21.626712226114641: 1, 21.425216578812325: 1, 21.35937
8006166477: 1, 21.330449615275349: 1, 21.276899963850184: 1, 21.2067842203253
25: 1, 21.168354905088634: 1, 21.117515281320461: 1, 21.010380982640189: 1, 2
0.808980078931334: 1, 20.732126643517855: 1, 20.723703870577214: 1, 20.664927
159238836: 1, 20.607387263288146: 1, 20.594305771884489: 1, 20.58998459157837
9: 1, 20.539842914940319: 1, 20.45554326148023: 1, 20.443942351318558: 1, 20.
39002313459925: 1, 20.380986238864622: 1, 20.308740511086349: 1, 20.296268441
097581: 1, 20.294180930059536: 1, 20.021437004598798: 1, 20.003450693880925:
1, 19.933643320364602: 1, 19.757868700244622: 1, 19.662945625493904: 1, 19.64
3256444297187: 1, 19.478683847069725: 1, 19.392205384781665: 1, 19.3799381014
32227: 1, 19.346947382662627: 1, 19.305791874677276: 1, 19.287885827923965:
1, 19.216318901435439: 1, 19.129056475732341: 1, 19.112609601963069: 1, 18.98
```

```
8461304967167: 1, 18.987570436707813: 1, 18.981677095576853: 1, 18.9117986999
44338: 1, 18.906809270147555: 1, 18.893784866346166: 1, 18.874666271075839:
1, 18.819535104409951: 1, 18.723894591517009: 1, 18.678116217583884: 1, 18.65
8697032125488: 1, 18.615617472718977: 1, 18.605682278758064: 1, 18.6050444821
96085: 1, 18.578716996287902: 1, 18.529775896964175: 1, 18.520477266489934:
1, 18.400391869897341: 1, 18.376833714840039: 1, 18.359535688460955: 1, 18.34
2026812281784: 1, 18.318193862287334: 1, 18.290716002517243: 1, 18.2721518193
97131: 1, 18.235409636525461: 1, 18.22988754343195: 1, 18.217393839859003: 1,
18.195403512653755: 1, 18.13297882264942: 1, 18.087436797712886: 1, 18.052050
82128699: 1, 17.986713281659405: 1, 17.946129366999234: 1, 17.94222491730479
2: 1, 17.913979318098615: 1, 17.910351002252423: 1, 17.904507672554296: 1, 1
7.880162964871989: 1, 17.803887453675543: 1, 17.780828824569557: 1, 17.770357
905023285: 1, 17.768690600352443: 1, 17.718852515062959: 1, 17.71049632963016
8: 1, 17.681645633358599: 1, 17.637664775030686: 1, 17.60357067759621: 1, 17.
588351269646807: 1, 17.577310450325403: 1, 17.557588686214928: 1, 17.41553872
9310697: 1, 17.381522185269681: 1, 17.316403851074085: 1, 17.301568010569319:
1, 17.267853538350561: 1, 17.262083621761597: 1, 17.247419908620145: 1, 17.21
5253608856365: 1, 17.192740014437547: 1, 17.182985012084313: 1, 17.1339417901
08132: 1, 17.099698982206483: 1, 17.016456162501044: 1, 16.96607033849093: 1,
16.874858373824292: 1, 16.731895372015035: 1, 16.711745489046457: 1, 16.70052
9613220684: 1, 16.699166524942978: 1, 16.505488745141289: 1, 16.4723497993477
04: 1, 16.470456409614769: 1, 16.44903152149033: 1, 16.437431666065731: 1, 1
6.411106704786057: 1, 16.384118775466227: 1, 16.323803418176567: 1, 16.323612
716162: 1, 16.314711108692418: 1, 16.274780899879325: 1, 16.25581926116595:
1, 16.242096073095304: 1, 16.233127068322318: 1, 16.207990247267858: 1, 16.14
2782794245552: 1, 16.120120453399782: 1, 16.114381005367662: 1, 16.0767727043
5822: 1, 16.065013360830424: 1, 16.025953442781798: 1, 16.018993661985348: 1,
16.015922934393448: 1, 15.993554553293805: 1, 15.987542411782176: 1, 15.95051
4202163918: 1, 15.939820909931701: 1, 15.875074591563902: 1, 15.8734038885483
49: 1, 15.855857228924679: 1, 15.828511388147962: 1, 15.825856187145133: 1, 1
5.762651534781345: 1, 15.759417391392715: 1, 15.757440433831356: 1, 15.719188
841361564: 1, 15.620902474268066: 1, 15.56895736554184: 1, 15.45718176570296
1: 1, 15.445538857596702: 1, 15.369076031899329: 1, 15.362641939958417: 1, 1
5.331330599769291: 1, 15.315410250652162: 1, 15.274591954340274: 1, 15.183435
903829608: 1, 15.14634657095668: 1, 15.121711648014672: 1, 15.09507724489220
4: 1, 15.067339422178028: 1, 15.066384682202592: 1, 14.979823800862967: 1, 1
4.947745891116181: 1, 14.941033888865917: 1, 14.888755473768098: 1, 14.805239
476228721: 1, 14.755094569003861: 1, 14.747792327571316: 1, 14.71883781368177
5: 1, 14.665333820055226: 1, 14.594932285619505: 1, 14.585537820777599: 1, 1
4.584048052488102: 1, 14.561786163776501: 1, 14.537726664379559: 1, 14.528435
800967735: 1, 14.52417993616826: 1, 14.504671278851918: 1, 14.49699993253965
5: 1, 14.454986579407169: 1, 14.453068715893977: 1, 14.406601405218609: 1, 1
4.315281821289943: 1, 14.309738462490269: 1, 14.263079562193901: 1, 14.254639
90959037: 1, 14.172852772988442: 1, 14.157391507259847: 1, 14.14241998288921
6: 1, 14.138371848578133: 1, 14.095309351568227: 1, 14.083807350196269: 1, 1
4.07742913262296: 1, 14.056616248316713: 1, 14.017108310343005: 1, 14.0098174
18874066: 1, 13.988243902782356: 1, 13.919625012050659: 1, 13.90546500440268
8: 1, 13.889827441568722: 1, 13.884705438798786: 1, 13.796885411819718: 1, 1
3.761190673571026: 1, 13.71571230684499: 1, 13.704885941850153: 1, 13.6867753
45377772: 1, 13.626230087559307: 1, 13.590138633018949: 1, 13.58132623178142:
1, 13.580531876850197: 1, 13.572865159099585: 1, 13.569835911323585: 1, 13.56
5806364603116: 1, 13.564640915464347: 1, 13.550405912667085: 1, 13.5332083928
64278: 1, 13.529102438113064: 1, 13.526799722005771: 1, 13.497975248310317:
1, 13.485150132718571: 1, 13.44738538462685: 1, 13.39928174319995: 1, 13.3900
26689190625: 1, 13.386706320825425: 1, 13.366637115150841: 1, 13.306584393628
366: 1, 13.301854942526282: 1, 13.292079243064904: 1, 13.271844541529248: 1,
13.258084904585136: 1, 13.213394409295814: 1, 13.205023175769048: 1, 13.14384
```

```
2634456878: 1, 13.109450580417684: 1, 13.091862682928424: 1, 13.0904154078865
89: 1, 13.08879019847182: 1, 13.08183437667104: 1, 13.067993413479481: 1, 13.
044542649661958: 1, 13.03657567549139: 1, 13.007816589139786: 1, 13.002552829
878253: 1, 12.98910141823122: 1, 12.970008970204756: 1, 12.965800316215109:
1, 12.957889738001471: 1, 12.934937570493574: 1, 12.86216005255714: 1, 12.860
922351927423: 1, 12.835523439661145: 1, 12.833685418949914: 1, 12.83324426713
6667: 1, 12.825887588197741: 1, 12.823226826998919: 1, 12.811324340503553: 1,
12.808533676341391: 1, 12.800119888618832: 1, 12.78326256746735: 1, 12.776306
048293797: 1, 12.764044528970386: 1, 12.696060589048775: 1, 12.67319708934499
2: 1, 12.65394399585006: 1, 12.649095000981639: 1, 12.646327878816923: 1, 12.
643781312640183: 1, 12.634911904550703: 1, 12.598763554615235: 1, 12.58195844
7221361: 1, 12.545391631662921: 1, 12.539177268526247: 1, 12.537092160568625:
1, 12.500587879120435: 1, 12.488066301054337: 1, 12.485163747610201: 1, 12.46
9874329962447: 1, 12.466101290606877: 1, 12.465145010965212: 1, 12.4481570359
60232: 1, 12.368865040968338: 1, 12.360417478020386: 1, 12.346499766660097:
1, 12.341788107621309: 1, 12.340425715338842: 1, 12.319450826191032: 1, 12.29
4926120040085: 1, 12.285055464011611: 1, 12.251419546395111: 1, 12.2463807737
6275: 1, 12.227628153088457: 1, 12.199050616736553: 1, 12.196426620726145: 1,
12.188582420582369: 1, 12.181184264707911: 1, 12.176415822610275: 1, 12.17193
826944956: 1, 12.137402112090779: 1, 12.095691328466151: 1, 12.08175403746892
4: 1, 12.08096811190803: 1, 12.036630924219432: 1, 12.030199267970639: 1, 12.
028946658277524: 1, 12.026910473294551: 1, 12.020192814137076: 1, 12.01881163
7430604: 1, 11.981861026228193: 1, 11.937665480492399: 1, 11.906897015824303:
1, 11.889778880726098: 1, 11.872742825365673: 1, 11.837714129212817: 1, 11.83
5688203148154: 1, 11.831222677222042: 1, 11.824748278691114: 1, 11.8207802442
57241: 1, 11.818771674425685: 1, 11.81477842829435: 1, 11.788833815839551: 1,
11.782350235218258: 1, 11.780727521384414: 1, 11.778363415608345: 1, 11.70419
0687727094: 1, 11.699889559678946: 1, 11.693409333682247: 1, 11.6556864615813
06: 1, 11.627344279479461: 1, 11.627112942511756: 1, 11.615079709288095: 1, 1
1.588438979680046: 1, 11.582744367670726: 1, 11.571408483416267: 1, 11.570701
192064378: 1, 11.565947920241008: 1, 11.527867956248006: 1, 11.51751138340411
5: 1, 11.507669478609429: 1, 11.492668586484127: 1, 11.477828678533568: 1, 1
1.470322099723427: 1, 11.451574643326637: 1, 11.443693571888529: 1, 11.440153
11641285: 1, 11.38481605417986: 1, 11.373786504950424: 1, 11.356398820823269:
1, 11.317224472167577: 1, 11.314912333016617: 1, 11.276745464932867: 1, 11.27
305553666343: 1, 11.199212366004755: 1, 11.194650253220987: 1, 11.17437616132
4641: 1, 11.153244316002649: 1, 11.150799153303867: 1, 11.14947524593444: 1,
11.040093384942482: 1, 11.039297023538468: 1, 11.038410542728617: 1, 11.03696
5316819092: 1, 11.03479118122139: 1, 11.010856689956723: 1, 10.99593373206365
3: 1, 10.98962351682847: 1, 10.984872384476688: 1, 10.965624673202838: 1, 10.
938105262262189: 1, 10.93289202008634: 1, 10.919702838181406: 1, 10.918544981
54394: 1, 10.917858234913878: 1, 10.917845908245036: 1, 10.91291279097657: 1,
10.879390483692704: 1, 10.879178461590689: 1, 10.868157307308941: 1, 10.86300
3832908724: 1, 10.806031527135438: 1, 10.775529199646217: 1, 10.7636700157148
16: 1, 10.736099728634525: 1, 10.721053729478369: 1, 10.716287591539665: 1, 1
0.676886414519041: 1, 10.67511958676617: 1, 10.661747503793507: 1, 10.6485925
66709992: 1, 10.615390202225178: 1, 10.612552726809596: 1, 10.60567052162076
1: 1, 10.600694722581528: 1, 10.589024469311719: 1, 10.586267073543247: 1, 1
0.581377381864945: 1, 10.575663323787472: 1, 10.560122379625554: 1, 10.549240
701504743: 1, 10.539103844482765: 1, 10.502089803650517: 1, 10.44369722827368
1: 1, 10.439032441945875: 1, 10.423162617855315: 1, 10.40268252699228: 1, 1
0.387245016054646: 1, 10.375153785911225: 1, 10.359892960188546: 1, 10.358198
403416077: 1, 10.353606208822839: 1, 10.335261746461846: 1, 10.33215947956293
6: 1, 10.325220297805913: 1, 10.316523767685622: 1, 10.31634016106592: 1, 10.
302739817365921: 1, 10.26601075069147: 1, 10.265318518112556: 1, 10.265145594
808487: 1, 10.263783090337407: 1, 10.260172876829715: 1, 10.252179469092948:
1, 10.247045331260701: 1, 10.235950809943326: 1, 10.234661665948579: 1, 10.20
```

```
5992552957031: 1, 10.203718832224761: 1, 10.200562160463582: 1, 10.1914361797
36851: 1, 10.183004308519433: 1, 10.178230088815427: 1, 10.122042169994542:
1, 10.077109936490976: 1, 10.072402809333271: 1, 10.06864135060421: 1, 10.032
565342198918: 1, 10.010265881732352: 1, 9.9623895572125374: 1, 9.954801008324
651: 1, 9.9346751145652696: 1, 9.9307326140438743: 1, 9.9268036575740712: 1,
9.9144812783045086: 1, 9.9023640196013574: 1, 9.8879574069412133: 1, 9.872967
8514581387: 1, 9.8577762238987354: 1, 9.8561686742191963: 1, 9.82835853444278
84: 1, 9.815603537842696: 1, 9.7887213931358801: 1, 9.773233924867176: 1, 9.7
730848375253636: 1, 9.7676461814853042: 1, 9.7343278424031396: 1, 9.716786056
9593056: 1, 9.7164801674076955: 1, 9.7108288947827504: 1, 9.7002451997069503:
1, 9.6983898474714021: 1, 9.6951866702566178: 1, 9.6897526202179272: 1, 9.685
8644798085329: 1, 9.6751667867469475: 1, 9.6744394317632807: 1, 9.63895722169
67476: 1, 9.5991442812641612: 1, 9.59631712805629: 1, 9.5915317613659195: 1,
9.5882127263532215: 1, 9.571880548483815: 1, 9.5606604190797295: 1, 9.5537063
516877723: 1, 9.547350498877698: 1, 9.5455274824186631: 1, 9.545079434314306
6: 1, 9.538160274424067: 1, 9.5326112501030309: 1, 9.5232272192133465: 1, 9.5
189782289526796: 1, 9.515061069809569: 1, 9.5149685068114191: 1, 9.5103195871
170758: 1, 9.4762984429919754: 1, 9.4713582242551855: 1, 9.4651512844352332:
1, 9.4618856648938401: 1, 9.4616932060599463: 1, 9.45285406112556: 1, 9.44913
24922107864: 1, 9.4450669636040079: 1, 9.4435290521266246: 1, 9.4246440674014
931: 1, 9.4215178211517649: 1, 9.4125790487325727: 1, 9.403307274969178: 1,
9.3974967717265319: 1, 9.3973377279907098: 1, 9.3699559754386925: 1, 9.352964
8812796413: 1, 9.3513528921696505: 1, 9.3488186064194139: 1, 9.34491888771378
5: 1, 9.3430165378950019: 1, 9.3315992468807245: 1, 9.3239266533727108: 1, 9.
3234615530229394: 1, 9.3182837990686345: 1, 9.3120994116835796: 1, 9.30565570
34690872: 1, 9.2896612306319319: 1, 9.2854433574585347: 1, 9.281880857763599
8: 1, 9.2751305932633716: 1, 9.2661369967518077: 1, 9.2651541940323057: 1, 9.
2473926381361764: 1, 9.2406505167604021: 1, 9.236161967130176: 1, 9.228498377
2179636: 1, 9.224029604962233: 1, 9.2122060565572248: 1, 9.1928978828387287:
1, 9.1868395406157077: 1, 9.1789578384967285: 1, 9.1760772119478116: 1, 9.175
2255375450549: 1, 9.1743842708251595: 1, 9.1701998491617278: 1, 9.15607796716
96118: 1, 9.1232331816021635: 1, 9.1221003045873843: 1, 9.1192619626238898:
1, 9.1182796161571655: 1, 9.1174997805554465: 1, 9.1028512795762513: 1, 9.093
4288934914029: 1, 9.0829506167892546: 1, 9.0723019642641489: 1, 9.06588898075
73519: 1, 9.0622562473132717: 1, 9.0621645477959394: 1, 9.0513769643182087:
1, 9.0411078539416856: 1, 9.0241997005318897: 1, 9.0240682708462199: 1, 9.014
978978429129: 1, 8.9835774134143218: 1, 8.9677525039721075: 1, 8.949759416728
4298: 1, 8.9375277105635043: 1, 8.9209307130863458: 1, 8.9202489487341303: 1,
8.920216179588186: 1, 8.8964910961931079: 1, 8.8911114069076014: 1, 8.8828490
872677275: 1, 8.8822326680953925: 1, 8.878786703310146: 1, 8.878382533031235
4: 1, 8.8656947354726938: 1, 8.8656194403340312: 1, 8.8468885188521096: 1, 8.
8270371000325181: 1, 8.8266575139316288: 1, 8.819781309219298: 1, 8.819342223
8171912: 1, 8.8081874351471043: 1, 8.8077998464057572: 1, 8.8024909863094791:
1, 8.7999834264268859: 1, 8.7802527564200794: 1, 8.7799054988861052: 1, 8.773
9271091081026: 1, 8.7645518917962608: 1, 8.7604029789658515: 1, 8.75352477636
95345: 1, 8.7513040313710491: 1, 8.74347098720067: 1, 8.736846556949164: 1,
8.718429043834373: 1, 8.7183204478364136: 1, 8.7137184158453511: 1, 8.6927882
836858039: 1, 8.68713756130132: 1, 8.684550216501556: 1, 8.6843306776652778:
1, 8.6753315097869059: 1, 8.6641382930815762: 1, 8.6414669833273621: 1, 8.639
1748029311639: 1, 8.6294650470577352: 1, 8.6276595195090398: 1, 8.62679902971
40607: 1, 8.6266243144800629: 1, 8.6245564411590188: 1, 8.622562878550724: 1,
8.6139323723687404: 1, 8.6093198505250434: 1, 8.6016709408516192: 1, 8.593160
0608834877: 1, 8.5673669881152144: 1, 8.5577167245753802: 1, 8.55770257306852
86: 1, 8.5566362576958355: 1, 8.5530139305034556: 1, 8.541149734598001: 1, 8.
5282842515985031: 1, 8.5187565836897612: 1, 8.4977158168086397: 1, 8.49333958
09406598: 1, 8.4860952449373972: 1, 8.4762461420110657: 1, 8.475983649958672
9: 1, 8.4719694283950293: 1, 8.4643350094831078: 1, 8.4619970515511636: 1, 8.
```

```
4619595531112388: 1, 8.4591813111996643: 1, 8.4544980298456238: 1, 8.45435869
91711965: 1, 8.4541932598658018: 1, 8.4538295796963858: 1, 8.452942236914648
1: 1, 8.4239246352254469: 1, 8.4127137488559072: 1, 8.408767147508529: 1, 8.4
074787045912576: 1, 8.407333031165015: 1, 8.397560803070359: 1, 8.39500296518
327: 1, 8.3930836218090192: 1, 8.3925525301041652: 1, 8.3768883721081586: 1,
8.3762340776834261: 1, 8.3707262593436553: 1, 8.3679132254019937: 1, 8.367232
1347911218: 1, 8.359636225552924: 1, 8.3526193453512096: 1, 8.344722332694269
4: 1, 8.3300071115310548: 1, 8.3298245393580377: 1, 8.3229944213327016: 1, 8.
3161850041534553: 1, 8.2949382214809031: 1, 8.2944901998205669: 1, 8.29025492
34747287: 1, 8.2856504974414378: 1, 8.2730602642407227: 1, 8.266675069658578
7: 1, 8.2540073832786174: 1, 8.2427827604751247: 1, 8.2342473709019526: 1, 8.
217018960375496: 1, 8.1950075750665459: 1, 8.1888056821660857: 1, 8.185242958
5633697: 1, 8.1814773689371645: 1, 8.1809086585853859: 1, 8.1779464969274525:
1, 8.1775729801776063: 1, 8.1601800817416645: 1, 8.1581070485354203: 1, 8.155
1686918285231: 1, 8.1470183351600074: 1, 8.1351363122087896: 1, 8.13189651526
18023: 1, 8.1301558230517941: 1, 8.1274690966357372: 1, 8.1183350911532219:
1, 8.1149993267402571: 1, 8.107612714745601: 1, 8.1021532404856504: 1, 8.0858
992916219901: 1, 8.0838016476729706: 1, 8.0810378640259319: 1, 8.077467697115
8001: 1, 8.0452139688518542: 1, 8.0429640235536048: 1, 8.0408544657189616: 1,
8.0368631573839586: 1, 8.0217117362421035: 1, 8.0199402107340134: 1, 8.013265
1735730391: 1, 8.0108039447913484: 1, 8.0055038247143351: 1, 8.00364019283000
34: 1, 7.9947980414989104: 1, 7.9892318502678146: 1, 7.9889045978399427: 1,
7.9803345801344792: 1, 7.9707091832343595: 1, 7.9699933612037821: 1, 7.961007
6811130508: 1, 7.9576119525789855: 1, 7.939015113623217: 1, 7.922850718602081
3: 1, 7.9019855733693696: 1, 7.8987980467033578: 1, 7.8987700722208327: 1, 7.
8981448878266383: 1, 7.8930223301537321: 1, 7.8785883057266268: 1, 7.87629127
02625183: 1, 7.8692104229363178: 1, 7.8627252641149035: 1, 7.862386438053519
6: 1, 7.8621083250160675: 1, 7.8489345393015268: 1, 7.8400259895099724: 1, 7.
833461746691107: 1, 7.8333383652804764: 1, 7.832384594525843: 1, 7.8311974605
309222: 1, 7.8311516378629484: 1, 7.8262662627706625: 1, 7.822307795703332:
1, 7.8153877303023496: 1, 7.8021317280460458: 1, 7.7859855712360053: 1, 7.774
5129682487297: 1, 7.7740612420980941: 1, 7.7725600602899716: 1, 7.76952171455
11597: 1, 7.7552637798649418: 1, 7.7550347464224503: 1, 7.7535349523861603:
1, 7.7512005577848626: 1, 7.7462635493020482: 1, 7.7400325599933435: 1, 7.733
3939703892138: 1, 7.7287901748555692: 1, 7.7252563655538742: 1, 7.68483356498
13331: 1, 7.6841133217218784: 1, 7.6744415697395256: 1, 7.6704620441740161:
1, 7.6656144938991835: 1, 7.6653592228172043: 1, 7.6474529802051299: 1, 7.646
372375189344: 1, 7.6452535460258: 1, 7.6431830640578431: 1, 7.637904278369580
5: 1, 7.6279088649750078: 1, 7.6162764918705133: 1, 7.6123059130362183: 1, 7.
6093515736020665: 1, 7.6022258461863181: 1, 7.5999191756660407: 1, 7.59285202
0043515: 1, 7.5813485844478148: 1, 7.5810329412064767: 1, 7.5672836150712159:
1, 7.5659338972620116: 1, 7.5539936922341768: 1, 7.5493372070419875: 1, 7.547
2036216422893: 1, 7.5290287735048942: 1, 7.5226178184598824: 1, 7.52098051422
13479: 1, 7.5208278126781565: 1, 7.5171622988275297: 1, 7.509223952285808: 1,
7.5071291931072315: 1, 7.5037774379236888: 1, 7.4956331566167496: 1, 7.491341
9697124919: 1, 7.4890509259875131: 1, 7.487632646883295: 1, 7.479651973873587
6: 1, 7.4751031865651552: 1, 7.471265222278638: 1, 7.4651866319550289: 1, 7.4
651626703924059: 1, 7.4643783288231802: 1, 7.4641824546036037: 1, 7.455968688
1063749: 1, 7.4487718830031282: 1, 7.4479937253599751: 1, 7.3998573133802772:
1, 7.3944879939344963: 1, 7.3926229483815131: 1, 7.3864013670835007: 1, 7.381
4407813786307: 1, 7.3780140228828106: 1, 7.3745346464912993: 1, 7.37327264464
54988: 1, 7.3643066555043921: 1, 7.3642561384298588: 1, 7.3635788274139378:
1, 7.3613532008804672: 1, 7.3531588482148598: 1, 7.3508518952306074: 1, 7.346
5580153344527: 1, 7.3450033875592444: 1, 7.3400402621983716: 1, 7.33681399591
1597: 1, 7.3338654701458736: 1, 7.3252542075680758: 1, 7.3230528732920401: 1,
7.3201677190628889: 1, 7.3134587339563657: 1, 7.3120252166947974: 1, 7.309765
1201776443: 1, 7.3085291801475947: 1, 7.3074145062865847: 1, 7.29357025696676
```

```
86: 1, 7.2912818451874362: 1, 7.2804814372013027: 1, 7.2786547016068361: 1,
7.2753932240390853: 1, 7.2688731534218096: 1, 7.2669629029757523: 1, 7.247135
0088214548: 1, 7.2438195286171521: 1, 7.2433344390610541: 1, 7.23439549962120
41: 1, 7.2242024664921622: 1, 7.2230900802191611: 1, 7.2153790673009217: 1,
7.2152966246695929: 1, 7.2089075512215537: 1, 7.2080389085669614: 1, 7.197460
5948018038: 1, 7.1960680206861003: 1, 7.1955880129061836: 1, 7.18204539678478
15: 1, 7.1802096575472794: 1, 7.1791571699664596: 1, 7.172189170067389: 1, 7.
1629993922485138: 1, 7.1470372468380088: 1, 7.1200840456408292: 1, 7.11993706
33517409: 1, 7.1065833825576217: 1, 7.1064054041898066: 1, 7.102532006668611
5: 1, 7.1023724639045387: 1, 7.090653233867088: 1, 7.078461704190671: 1, 7.07
70211176141169: 1, 7.0702968987909189: 1, 7.0701651902920588: 1, 7.0665688021
219015: 1, 7.0620780815372584: 1, 7.0338590790652935: 1, 7.0334694460694527:
1, 7.0299223368138843: 1, 7.0207508754702923: 1, 7.0169216390945461: 1, 7.016
5934205061404: 1, 7.0128248790089147: 1, 7.0064642211242987: 1, 7.00585353425
16031: 1, 7.0057148526547639: 1, 6.9973308188951755: 1, 6.9972015463028567:
1, 6.9960392721431823: 1, 6.9869693752591084: 1, 6.9852831964880249: 1, 6.983
2095691499756: 1, 6.9789011132789076: 1, 6.9779547426739779: 1, 6.97769203178
31523: 1, 6.9590025929823849: 1, 6.9576025213050086: 1, 6.957365884923739: 1,
6.9533673538470637: 1, 6.9521959957311639: 1, 6.9468761111992698: 1, 6.941728
5493862728: 1, 6.9398957852411058: 1, 6.9321953126970364: 1, 6.92227471746846
67: 1, 6.9154291874374607: 1, 6.9099438771027559: 1, 6.9069737090142516: 1,
6.9057468169620719: 1, 6.9033307565503348: 1, 6.8888811599707473: 1, 6.882277
0896701417: 1, 6.871314354289435: 1, 6.8678826495596184: 1, 6.867281089957285
4: 1, 6.8658030930596778: 1, 6.8650407722087259: 1, 6.8625874656695514: 1, 6.
8576717274520984: 1, 6.8482499603310156: 1, 6.8169742651822425: 1, 6.80092503
76529931: 1, 6.7687713498474649: 1, 6.7687431220370424: 1, 6.763478846024059
8: 1, 6.7605897897847891: 1, 6.7585040161682723: 1, 6.7573020051941066: 1, 6.
7502448401927904: 1, 6.7421977246840497: 1, 6.7361079294024933: 1, 6.73380949
00030772: 1, 6.7329184543912408: 1, 6.729274347112181: 1, 6.7201410411351352:
1, 6.7191490842327655: 1, 6.7179030921023504: 1, 6.7120763446223988: 1, 6.704
0602957130346: 1, 6.7011762123525713: 1, 6.6967380243707959: 1, 6.69489830450
00672: 1, 6.6940253511401915: 1, 6.6833858669240138: 1, 6.6820955285609411:
1, 6.6768167761088479: 1, 6.6752257627699656: 1, 6.6617220127123495: 1, 6.658
5220421215583: 1, 6.6581888646506453: 1, 6.653034804339006: 1, 6.652502230866
4647: 1, 6.6515188642919805: 1, 6.6397566650774769: 1, 6.6359309675231017: 1,
6.6223632677175326: 1, 6.6180448705095847: 1, 6.608156057416287: 1, 6.6079413
052548679: 1, 6.6039728332875969: 1, 6.6023096438676054: 1, 6.597236559865254
3: 1, 6.5968198901773949: 1, 6.5951815656396784: 1, 6.5888219653167868: 1, 6.
5871405644995296: 1, 6.5859848566823063: 1, 6.5841376029427696: 1, 6.58365733
17530789: 1, 6.5829595529603973: 1, 6.5798158555474355: 1, 6.569367774005509
2: 1, 6.5627619336968488: 1, 6.5618113859080873: 1, 6.5530847960116843: 1, 6.
5502113674347235: 1, 6.5461347320914189: 1, 6.5414678668400725: 1, 6.54091569
17588112: 1, 6.540260546841246: 1, 6.5350036528794453: 1, 6.5349153614684186:
1, 6.5340331764434643: 1, 6.5336761520065396: 1, 6.5329623153467296: 1, 6.529
3025654816281: 1, 6.5193650293428727: 1, 6.5135989425953031: 1, 6.49931194083
15891: 1, 6.4988442769590442: 1, 6.4943472044009134: 1, 6.4916889678949836:
1, 6.4743267557384883: 1, 6.4695082721312414: 1, 6.4621630519062867: 1, 6.460
7209921485831: 1, 6.4565055641741358: 1, 6.453885335359888: 1, 6.452919967622
0541: 1, 6.4488026557803657: 1, 6.4465357350049528: 1, 6.443488662443249: 1,
6.4423630415169058: 1, 6.4410877037596483: 1, 6.4354389275005079: 1, 6.434540
5711258286: 1, 6.4303708054470885: 1, 6.4268698988344219: 1, 6.42416142064162
35: 1, 6.4125347621574376: 1, 6.4123631412222526: 1, 6.4033248379042051: 1,
6.3993526838432206: 1, 6.3940024421736004: 1, 6.3899430839250044: 1, 6.376221
644750002: 1, 6.3645831248576963: 1, 6.361778981018209: 1, 6.353959516907112
4: 1, 6.3482797206598907: 1, 6.3413214339844304: 1, 6.3407207708226263: 1, 6.
33986439602569: 1, 6.3357812829057458: 1, 6.3320485235012285: 1, 6.3307983274
332225: 1, 6.3275146586522935: 1, 6.3274084071594903: 1, 6.3215932996922701:
```

```
1, 6.3210273646855066: 1, 6.3204331721217857: 1, 6.317203981163841: 1, 6.3045
119155792522: 1, 6.3042591148285574: 1, 6.3021564697502646: 1, 6.297577587913
1932: 1, 6.2939945960604931: 1, 6.2914543842233952: 1, 6.280714621058352: 1,
6.2746416216504413: 1, 6.268776880643661: 1, 6.2680716844247071: 1, 6.2657239
055130365: 1, 6.2643031790295645: 1, 6.2570333515453456: 1, 6.256703443356489
2: 1, 6.2507227897842732: 1, 6.2466058261964861: 1, 6.2385384687324805: 1, 6.
2291895032810007: 1, 6.214445032282355: 1, 6.2138934240034311: 1, 6.211610174
1086629: 1, 6.2094253212099693: 1, 6.2014672204452275: 1, 6.195596432673252:
1, 6.193135150164693: 1, 6.1858865949988031: 1, 6.1836966954222525: 1, 6.1793
646258852393: 1, 6.175978548243247: 1, 6.1746307529060518: 1, 6.1733744795914
545: 1, 6.1533036642877512: 1, 6.1500940158400228: 1, 6.1493676807802995: 1,
6.1329094988941719: 1, 6.1240666197336715: 1, 6.1165377987241492: 1, 6.110727
7116901244: 1, 6.1093889641097325: 1, 6.0996358843934164: 1, 6.08015260592649
65: 1, 6.0741231892157312: 1, 6.0666613301941661: 1, 6.0650173060516739: 1,
6.0581926138567397: 1, 6.0561320994594929: 1, 6.0471651799571493: 1, 6.045923
7604162341: 1, 6.0457740678457528: 1, 6.0455268594621892: 1, 6.04460681730087
76: 1, 6.0391398942042898: 1, 6.03399118848297: 1, 6.0319052634740125: 1, 6.0
275925938481247: 1, 6.0273388476423264: 1, 6.0210538057302072: 1, 6.014751797
4134317: 1, 6.0137712959069223: 1, 6.0087799945050957: 1, 6.0056842737422489:
1, 6.0007616824273438: 1, 5.9990894243978179: 1, 5.99790670466595: 1, 5.99504
75383908062: 1, 5.9943590340028949: 1, 5.992967423068384: 1, 5.98852620865070
23: 1, 5.9856036329738256: 1, 5.9825239528690757: 1, 5.9806625518549401: 1,
5.9782125062568348: 1, 5.9743494679279108: 1, 5.9743134695120252: 1, 5.974257
1670018911: 1, 5.9717455791709435: 1, 5.9671089530120724: 1, 5.96605962613946
69: 1, 5.9653989997788228: 1, 5.9624531914901979: 1, 5.9488405063090877: 1,
5.9420844856747372: 1, 5.9337987790113242: 1, 5.9307060577885178: 1, 5.927486
5570074384: 1, 5.9241612997771904: 1, 5.9201500473356932: 1, 5.92009752224695
07: 1, 5.9190828152778252: 1, 5.9182361548056681: 1, 5.9170297509503023: 1,
5.9144580418069577: 1, 5.910559404489609: 1, 5.9051548253883999: 1, 5.9002186
90504242: 1, 5.8873722434018738: 1, 5.871404024925857: 1, 5.8670768979669523:
1, 5.8654303562415979: 1, 5.8617513639175707: 1, 5.8475161674502401: 1, 5.842
6595680746187: 1, 5.8415991465129595: 1, 5.8346535024340209: 1, 5.83076857131
38442: 1, 5.8293405683279991: 1, 5.8211780477644597: 1, 5.8208130943119176:
1, 5.8154983891195711: 1, 5.8153733786237494: 1, 5.8087456262875339: 1, 5.807
5057897151394: 1, 5.7995040843942434: 1, 5.7974988265364651: 1, 5.79499411535
91906: 1, 5.793836075878283: 1, 5.7892952176308548: 1, 5.789003260757501: 1,
5.7841207187482579: 1, 5.7838936841321136: 1, 5.7838456266095077: 1, 5.781493
8379058978: 1, 5.7768375955151265: 1, 5.7719204394558012: 1, 5.76262315536522
07: 1, 5.7621660573911493: 1, 5.7531247554422968: 1, 5.7453866049428317: 1,
5.7433368265333273: 1, 5.7428307947415931: 1, 5.7417641953826619: 1, 5.738155
4761303812: 1, 5.7264140621281747: 1, 5.72558441042968: 1, 5.705626790298203
4: 1, 5.6998884408488166: 1, 5.6974992188226015: 1, 5.6947479913943848: 1, 5.
688145661063376: 1, 5.6865261224980408: 1, 5.6823213089454336: 1, 5.679980209
5970264: 1, 5.6676547737789091: 1, 5.6645453084835378: 1, 5.6579698095043574:
1, 5.652071280054753: 1, 5.6494833777189033: 1, 5.6486489852921684: 1, 5.6458
214917580412: 1, 5.6452298810538855: 1, 5.6449557023789909: 1, 5.636586528896
6175: 1, 5.6320593647724522: 1, 5.6311113948650569: 1, 5.6233261069726295: 1,
5.6030357305963543: 1, 5.602741710033019: 1, 5.6002297301936883: 1, 5.5993981
929027985: 1, 5.5990775044737253: 1, 5.5987858746986801: 1, 5.591005440765158
6: 1, 5.5863010780956248: 1, 5.5853201287418646: 1, 5.5848667003611894: 1, 5.
5848084189567277: 1, 5.5814006399475424: 1, 5.5721616811245349: 1, 5.57183477
92507474: 1, 5.564086238025908: 1, 5.563265851982413: 1, 5.5609322437517763:
1, 5.5580996753586032: 1, 5.5453907288713475: 1, 5.5425918263959071: 1, 5.539
932309964513: 1, 5.539830220222453: 1, 5.5345447296147636: 1, 5.5333047247559
204: 1, 5.5261284586580208: 1, 5.5195063032395799: 1, 5.5179790218652505: 1,
5.5095292869017092: 1, 5.497638410402276: 1, 5.4975322736449685: 1, 5.4971765
225995188: 1, 5.494757580435925: 1, 5.4930722280676454: 1, 5.487530723909476
```

```
8: 1, 5.4786764257342444: 1, 5.4743514026986242: 1, 5.4657713444025413: 1, 5.
4559761603332229: 1, 5.4459880739905158: 1, 5.4396610594152142: 1, 5.43134921
45015179: 1, 5.42959086230643: 1, 5.4289033374607873: 1, 5.4245433330170565:
1, 5.4233006088562297: 1, 5.4217306455934855: 1, 5.4195890878476609: 1, 5.417
4439077978738: 1, 5.416746912212882: 1, 5.4134442256298367: 1, 5.411810632662
3252: 1, 5.41173525481139: 1, 5.4050922323395998: 1, 5.4037013199301427: 1,
5.3997416876642514: 1, 5.3995436207800909: 1, 5.3989229421066272: 1, 5.394525
0061359378: 1, 5.3904651484102644: 1, 5.388222116906489: 1, 5.38352975004537
54: 1, 5.3813021157783973: 1, 5.3759504145779662: 1, 5.3709257328391509: 1,
5.3660324404189828: 1, 5.3653853819639705: 1, 5.3619327144101563: 1, 5.359716
7081251094: 1, 5.3576103317218422: 1, 5.3537887362971279: 1, 5.35127701928456
2: 1, 5.3480546466155143: 1, 5.3426140488572633: 1, 5.3424904239817597: 1, 5.
3406857628264017: 1, 5.333502909295933: 1, 5.3322892212086561: 1, 5.327354323
078521: 1, 5.3238770812642908: 1, 5.316315279417636: 1, 5.3139664281518524:
1, 5.3043553882933381: 1, 5.3003529350824579: 1, 5.2976566978425073: 1, 5.289
9988279203702: 1, 5.2788853251908305: 1, 5.2745901291141601: 1, 5.27026797001
03445: 1, 5.2681835781016959: 1, 5.2606434968232305: 1, 5.2588965531571183:
1, 5.2568027900634524: 1, 5.2510417914034084: 1, 5.2486301187643845: 1, 5.248
1033223844564: 1, 5.2476657465874332: 1, 5.2444122179176969: 1, 5.24409923396
08208: 1, 5.2432452666213996: 1, 5.240671153344211: 1, 5.2382204976327049: 1,
5.2374410247361016: 1, 5.2362331804703084: 1, 5.2359513015736896: 1, 5.230592
7329835331: 1, 5.2295617267407959: 1, 5.2247952396477242: 1, 5.22132913868533
07: 1, 5.2198219018924048: 1, 5.2182465812217353: 1, 5.2160916578861274: 1,
5.2154653868714336: 1, 5.2147665634671005: 1, 5.2145019077744319: 1, 5.210512
4850065074: 1, 5.2072056124623245: 1, 5.2060418257125587: 1, 5.20279699668692
34: 1, 5.2013957969781952: 1, 5.2004588147540822: 1, 5.1933337060488718: 1,
5.1928961348527611: 1, 5.1841475829151502: 1, 5.1831159544961469: 1, 5.173597
2073583225: 1, 5.1733808137814137: 1, 5.1727023292283372: 1, 5.16576727736180
62: 1, 5.1643870431166556: 1, 5.1615414643019939: 1, 5.1606980139946161: 1,
5.1560586099520407: 1, 5.1555180218431182: 1, 5.1538438498296779: 1, 5.151889
0773947295: 1, 5.1512729686285166: 1, 5.1459992734845734: 1, 5.14188261550642
43: 1, 5.1399320460880311: 1, 5.1323120824723469: 1, 5.1161308591341328: 1,
5.1155282919577294: 1, 5.1040422255424547: 1, 5.1028568040041149: 1, 5.099379
3759525161: 1, 5.0954240125654549: 1, 5.0946662431552907: 1, 5.08879714977470
78: 1, 5.0843246405784805: 1, 5.0829983134345955: 1, 5.0778948312920589: 1,
5.0700271074800254: 1, 5.0638444588713485: 1, 5.05942491343691: 1, 5.05939364
87902065: 1, 5.0593097857391873: 1, 5.052222960423201: 1, 5.0520263843766831:
1, 5.0519000715073163: 1, 5.0441545956432794: 1, 5.0420484068980533: 1, 5.041
6303010917511: 1, 5.0391046004698605: 1, 5.0370958754100617: 1, 5.03494786519
33284: 1, 5.0336860362428997: 1, 5.0286519649415231: 1, 5.0281417421721564:
1, 5.0275630569290426: 1, 5.0229346502299492: 1, 5.0225490553034202: 1, 5.018
622661434307: 1, 5.0141604098890129: 1, 5.0072621459363313: 1, 5.004893735240
1097: 1, 5.0038340093683349: 1, 5.0008001576368324: 1, 5.0006831253667396: 1,
4.9919909902200379: 1, 4.9840782960072536: 1, 4.9821404111232219: 1, 4.981523
9911037024: 1, 4.9769809474081068: 1, 4.973064445594539: 1, 4.973061453635928
3: 1, 4.9656145199452206: 1, 4.9650741689385214: 1, 4.9609414850401272: 1, 4.
9609247825530227: 1, 4.9605720795129598: 1, 4.9604395547869178: 1, 4.95441091
52130789: 1, 4.9508989742939695: 1, 4.9497113727680562: 1, 4.949384484814679
3: 1, 4.9469700918614503: 1, 4.9462635797868053: 1, 4.9286686187121695: 1, 4.
9280600458876398: 1, 4.922215536370234: 1, 4.9220212823175711: 1, 4.909628386
5424013: 1, 4.9082849972945626: 1, 4.9044700345642465: 1, 4.9004431576873744:
1, 4.8948779982736754: 1, 4.8931892933931973: 1, 4.8927686610821555: 1, 4.886
4084158933538: 1, 4.8804269271937919: 1, 4.8794664006692736: 1, 4.87877928029
97734: 1, 4.8734276695920995: 1, 4.872706995439394: 1, 4.8708288776155104: 1,
4.8678931311131777: 1, 4.865559152592124: 1, 4.8641401710414209: 1, 4.8521291
795881467: 1, 4.8517165734128538: 1, 4.8503524858828122: 1, 4.848051142125845
2: 1, 4.8427855272917641: 1, 4.8424553669524029: 1, 4.8418842545666987: 1, 4.
```

```
8415217950379237: 1, 4.8402412019369692: 1, 4.8341864658685987: 1, 4.83208632
82988418: 1, 4.8313929861689546: 1, 4.8252275883249238: 1, 4.822729335532278
2: 1, 4.8208759630941573: 1, 4.8192894519721854: 1, 4.8168380737608203: 1, 4.
8155869478189892: 1, 4.8145388929416439: 1, 4.8132508938377212: 1, 4.80912349
76475894: 1, 4.8007446109947489: 1, 4.7984620775553708: 1, 4.795150117960758
1: 1, 4.7921305429961851: 1, 4.792034289551955: 1, 4.7885740912515677: 1, 4.7
855256589708333: 1, 4.7837558677680718: 1, 4.7832831301522987: 1, 4.783188657
0424524: 1, 4.7780221026267533: 1, 4.7778481403490733: 1, 4.7735631893432497:
1, 4.7693875230633465: 1, 4.7671174377320433: 1, 4.7619783668563596: 1, 4.761
8726812447925: 1, 4.7608078677478813: 1, 4.7559652595202326: 1, 4.75511833766
96127: 1, 4.751849573611362: 1, 4.7505239939314956: 1, 4.7470685686282801: 1,
4.7415552216998513: 1, 4.7406399887595558: 1, 4.7388266260924388: 1, 4.738788
1648465218: 1, 4.7387836206134653: 1, 4.7383526849114297: 1, 4.73679740132687
5: 1, 4.7356022804262201: 1, 4.7327634744956439: 1, 4.72783305822065: 1, 4.72
75884033798112: 1, 4.7256816986860377: 1, 4.7243132960267591: 1, 4.7202872041
00452: 1, 4.7202540285688803: 1, 4.7187516342345734: 1, 4.7125118963655614:
1, 4.7116752099402168: 1, 4.7107677556619425: 1, 4.7013190828354308: 1, 4.701
0663245928814: 1, 4.6986139924290224: 1, 4.6961004869152232: 1, 4.69224445873
87302: 1, 4.6911942300096587: 1, 4.6910073470371421: 1, 4.6870534224946931:
1, 4.6853362210170921: 1, 4.6844939104342007: 1, 4.6831215103944661: 1, 4.678
0796529498128: 1, 4.6779855512132302: 1, 4.6724783615732486: 1, 4.67187710289
55581: 1, 4.6716432540608475: 1, 4.6666261494069303: 1, 4.662169922122561: 1,
4.6592237076688701: 1, 4.6547713547987479: 1, 4.6520205234122507: 1, 4.648220
5852757996: 1, 4.6468200038131959: 1, 4.6456267758701797: 1, 4.64388931924704
98: 1, 4.6374399976578404: 1, 4.6270811590187471: 1, 4.6261657358066506: 1,
4.6245470234049675: 1, 4.623222375526959: 1, 4.6181321623517597: 1, 4.6150842
448865417: 1, 4.614484313673592: 1, 4.6130411882327458: 1, 4.605472253124585
1: 1, 4.6029641059406758: 1, 4.5969048602659486: 1, 4.5916499297345608: 1, 4.
5892451152055491: 1, 4.5850052729203181: 1, 4.5839570060578181: 1, 4.57191629
36108244: 1, 4.5662796205953464: 1, 4.5502414549443539: 1, 4.549959696231255
2: 1, 4.5444826996553687: 1, 4.544339892504099: 1, 4.5427929031172036: 1, 4.5
412244719989365: 1, 4.5361932099546838: 1, 4.530368795707739: 1, 4.5295396166
527446: 1, 4.5293264464885405: 1, 4.5245804167388082: 1, 4.5222839932301726:
1, 4.522042566202285: 1, 4.5215897383032946: 1, 4.5213333969389984: 1, 4.5208
783941297455: 1, 4.5190243576770097: 1, 4.5176674949472648: 1, 4.511010881979
125: 1, 4.5109913113175466: 1, 4.5053493878135988: 1, 4.5051751836208389: 1,
4.4999802320425442: 1, 4.4953691111115202: 1, 4.4807422090908755: 1, 4.478773
379907091: 1, 4.4786615544077399: 1, 4.4781382842862811: 1, 4.478045913711437
9: 1, 4.4746003706516291: 1, 4.4737110090248073: 1, 4.4711351401581663: 1, 4.
4643113184439072: 1, 4.4638770886015049: 1, 4.4562078947140007: 1, 4.45510308
63187435: 1, 4.4470146914891284: 1, 4.4409672503522142: 1, 4.435917376979003
5: 1, 4.4342789023946594: 1, 4.4331925406078536: 1, 4.4260538457732856: 1, 4.
4222636491876273: 1, 4.4194590275192978: 1, 4.4142765748400423: 1, 4.41396881
29086769: 1, 4.4131702011349496: 1, 4.4118627193207365: 1, 4.407508080365937:
1, 4.4070860393807614: 1, 4.4022993284677607: 1, 4.3982584889414236: 1, 4.394
6799614218639: 1, 4.3842205603490036: 1, 4.3838267086951834: 1, 4.38366637343
08763: 1, 4.3811552923480299: 1, 4.3796172875245469: 1, 4.3749879631161921:
1, 4.3745563433237731: 1, 4.3727188904360226: 1, 4.3710426282016126: 1, 4.363
7108324474783: 1, 4.363118056379605: 1, 4.3605671852761825: 1, 4.358047756409
2899: 1, 4.3577216438539272: 1, 4.3561096854755803: 1, 4.3542407438959527: 1,
4.3520517834954804: 1, 4.3518256652871763: 1, 4.3513915548289797: 1, 4.349327
6363899449: 1, 4.3484774545187541: 1, 4.3481810030877455: 1, 4.34765188110856
83: 1, 4.3470453218823595: 1, 4.3444534805093848: 1, 4.3443893255537178: 1,
4.3422227564616467: 1, 4.334751321243723: 1, 4.331332419898045: 1, 4.33059987
06430353: 1, 4.3297662975635616: 1, 4.3188999132492611: 1, 4.31592990233964:
1, 4.3116781019335981: 1, 4.3013824972794685: 1, 4.3007020501435376: 1, 4.299
9589475804987: 1, 4.2998448681060033: 1, 4.2987936069465382: 1, 4.29854877789
```

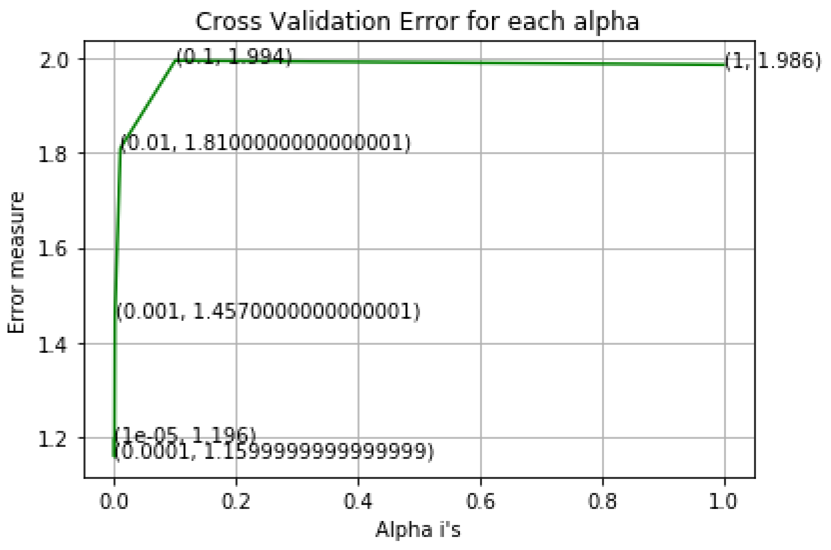
```
13365: 1, 4.2985181753953707: 1, 4.2972330301547546: 1, 4.2913783250571162:
1, 4.2828812713499298: 1, 4.2816912725770919: 1, 4.2795769086288908: 1, 4.270
714495987411: 1, 4.2704490378460536: 1, 4.2688300791379321: 1, 4.262596978838
2081: 1, 4.2618825460823695: 1, 4.2609922278872654: 1, 4.2606461846200432: 1,
4.2588902215619067: 1, 4.2532191078835853: 1, 4.2410269805636807: 1, 4.235743
0833384919: 1, 4.2274827638514569: 1, 4.2258542546726341: 1, 4.22479352171790
18: 1, 4.2199498870753009: 1, 4.2193804736358453: 1, 4.2188880373947244: 1,
4.2176358185943288: 1, 4.2170573127541822: 1, 4.2167009053800193: 1, 4.215256
4868919153: 1, 4.2124245448325874: 1, 4.2089377168908078: 1, 4.20877077878309
35: 1, 4.2084615836313262: 1, 4.2023259964443334: 1, 4.2006152300207669: 1,
4.1992300450580231: 1, 4.1901371931542908: 1, 4.1878725661310403: 1, 4.180230
636934084: 1, 4.1756354154995412: 1, 4.1692078804846959: 1, 4.164974902997864
9: 1, 4.1562734045377985: 1, 4.1547911254825287: 1, 4.1537605352801155: 1, 4.
1484931487226291: 1, 4.1477894920827865: 1, 4.1444758438822955: 1, 4.13581267
08632357: 1, 4.1333164146148169: 1, 4.1321598826758787: 1, 4.129148342475306
6: 1, 4.1281341374974865: 1, 4.1198075799203364: 1, 4.1192039783542755: 1, 4.
1178880397113655: 1, 4.1167132727350175: 1, 4.1138331013692326: 1, 4.11112970
25487987: 1, 4.110158237959836: 1, 4.1099990567915095: 1, 4.1099752855246745:
1, 4.1059233998204991: 1, 4.097260464499394: 1, 4.0945716556244234: 1, 4.0888
850615664367: 1, 4.0859507065450593: 1, 4.083993972994139: 1, 4.0819366559339
656: 1, 4.0805114581056596: 1, 4.069470505653034: 1, 4.0642061749018907: 1,
4.0610114981802834: 1, 4.0557534723128796: 1, 4.0515789961038848: 1, 4.049885
8180566426: 1, 4.0459050255072722: 1, 4.0419385549706757: 1, 4.04075720292633
46: 1, 4.0370496232663289: 1, 4.0316971725203539: 1, 4.0257291098600492: 1,
4.0241930105481787: 1, 4.0217803919435555: 1, 4.0211185509104421: 1, 4.018612
9430313153: 1, 4.0160658807788838: 1, 4.0097823104725485: 1, 4.00354249851669
32: 1, 4.0025449428932189: 1, 4.0016494095735204: 1, 4.0006795615048354: 1,
3.9996166942458973: 1, 3.9908481322061271: 1, 3.9895914858753456: 1, 3.983646
7090078362: 1, 3.981754606257776: 1, 3.9797642131857414: 1, 3.977712488654399
1: 1, 3.9769001875314931: 1, 3.9744909671182933: 1, 3.9741652881432783: 1, 3.
9706283088541454: 1, 3.9689895869145539: 1, 3.9679748321880468: 1, 3.96368863
76975551: 1, 3.9620635585772304: 1, 3.9613322946826899: 1, 3.960630624010118
8: 1, 3.9578461194777184: 1, 3.950451919879149: 1, 3.9480678797054374: 1, 3.9
392526114897382: 1, 3.9376979738986222: 1, 3.9335267201663164: 1, 3.932920501
2078536: 1, 3.9266847996475209: 1, 3.9248830075457835: 1, 3.9241887127617954:
1, 3.9175040763068689: 1, 3.9157782852841136: 1, 3.9135631807641942: 1, 3.912
6257680792502: 1, 3.9052513516060179: 1, 3.9009799406110859: 1, 3.89783611139
68556: 1, 3.8976589352455373: 1, 3.8974105036905593: 1, 3.8973024660720914:
1, 3.8964090500541024: 1, 3.8955721590374712: 1, 3.8919559800410641: 1, 3.885
2227477821422: 1, 3.8829327704258181: 1, 3.8821018081615146: 1, 3.87945621714
01932: 1, 3.8791281997772806: 1, 3.874491600066611: 1, 3.8737895878266211: 1,
3.8725069117081432: 1, 3.8710998289475582: 1, 3.8679034705914006: 1, 3.855635
2444497928: 1, 3.8511109391575817: 1, 3.8486450370298466: 1, 3.83677637133089
8: 1, 3.8349508198475624: 1, 3.8319045750820333: 1, 3.829440870618031: 1, 3.8
234186593230839: 1, 3.8189294895827257: 1, 3.8161049414910098: 1, 3.815418954
3168986: 1, 3.8104488085093311: 1, 3.803927827682279: 1, 3.8023301332004387:
1, 3.801918778762825: 1, 3.7937457111501227: 1, 3.7936807156331045: 1, 3.7931
850035262751: 1, 3.7856101789271444: 1, 3.7740220393967947: 1, 3.773200279033
504: 1, 3.771704904893713: 1, 3.7706754581134874: 1, 3.7655332892944822: 1,
3.7642772019293647: 1, 3.761514971224531: 1, 3.7557844412042485: 1, 3.7535846
783922517: 1, 3.7511540413437463: 1, 3.7487782385471471: 1, 3.727801711207621
2: 1, 3.7266906442238685: 1, 3.7190449623020445: 1, 3.7163755829663208: 1, 3.
7156640230837583: 1, 3.7078836713529375: 1, 3.6984557825995878: 1, 3.69760942
33354626: 1, 3.6917921423363715: 1, 3.6892562606407471: 1, 3.683620954525035
4: 1, 3.6746094820056419: 1, 3.6736027187067157: 1, 3.6682370989222401: 1, 3.
66528821585861: 1, 3.6646287728087632: 1, 3.663891861275097: 1, 3.66367442735
64918: 1, 3.6567287666786923: 1, 3.6500779726803803: 1, 3.6451626210641201:
```

```
1, 3.6402149285270315: 1, 3.6341077272384963: 1, 3.6335729155883234: 1, 3.629
5871240424136: 1, 3.6159913217389388: 1, 3.6097735382572855: 1, 3.60848347165
33232: 1, 3.6054896865938821: 1, 3.5950261241105577: 1, 3.5937979814231165:
1, 3.5932821008867957: 1, 3.5910560860496448: 1, 3.5816935449376714: 1, 3.560
7899034894106: 1, 3.5569160298286278: 1, 3.5460742644051226: 1, 3.53134391460
14871: 1, 3.5226164900167829: 1, 3.5187681218930331: 1, 3.5162834831635577:
1, 3.5145832815385267: 1, 3.5132964928875379: 1, 3.5096499226309401: 1, 3.507
5445613422516: 1, 3.4987931914791117: 1, 3.4955834063067082: 1, 3.48779123493
10503: 1, 3.4777179489623289: 1, 3.4673244451145848: 1, 3.4654264301193138:
1, 3.4648825830489565: 1, 3.4642911817372086: 1, 3.4585243476297718: 1, 3.452
9210428340273: 1, 3.4464761894966092: 1, 3.4458125114726035: 1, 3.43616815057
54378: 1, 3.4267550871944672: 1, 3.4266099451741812: 1, 3.4088197929273929:
1, 3.4006253270660141: 1, 3.4002855245781189: 1, 3.3848400103715051: 1, 3.379
0954949847851: 1, 3.3789234267133228: 1, 3.3671599927216866: 1, 3.35796137667
23213: 1, 3.3517555861185087: 1, 3.3431441321035269: 1, 3.3313607903989237:
1, 3.3265200131745689: 1, 3.3224518210720713: 1, 3.3173769189049307: 1, 3.316
2986501461824: 1, 3.312525116951063: 1, 3.3104754692847878: 1, 3.300206244393
3662: 1, 3.2987582261885358: 1, 3.2859598755046777: 1, 3.279863091541698: 1,
3.2653072324453776: 1, 3.2617778534874464: 1, 3.2579742676441814: 1, 3.257854
4557970868: 1, 3.2477444847819048: 1, 3.245288922958137: 1, 3.23791989654729
4: 1, 3.2123813508891832: 1, 3.2041844261724344: 1, 3.1577602239677587: 1, 3.
1442126150372092: 1, 3.1363558686460125: 1, 3.1175142164303082: 1, 3.10922170
58184604: 1, 3.0876704062176095: 1, 3.0748926511041845: 1, 2.951936795705911
7: 1, 2.9461957824335907: 1, 2.9305630064901043: 1, 2.9251927191671174: 1, 2.
8396009775112816: 1, 2.7842819986502798: 1, 2.7828241516102472: 1, 2.76909674
912757: 1, 2.6870550294423632: 1})
```

In [80]:
```
# 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/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
```

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

```
For values of alpha =  1e-05 The log loss is: 1.1960512212
For values of alpha =  0.0001 The log loss is: 1.15973355233
For values of alpha =  0.001 The log loss is: 1.45675696322
For values of alpha =  0.01 The log loss is: 1.80950836141
For values of alpha =  0.1 The log loss is: 1.9944697757
For values of alpha =  1 The log loss is: 1.98564379411
```



```
For values of best alpha =  0.0001 The train log loss is: 0.720170212981
For values of best alpha =  0.0001 The cross validation log loss is: 1.159733
55233
For values of best alpha =  0.0001 The test log loss is: 1.18788114141
```

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 = TfidfVectorizer(min_df=3, max_features=2000)
    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")
```

```
94.8 % of word of test data appeared in train data
93.2 % 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 bel
ongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_
y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [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 [95]:
```
# 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 = TfidfVectorizer(min_df=3, max_features=2000)

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

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

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

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

Stacking the three types of features

In [96]:
```
# 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_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,t
rain_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,tes
t_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_vari
ation_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_fea
ture_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_r
esponseCoding))
```

In [97]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 4200)
(number of data points * number of features) in test data =  (665, 4200)
(number of data points * number of features) in cross validation data = (532,
4200)
```

In [98]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_r
esponseCoding.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

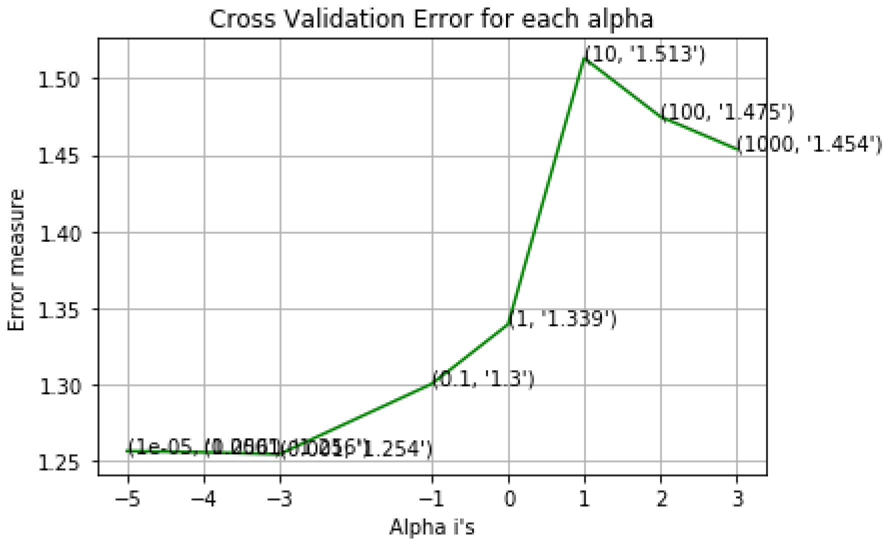
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [99]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

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


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

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i
]))
```

```
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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


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

```
for alpha = 1e-05
Log Loss : 1.25644765516
for alpha = 0.0001
Log Loss : 1.25571030272
for alpha = 0.001
Log Loss : 1.25438503882
for alpha = 0.1
Log Loss : 1.30034720989
for alpha = 1
Log Loss : 1.3393359547
for alpha = 10
Log Loss : 1.51324784642
for alpha = 100
Log Loss : 1.47502706228
for alpha = 1000
Log Loss : 1.45384818142
```

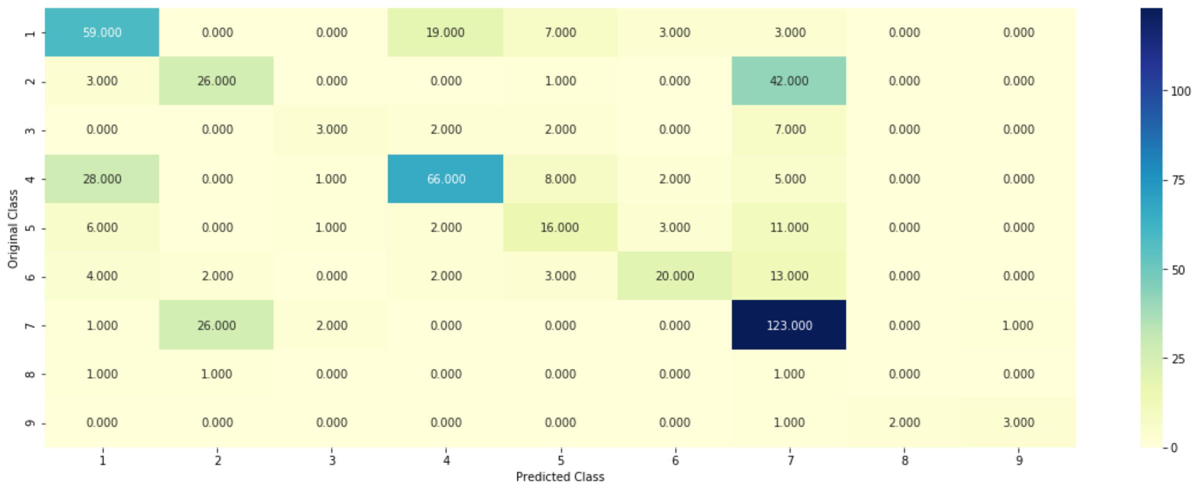


```
For values of best alpha =  0.001 The train log loss is: 0.567522256295
For values of best alpha =  0.001 The cross validation log loss is: 1.2543850
3882
For values of best alpha =  0.001 The test log loss is: 1.28478867977
```

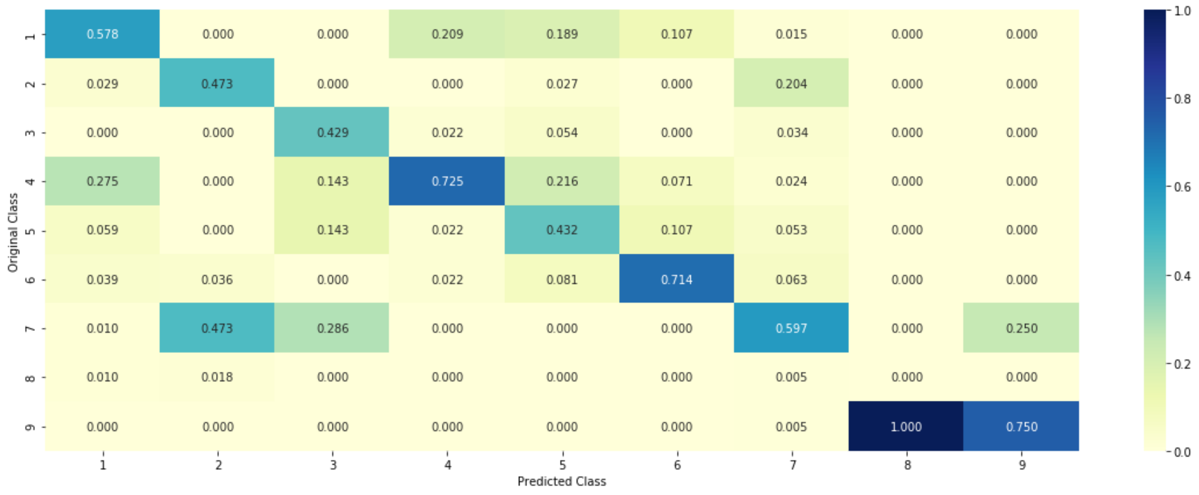
4.1.1.2. Testing the model with best hyper paramters

In [100]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

# 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------

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

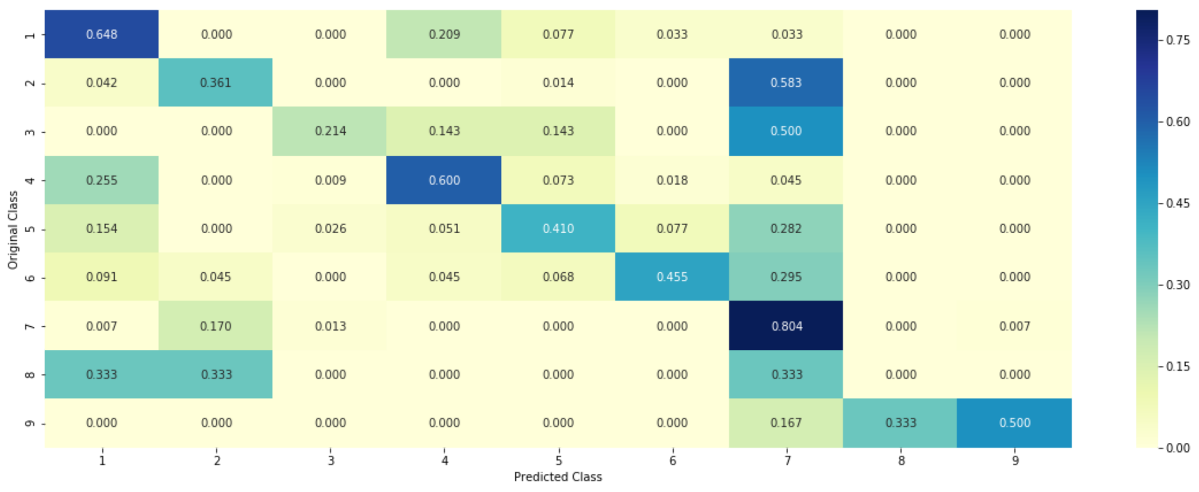
```
Log Loss : 1.25438503882
Number of missclassified point : 0.40601503759398494
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [106]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0705  0.0936  0.0117  0.0715  0.0353  0.0
346  0.676   0.0037  0.0032]]
Actual Class : 7
--------------------------------------------------
15 Text feature [activation] present in test data point [True]
16 Text feature [activated] present in test data point [True]
18 Text feature [cells] present in test data point [True]
20 Text feature [signaling] present in test data point [True]
21 Text feature [expressed] present in test data point [True]
26 Text feature [inhibition] present in test data point [True]
27 Text feature [also] present in test data point [True]
29 Text feature [indeed] present in test data point [True]
30 Text feature [grown] present in test data point [True]
31 Text feature [cell] present in test data point [True]
32 Text feature [activating] present in test data point [True]
33 Text feature [hours] present in test data point [True]
35 Text feature [shown] present in test data point [True]
36 Text feature [compared] present in test data point [True]
37 Text feature [10] present in test data point [True]
40 Text feature [measured] present in test data point [True]
41 Text feature [addition] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [phosphorylated] present in test data point [True]
44 Text feature [mutational] present in test data point [True]
45 Text feature [suggest] present in test data point [True]
46 Text feature [treated] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [mutagenesis] present in test data point [True]
51 Text feature [treatment] present in test data point [True]
52 Text feature [previous] present in test data point [True]
53 Text feature [constitutive] present in test data point [True]
54 Text feature [materials] present in test data point [True]
55 Text feature [high] present in test data point [True]
60 Text feature [recent] present in test data point [True]
62 Text feature [sensitive] present in test data point [True]
65 Text feature [including] present in test data point [True]
66 Text feature [showed] present in test data point [True]
69 Text feature [inhibiting] present in test data point [True]
74 Text feature [inhibitor] present in test data point [True]
75 Text feature [activate] present in test data point [True]
76 Text feature [respectively] present in test data point [True]
77 Text feature [eight] present in test data point [True]
78 Text feature [without] present in test data point [True]
79 Text feature [tyrosine] present in test data point [True]
80 Text feature [considered] present in test data point [True]
82 Text feature [mutated] present in test data point [True]
83 Text feature [absence] present in test data point [True]
86 Text feature [differences] present in test data point [True]
87 Text feature [two] present in test data point [True]
89 Text feature [results] present in test data point [True]
94 Text feature [active] present in test data point [True]
98 Text feature [1a] present in test data point [True]
Out of the top  100  features  48 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [107]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0706  0.0589  0.0108  0.7033  0.0341  0.0
325  0.0835  0.0033  0.0029]]
Actual Class : 2
--------------------------------------------------
11 Text feature [activity] present in test data point [True]
13 Text feature [prostate] present in test data point [True]
16 Text feature [protein] present in test data point [True]
17 Text feature [acid] present in test data point [True]
18 Text feature [results] present in test data point [True]
21 Text feature [whereas] present in test data point [True]
22 Text feature [whether] present in test data point [True]
24 Text feature [minutes] present in test data point [True]
25 Text feature [shown] present in test data point [True]
26 Text feature [determine] present in test data point [True]
28 Text feature [importance] present in test data point [True]
31 Text feature [also] present in test data point [True]
32 Text feature [catalytic] present in test data point [True]
35 Text feature [suppressor] present in test data point [True]
37 Text feature [type] present in test data point [True]
39 Text feature [mutational] present in test data point [True]
40 Text feature [functionally] present in test data point [True]
42 Text feature [function] present in test data point [True]
44 Text feature [two] present in test data point [True]
45 Text feature [indicate] present in test data point [True]
46 Text feature [although] present in test data point [True]
47 Text feature [related] present in test data point [True]
48 Text feature [30] present in test data point [True]
49 Text feature [index] present in test data point [True]
50 Text feature [materials] present in test data point [True]
52 Text feature [wild] present in test data point [True]
53 Text feature [thus] present in test data point [True]
57 Text feature [eight] present in test data point [True]
60 Text feature [contained] present in test data point [True]
61 Text feature [similar] present in test data point [True]
63 Text feature [associated] present in test data point [True]
65 Text feature [bind] present in test data point [True]
66 Text feature [analyzed] present in test data point [True]
68 Text feature [therefore] present in test data point [True]
69 Text feature [tagged] present in test data point [True]
70 Text feature [three] present in test data point [True]
71 Text feature [buffer] present in test data point [True]
72 Text feature [previous] present in test data point [True]
74 Text feature [mutant] present in test data point [True]
75 Text feature [express] present in test data point [True]
77 Text feature [suggesting] present in test data point [True]
79 Text feature [low] present in test data point [True]
81 Text feature [hours] present in test data point [True]
82 Text feature [within] present in test data point [True]
83 Text feature [analysis] present in test data point [True]
85 Text feature [terminal] present in test data point [True]
86 Text feature [generate] present in test data point [True]
87 Text feature [using] present in test data point [True]
89 Text feature [laboratory] present in test data point [True]
90 Text feature [gene] present in test data point [True]
91 Text feature [addition] present in test data point [True]
96 Text feature [role] present in test data point [True]
```

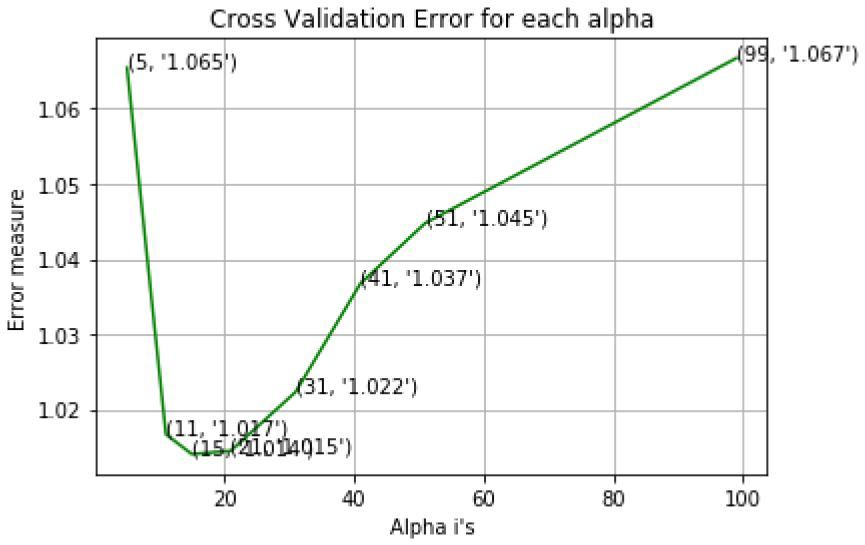
```
97 Text feature [per] present in test data point [True]
Out of the top  100  features  53 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [109]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = 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 i
s:",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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.06542184781
for alpha = 11
Log Loss : 1.01670254895
for alpha = 15
Log Loss : 1.01407011841
for alpha = 21
Log Loss : 1.01450737488
for alpha = 31
Log Loss : 1.0223245545
for alpha = 41
Log Loss : 1.03674360711
for alpha = 51
Log Loss : 1.04477641507
for alpha = 99
Log Loss : 1.0667031997
```

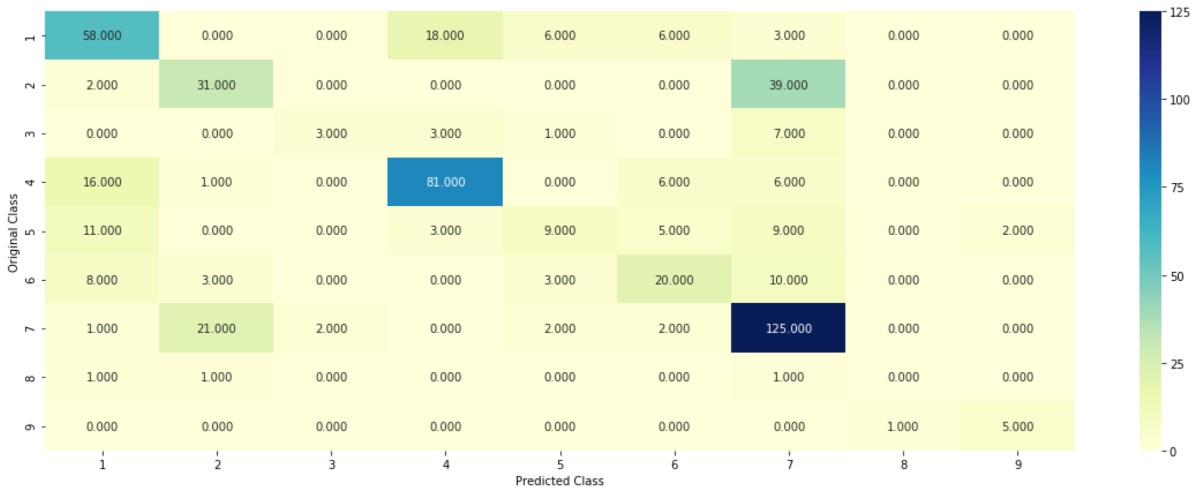


```
For values of best alpha =  15 The train log loss is: 0.673042085942
For values of best alpha =  15 The cross validation log loss is: 1.0140701184
1
For values of best alpha =  15 The test log loss is: 1.12012807589
```

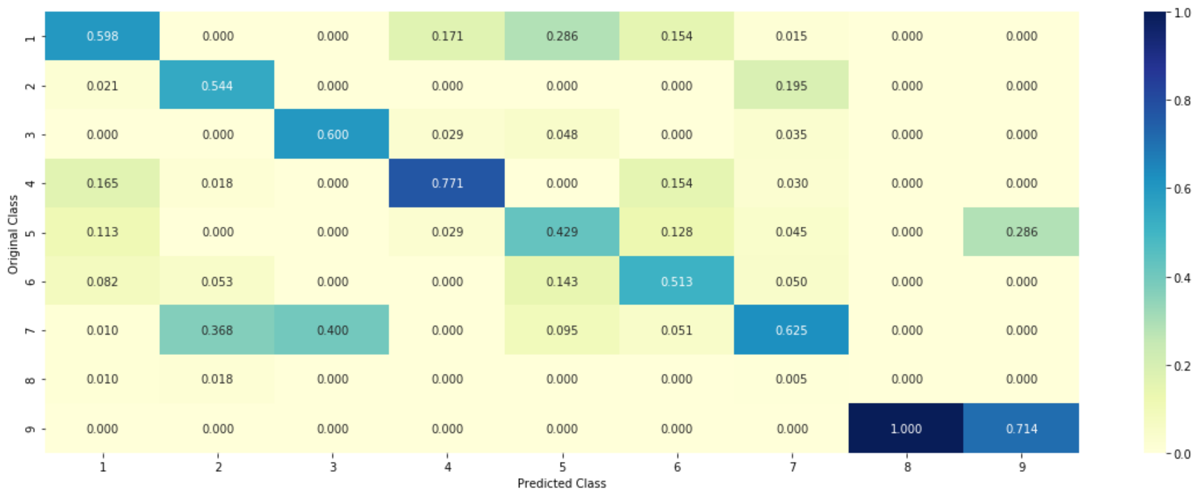
4.2.2. Testing the model with best hyper paramters

In [110]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

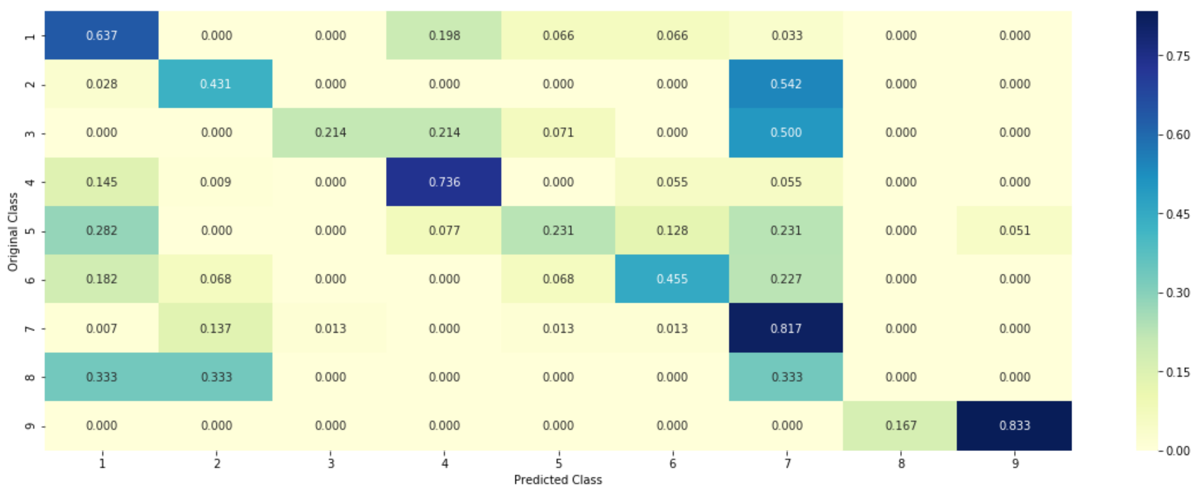
```
Log loss : 1.01407011841
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

In [113]:
```
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 : 7
Actual Class : 2
The  15  nearest neighbours of the test points belongs to classes [2 4 4 6 4
4 4 4 4 4 4 4 4 4 4]
Fequency of nearest points : Counter({4: 13, 2: 1, 6: 1})
```

4.2.4. Sample Query Point-2

In [114]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belon
gs to classes [4 4 4 7 7 7 7 7 7 7 7 7 7 2 7]
Fequency of nearest points : Counter({7: 11, 4: 3, 2: 1})
```

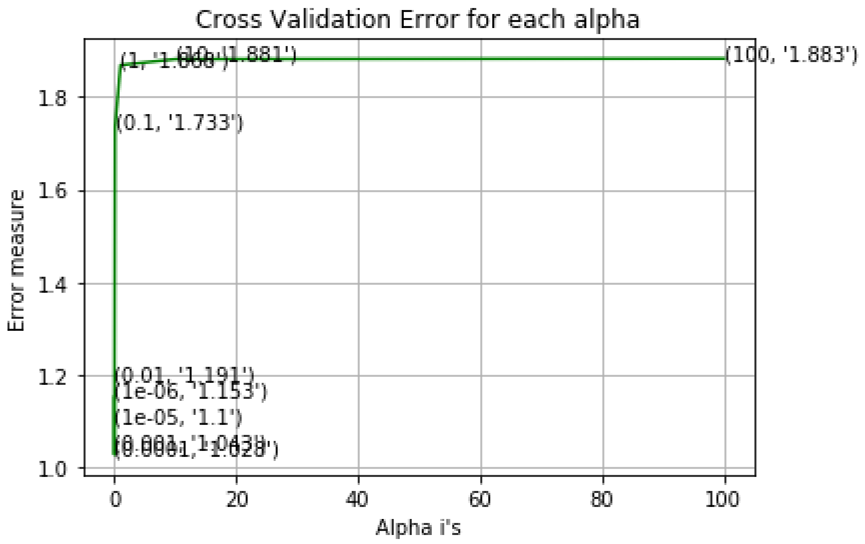
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [115]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', 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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.15254206517
for alpha = 1e-05
Log Loss : 1.10007837243
for alpha = 0.0001
Log Loss : 1.02784647818
for alpha = 0.001
Log Loss : 1.04344030844
for alpha = 0.01
Log Loss : 1.19059569788
for alpha = 0.1
Log Loss : 1.73346089167
for alpha = 1
Log Loss : 1.86846504731
for alpha = 10
Log Loss : 1.88137426353
for alpha = 100
Log Loss : 1.88268440907
```

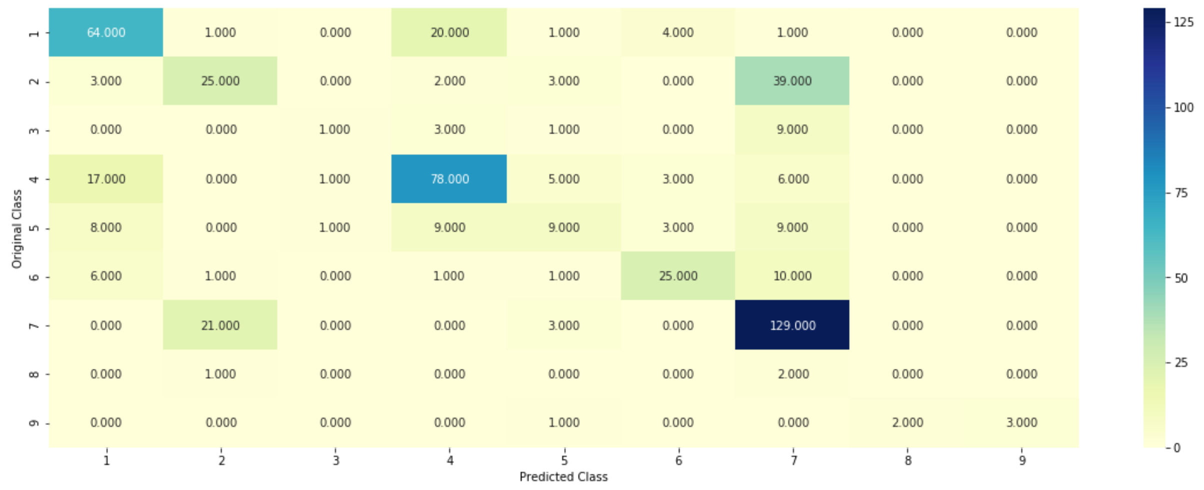


```
For values of best alpha =  0.0001 The train log loss is: 0.416398918875
For values of best alpha =  0.0001 The cross validation log loss is: 1.027846
47818
For values of best alpha =  0.0001 The test log loss is: 1.06247316449
```

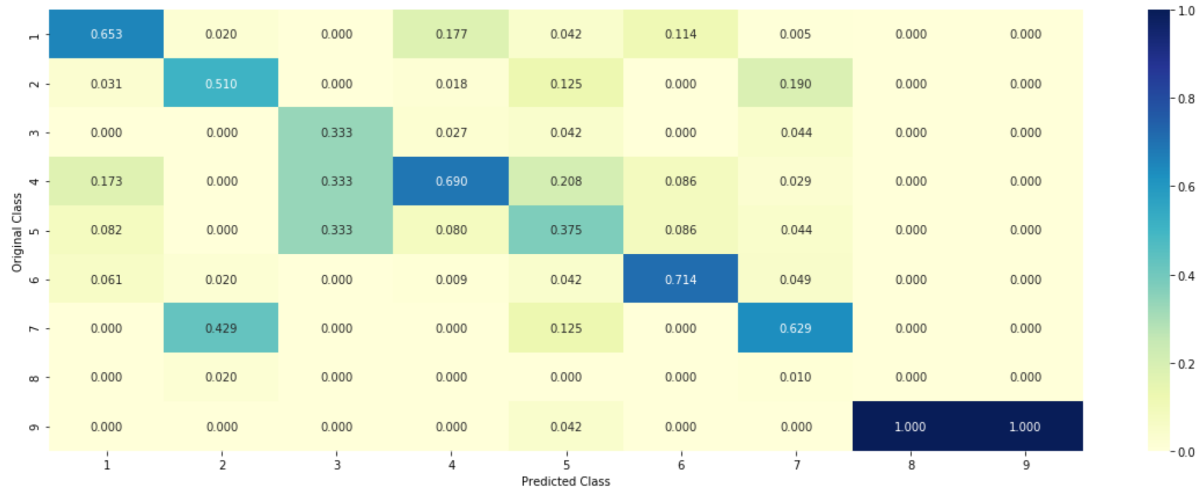
4.3.1.2. Testing the model with best hyper paramters

In [116]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

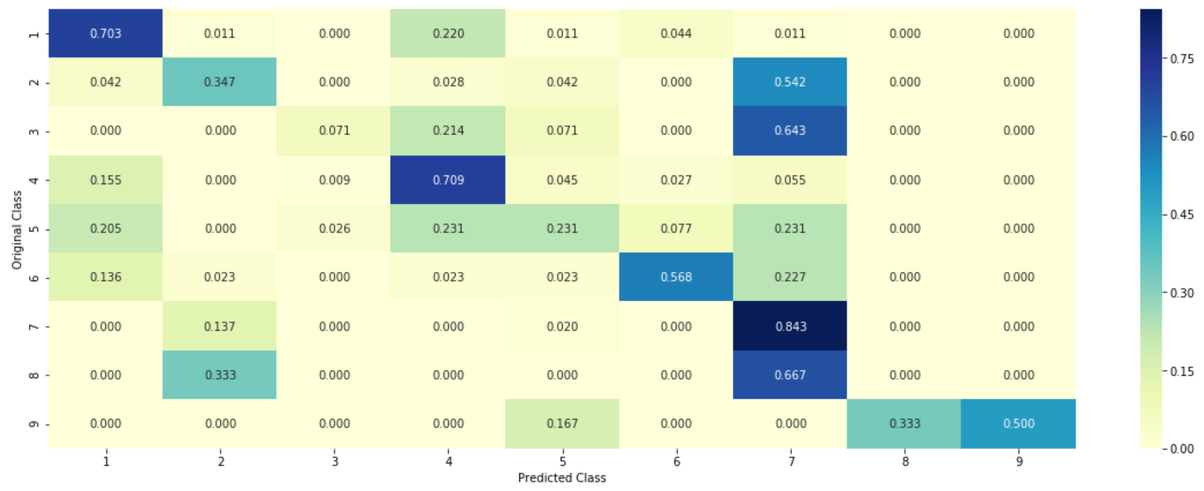
```
Log loss : 1.02784647818
Number of mis-classified points : 0.372180451127819556
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

```
In [118]: def get_imp_feature_names(text, indices, removed_ind = []):
              word_present = 0
              tabulte_list = []
              incresingorder_ind = 0
              for i in indices:
                  if i < train_gene_feature_onehotCoding.shape[1]:
                      tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                  elif i< 18:
                      tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                  if ((i > 17) & (i not in removed_ind)) :
                      word = train_text_features[i]
                      yes_no = True if word in text.split() else False
                      if yes_no:
                          word_present += 1
                      tabulte_list.append([incresingorder_ind,train_text_features[i], ye
          s_no])
                  incresingorder_ind += 1
              print(word_present, "most importent features are present in our query poin
          t")
              print("-"*50)
              print("The features that are most importent of the ",predicted_cls[0]," cl
          ass:")
              print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or
           Not']))
```

4.3.1.3.1. Correctly Classified point

In [121]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0117  0.1213  0.008   0.2716  0.0355  0.0
202  0.519   0.012   0.0008]]
Actual Class : 7
--------------------------------------------------
10 Text feature [activated] present in test data point [True]
35 Text feature [transformed] present in test data point [True]
61 Text feature [signaling] present in test data point [True]
69 Text feature [activation] present in test data point [True]
80 Text feature [receptor] present in test data point [True]
98 Text feature [life] present in test data point [True]
101 Text feature [us] present in test data point [True]
105 Text feature [ligand] present in test data point [True]
124 Text feature [institute] present in test data point [True]
149 Text feature [activate] present in test data point [True]
160 Text feature [constitutive] present in test data point [True]
173 Text feature [phosphate] present in test data point [True]
202 Text feature [overexpressed] present in test data point [True]
218 Text feature [cancers] present in test data point [True]
254 Text feature [dose] present in test data point [True]
260 Text feature [hence] present in test data point [True]
268 Text feature [wt] present in test data point [True]
271 Text feature [expressed] present in test data point [True]
288 Text feature [contrast] present in test data point [True]
340 Text feature [profiles] present in test data point [True]
363 Text feature [inhibition] present in test data point [True]
370 Text feature [induced] present in test data point [True]
372 Text feature [independently] present in test data point [True]
376 Text feature [promega] present in test data point [True]
395 Text feature [3t3] present in test data point [True]
397 Text feature [akt1] present in test data point [True]
398 Text feature [previously] present in test data point [True]
406 Text feature [obtained] present in test data point [True]
411 Text feature [malignancies] present in test data point [True]
422 Text feature [factor] present in test data point [True]
441 Text feature [without] present in test data point [True]
447 Text feature [respect] present in test data point [True]
494 Text feature [activating] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [123]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0186  0.1669  0.0072  0.6488  0.0473  0.0
678  0.0379  0.0036  0.002 ]]
Actual Class : 2
--------------------------------------------------
52 Text feature [suppressor] present in test data point [True]
119 Text feature [none] present in test data point [True]
134 Text feature [importantly] present in test data point [True]
151 Text feature [mm] present in test data point [True]
154 Text feature [tumorigenesis] present in test data point [True]
164 Text feature [minutes] present in test data point [True]
169 Text feature [laboratory] present in test data point [True]
188 Text feature [contribute] present in test data point [True]
192 Text feature [plus] present in test data point [True]
208 Text feature [demonstrated] present in test data point [True]
214 Text feature [ligase] present in test data point [True]
222 Text feature [western] present in test data point [True]
233 Text feature [293t] present in test data point [True]
245 Text feature [del] present in test data point [True]
285 Text feature [ii] present in test data point [True]
288 Text feature [year] present in test data point [True]
301 Text feature [polymorphism] present in test data point [True]
310 Text feature [catalytic] present in test data point [True]
317 Text feature [hd] present in test data point [True]
322 Text feature [may] present in test data point [True]
323 Text feature [cytokine] present in test data point [True]
329 Text feature [function] present in test data point [True]
331 Text feature [plates] present in test data point [True]
335 Text feature [heterozygous] present in test data point [True]
360 Text feature [phase] present in test data point [True]
362 Text feature [loss] present in test data point [True]
370 Text feature [plasmid] present in test data point [True]
377 Text feature [substrate] present in test data point [True]
380 Text feature [index] present in test data point [True]
385 Text feature [suggesting] present in test data point [True]
386 Text feature [nucleotide] present in test data point [True]
391 Text feature [possible] present in test data point [True]
393 Text feature [sigma] present in test data point [True]
395 Text feature [domains] present in test data point [True]
407 Text feature [along] present in test data point [True]
415 Text feature [tyrosine] present in test data point [True]
418 Text feature [regulating] present in test data point [True]
425 Text feature [terms] present in test data point [True]
430 Text feature [show] present in test data point [True]
433 Text feature [e2] present in test data point [True]
434 Text feature [ca] present in test data point [True]
437 Text feature [majority] present in test data point [True]
447 Text feature [transferred] present in test data point [True]
461 Text feature [allelic] present in test data point [True]
464 Text feature [third] present in test data point [True]
468 Text feature [decreased] present in test data point [True]
470 Text feature [whether] present in test data point [True]
472 Text feature [observed] present in test data point [True]
474 Text feature [either] present in test data point [True]
476 Text feature [one] present in test data point [True]
481 Text feature [made] present in test data point [True]
491 Text feature [properties] present in test data point [True]
```

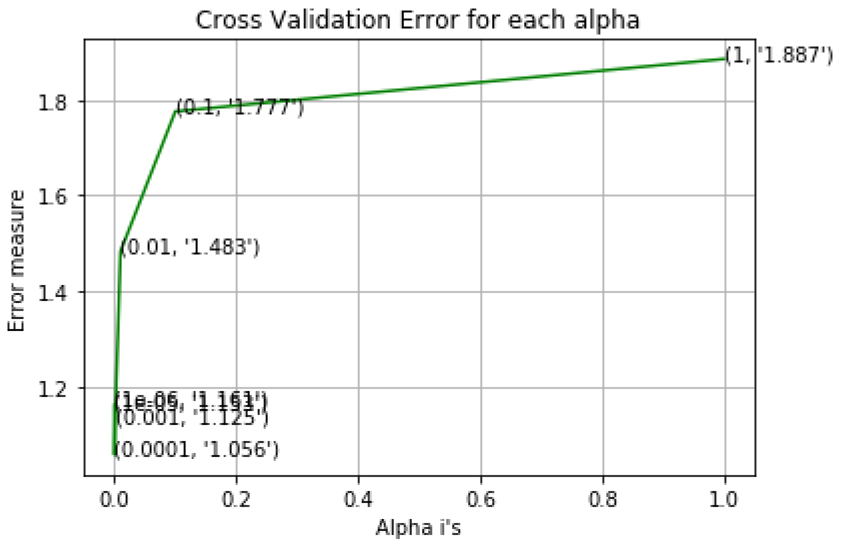
```
492 Text feature [medium] present in test data point [True]
497 Text feature [representative] present in test data point [True]
498 Text feature [separated] present in test data point [True]
Out of the top  500  features  55 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [124]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.16091267415
for alpha = 1e-05
Log Loss : 1.15304955486
for alpha = 0.0001
Log Loss : 1.05644863491
for alpha = 0.001
Log Loss : 1.12527817283
for alpha = 0.01
Log Loss : 1.48253075822
for alpha = 0.1
Log Loss : 1.77657189411
for alpha = 1
Log Loss : 1.88667284284
```

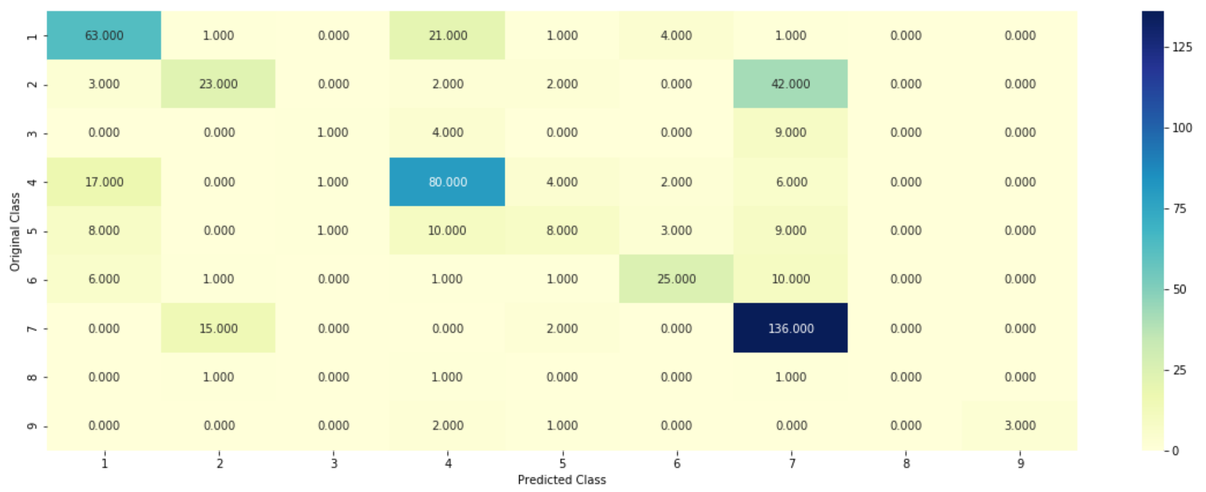


```
For values of best alpha =  0.0001 The train log loss is: 0.410436392156
For values of best alpha =  0.0001 The cross validation log loss is: 1.056448
63491
For values of best alpha =  0.0001 The test log loss is: 1.09678933769
```

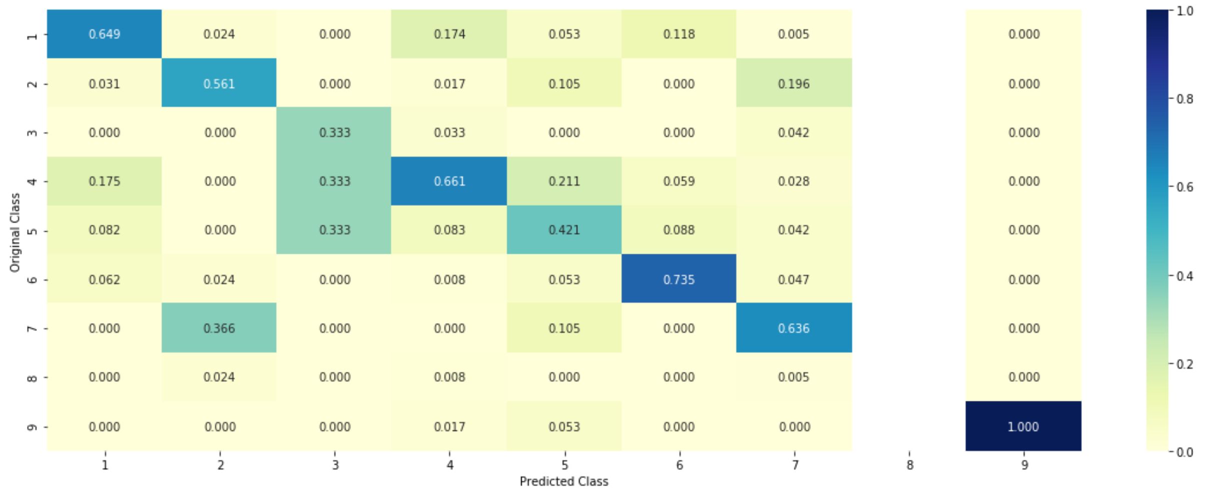
4.3.2.2. Testing model with best hyper parameters

In [125]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCo
ding, cv_y, clf)
```

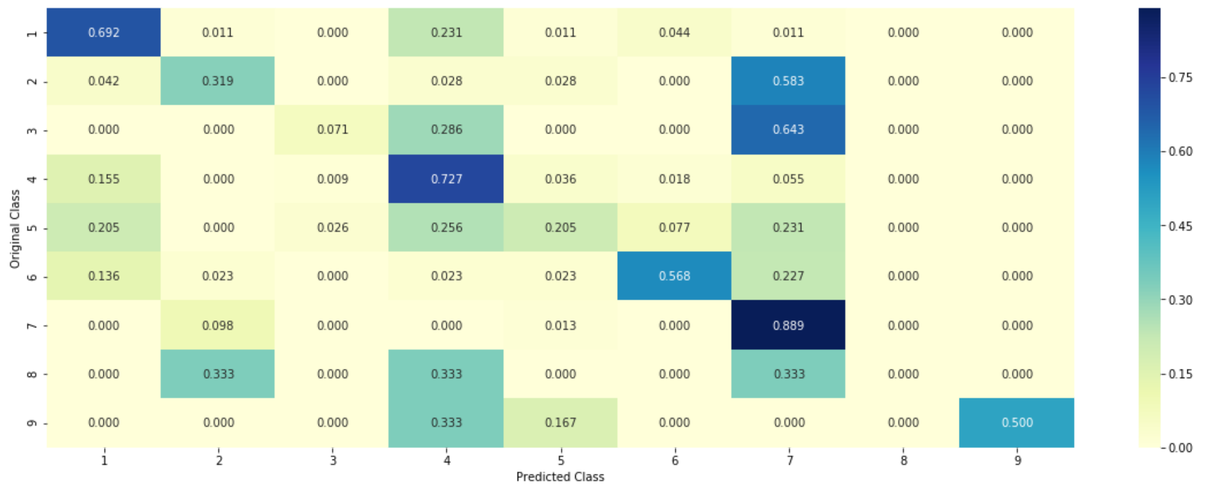
```
Log loss : 1.05644863491
Number of mis-classified points : 0.362781954887218O4
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [127]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[  1.34000000e-02   1.09600000e-01   5.100000
00e-03   3.33300000e-01
    3.18000000e-02   1.67000000e-02   4.85800000e-01   3.90000000e-03
    4.00000000e-04]]
Actual Class : 7
--------------------------------------------------
25 Text feature [activated] present in test data point [True]
95 Text feature [transformed] present in test data point [True]
123 Text feature [signaling] present in test data point [True]
135 Text feature [activation] present in test data point [True]
136 Text feature [institute] present in test data point [True]
139 Text feature [receptor] present in test data point [True]
160 Text feature [us] present in test data point [True]
161 Text feature [life] present in test data point [True]
164 Text feature [ligand] present in test data point [True]
212 Text feature [cancers] present in test data point [True]
252 Text feature [activate] present in test data point [True]
265 Text feature [constitutive] present in test data point [True]
271 Text feature [phosphate] present in test data point [True]
277 Text feature [contrast] present in test data point [True]
283 Text feature [wt] present in test data point [True]
284 Text feature [expressed] present in test data point [True]
286 Text feature [overexpressed] present in test data point [True]
287 Text feature [hence] present in test data point [True]
302 Text feature [profiles] present in test data point [True]
320 Text feature [previously] present in test data point [True]
333 Text feature [independently] present in test data point [True]
355 Text feature [dose] present in test data point [True]
382 Text feature [without] present in test data point [True]
392 Text feature [inhibition] present in test data point [True]
427 Text feature [respect] present in test data point [True]
429 Text feature [akt1] present in test data point [True]
432 Text feature [obtained] present in test data point [True]
488 Text feature [considered] present in test data point [True]
492 Text feature [malignancies] present in test data point [True]
493 Text feature [transduced] present in test data point [True]
494 Text feature [positive] present in test data point [True]
497 Text feature [factor] present in test data point [True]
499 Text feature [3t3] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [128]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0183  0.1687  0.0055  0.6524  0.0456  0.0
635  0.0414  0.0033  0.0013]]
Actual Class : 2
--------------------------------------------------
48 Text feature [suppressor] present in test data point [True]
116 Text feature [importantly] present in test data point [True]
134 Text feature [none] present in test data point [True]
143 Text feature [mm] present in test data point [True]
168 Text feature [tumorigenesis] present in test data point [True]
180 Text feature [laboratory] present in test data point [True]
181 Text feature [contribute] present in test data point [True]
184 Text feature [plus] present in test data point [True]
188 Text feature [demonstrated] present in test data point [True]
190 Text feature [minutes] present in test data point [True]
206 Text feature [western] present in test data point [True]
232 Text feature [ligase] present in test data point [True]
234 Text feature [293t] present in test data point [True]
248 Text feature [del] present in test data point [True]
278 Text feature [may] present in test data point [True]
296 Text feature [hd] present in test data point [True]
297 Text feature [year] present in test data point [True]
312 Text feature [polymorphism] present in test data point [True]
314 Text feature [function] present in test data point [True]
327 Text feature [tyrosine] present in test data point [True]
330 Text feature [cytokine] present in test data point [True]
332 Text feature [heterozygous] present in test data point [True]
333 Text feature [suggesting] present in test data point [True]
338 Text feature [ii] present in test data point [True]
350 Text feature [plates] present in test data point [True]
358 Text feature [possible] present in test data point [True]
360 Text feature [loss] present in test data point [True]
367 Text feature [along] present in test data point [True]
375 Text feature [index] present in test data point [True]
379 Text feature [third] present in test data point [True]
383 Text feature [catalytic] present in test data point [True]
388 Text feature [substrate] present in test data point [True]
397 Text feature [show] present in test data point [True]
403 Text feature [majority] present in test data point [True]
414 Text feature [phase] present in test data point [True]
417 Text feature [e2] present in test data point [True]
419 Text feature [domains] present in test data point [True]
424 Text feature [plasmid] present in test data point [True]
427 Text feature [regulating] present in test data point [True]
428 Text feature [ca] present in test data point [True]
433 Text feature [sigma] present in test data point [True]
435 Text feature [transferred] present in test data point [True]
437 Text feature [terms] present in test data point [True]
446 Text feature [expected] present in test data point [True]
450 Text feature [properties] present in test data point [True]
451 Text feature [whether] present in test data point [True]
456 Text feature [either] present in test data point [True]
457 Text feature [medium] present in test data point [True]
460 Text feature [made] present in test data point [True]
463 Text feature [separated] present in test data point [True]
474 Text feature [nucleotide] present in test data point [True]
478 Text feature [ha] present in test data point [True]
```

```
489 Text feature [observed] present in test data point [True]
490 Text feature [sds] present in test data point [True]
499 Text feature [allelic] present in test data point [True]
Out of the top  500  features  55 are present in query point
```

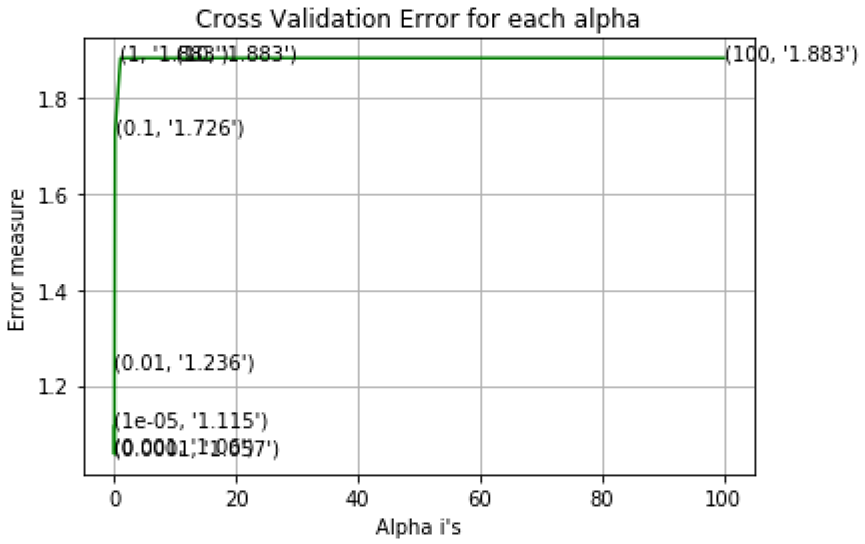
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [129]:
```
# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss=
'hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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


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

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

```
for C = 1e-05
Log Loss : 1.11477588289
for C = 0.0001
Log Loss : 1.05659891897
for C = 0.001
Log Loss : 1.05978031326
for C = 0.01
Log Loss : 1.23579408587
for C = 0.1
Log Loss : 1.7255274417
for C = 1
Log Loss : 1.88284779035
for C = 10
Log Loss : 1.88284782707
for C = 100
Log Loss : 1.88284781507
```

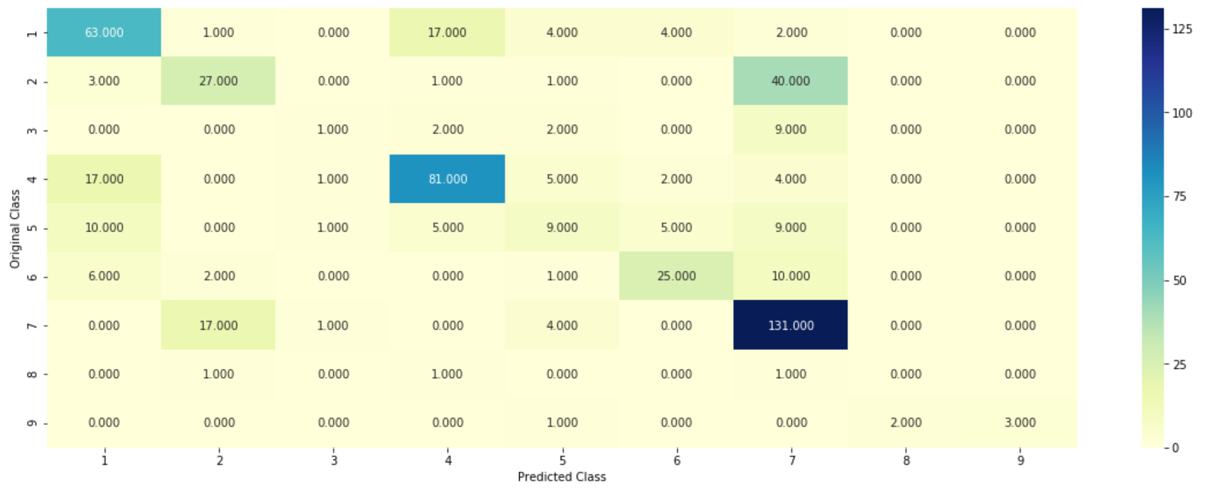


```
For values of best alpha =  0.0001 The train log loss is: 0.484990336912
For values of best alpha =  0.0001 The cross validation log loss is: 1.056598
91897
For values of best alpha =  0.0001 The test log loss is: 1.10493482947
```

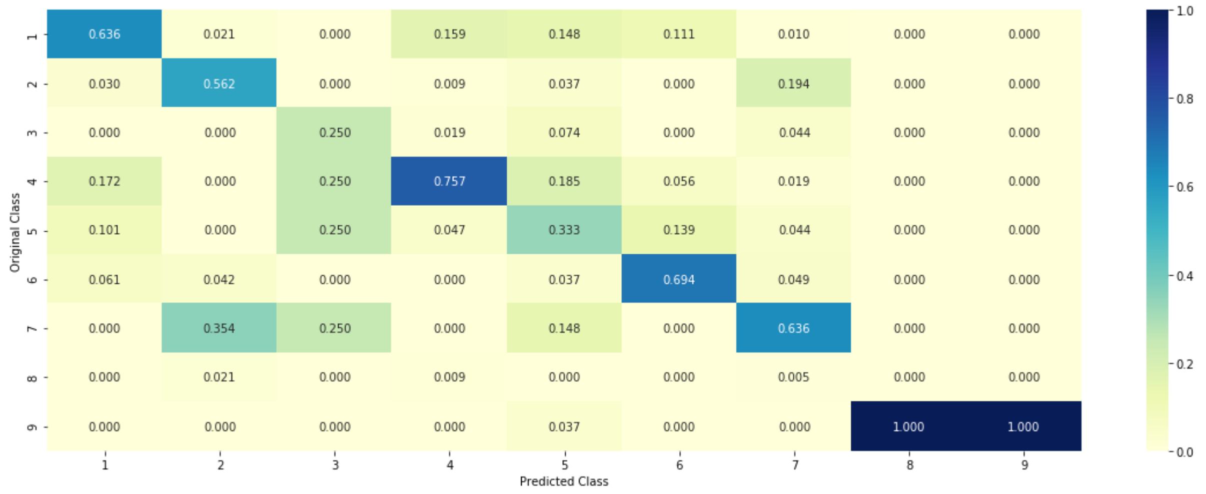
4.4.2. Testing model with best hyper parameters

In [130]:
```
# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight
='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

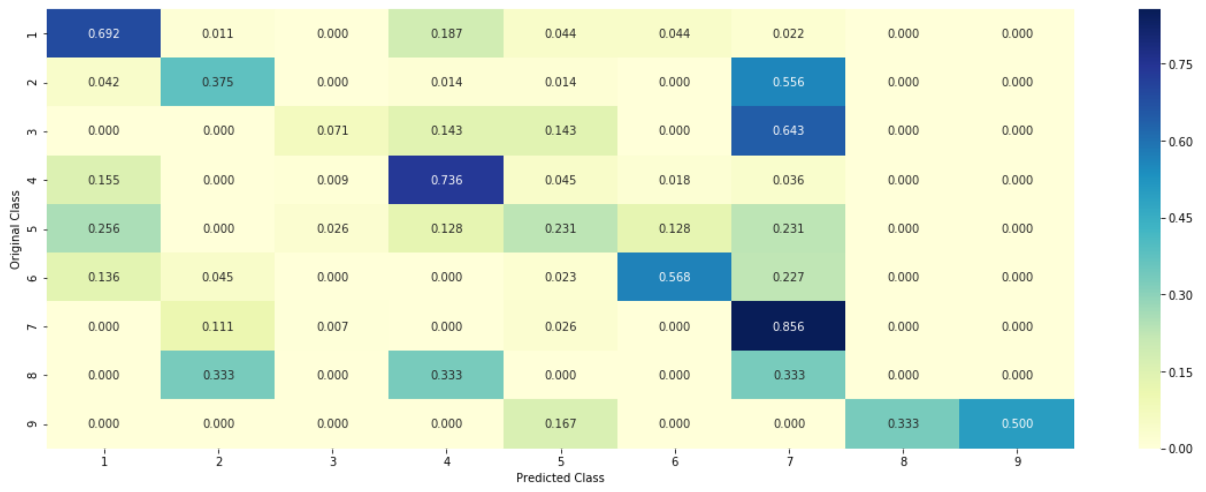
```
Log loss : 1.05659891897
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [133]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 50
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[ 0.062   0.0455  0.0417  0.0978  0.6907  0.0
26   0.0323  0.0018  0.0022]]
Actual Class : 5
--------------------------------------------------
139 Text feature [control] present in test data point [True]
140 Text feature [pathogenic] present in test data point [True]
142 Text feature [drug] present in test data point [True]
144 Text feature [characterization] present in test data point [True]
146 Text feature [effects] present in test data point [True]
147 Text feature [mrna] present in test data point [True]
149 Text feature [vus] present in test data point [True]
152 Text feature [occurrence] present in test data point [True]
189 Text feature [clones] present in test data point [True]
193 Text feature [differentiation] present in test data point [True]
197 Text feature [carrying] present in test data point [True]
200 Text feature [allow] present in test data point [True]
202 Text feature [pathology] present in test data point [True]
203 Text feature [breast] present in test data point [True]
205 Text feature [relevant] present in test data point [True]
208 Text feature [vitro] present in test data point [True]
210 Text feature [observed] present in test data point [True]
211 Text feature [fig] present in test data point [True]
215 Text feature [copy] present in test data point [True]
217 Text feature [still] present in test data point [True]
219 Text feature [s1] present in test data point [True]
223 Text feature [representation] present in test data point [True]
225 Text feature [alleles] present in test data point [True]
227 Text feature [contains] present in test data point [True]
228 Text feature [predictive] present in test data point [True]
230 Text feature [rmce] present in test data point [True]
234 Text feature [s3] present in test data point [True]
235 Text feature [classifi] present in test data point [True]
236 Text feature [aberrations] present in test data point [True]
237 Text feature [assays] present in test data point [True]
238 Text feature [alterations] present in test data point [True]
243 Text feature [need] present in test data point [True]
245 Text feature [showed] present in test data point [True]
251 Text feature [evaluated] present in test data point [True]
252 Text feature [fact] present in test data point [True]
256 Text feature [100] present in test data point [True]
259 Text feature [cation] present in test data point [True]
263 Text feature [affected] present in test data point [True]
264 Text feature [brca2] present in test data point [True]
267 Text feature [f3] present in test data point [True]
269 Text feature [author] present in test data point [True]
271 Text feature [s2] present in test data point [True]
274 Text feature [conservation] present in test data point [True]
279 Text feature [silico] present in test data point [True]
280 Text feature [60] present in test data point [True]
288 Text feature [bac] present in test data point [True]
292 Text feature [s4] present in test data point [True]
293 Text feature [pa] present in test data point [True]
296 Text feature [variant] present in test data point [True]
301 Text feature [involved] present in test data point [True]
302 Text feature [44] present in test data point [True]
303 Text feature [assessed] present in test data point [True]
```

```
309 Text feature [nih] present in test data point [True]
311 Text feature [effect] present in test data point [True]
312 Text feature [stem] present in test data point [True]
313 Text feature [early] present in test data point [True]
316 Text feature [unique] present in test data point [True]
317 Text feature [measure] present in test data point [True]
319 Text feature [repeat] present in test data point [True]
321 Text feature [data] present in test data point [True]
324 Text feature [even] present in test data point [True]
327 Text feature [www] present in test data point [True]
329 Text feature [blot] present in test data point [True]
331 Text feature [according] present in test data point [True]
333 Text feature [cd] present in test data point [True]
334 Text feature [38] present in test data point [True]
343 Text feature [support] present in test data point [True]
344 Text feature [express] present in test data point [True]
348 Text feature [investigate] present in test data point [True]
349 Text feature [observation] present in test data point [True]
350 Text feature [function] present in test data point [True]
354 Text feature [evaluation] present in test data point [True]
355 Text feature [could] present in test data point [True]
357 Text feature [number] present in test data point [True]
361 Text feature [around] present in test data point [True]
364 Text feature [contain] present in test data point [True]
365 Text feature [note] present in test data point [True]
368 Text feature [evaluate] present in test data point [True]
370 Text feature [factors] present in test data point [True]
373 Text feature [system] present in test data point [True]
376 Text feature [nevertheless] present in test data point [True]
379 Text feature [derived] present in test data point [True]
380 Text feature [allows] present in test data point [True]
387 Text feature [context] present in test data point [True]
388 Text feature [cross] present in test data point [True]
393 Text feature [assay] present in test data point [True]
394 Text feature [suggesting] present in test data point [True]
395 Text feature [however] present in test data point [True]
401 Text feature [triplicate] present in test data point [True]
406 Text feature [null] present in test data point [True]
410 Text feature [designed] present in test data point [True]
417 Text feature [reported] present in test data point [True]
421 Text feature [deletion] present in test data point [True]
428 Text feature [unable] present in test data point [True]
432 Text feature [consequences] present in test data point [True]
433 Text feature [sequence] present in test data point [True]
434 Text feature [expressing] present in test data point [True]
435 Text feature [scored] present in test data point [True]
437 Text feature [single] present in test data point [True]
438 Text feature [including] present in test data point [True]
440 Text feature [plasmid] present in test data point [True]
444 Text feature [terminal] present in test data point [True]
447 Text feature [variants] present in test data point [True]
453 Text feature [34] present in test data point [True]
455 Text feature [despite] present in test data point [True]
456 Text feature [shown] present in test data point [True]
460 Text feature [55] present in test data point [True]
463 Text feature [repair] present in test data point [True]
470 Text feature [far] present in test data point [True]
```

```
473 Text feature [functional] present in test data point [True]
479 Text feature [cdna] present in test data point [True]
483 Text feature [transactivation] present in test data point [True]
486 Text feature [42] present in test data point [True]
492 Text feature [may] present in test data point [True]
496 Text feature [wild] present in test data point [True]
497 Text feature [construct] present in test data point [True]
Out of the top  500  features  116 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [134]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0529  0.0898  0.0109  0.4105  0.0531  0.0
338  0.34    0.0083  0.0007]]
Actual Class : 7
--------------------------------------------------
172 Text feature [little] present in test data point [True]
175 Text feature [recombinant] present in test data point [True]
176 Text feature [contribute] present in test data point [True]
184 Text feature [plated] present in test data point [True]
185 Text feature [del] present in test data point [True]
191 Text feature [hd] present in test data point [True]
193 Text feature [strategies] present in test data point [True]
198 Text feature [none] present in test data point [True]
199 Text feature [absent] present in test data point [True]
202 Text feature [observed] present in test data point [True]
203 Text feature [suggesting] present in test data point [True]
204 Text feature [lacking] present in test data point [True]
210 Text feature [tyrosine] present in test data point [True]
212 Text feature [unclear] present in test data point [True]
222 Text feature [western] present in test data point [True]
236 Text feature [suggest] present in test data point [True]
240 Text feature [lobe] present in test data point [True]
243 Text feature [4a] present in test data point [True]
244 Text feature [one] present in test data point [True]
249 Text feature [present] present in test data point [True]
252 Text feature [majority] present in test data point [True]
254 Text feature [strategy] present in test data point [True]
255 Text feature [cytokine] present in test data point [True]
257 Text feature [effective] present in test data point [True]
464 Text feature [show] present in test data point [True]
477 Text feature [motif] present in test data point [True]
479 Text feature [functional] present in test data point [True]
482 Text feature [plasma] present in test data point [True]
491 Text feature [either] present in test data point [True]
495 Text feature [cannot] present in test data point [True]
497 Text feature [except] present in test data point [True]
Out of the top  500  features  31 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [135]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cla
sses_, eps=1e-15))
```

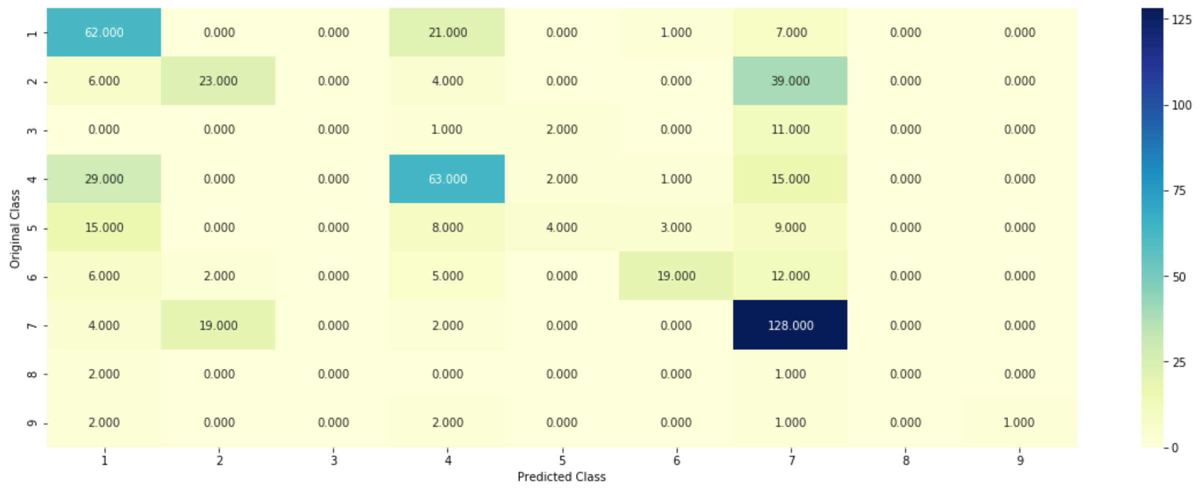
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''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],c
v_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_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.21859175956
for n_estimators = 100 and max depth =  10
Log Loss : 1.24464431871
for n_estimators = 200 and max depth =  5
Log Loss : 1.21651100694
for n_estimators = 200 and max depth =  10
Log Loss : 1.22932652314
for n_estimators = 500 and max depth =  5
Log Loss : 1.20588453695
for n_estimators = 500 and max depth =  10
Log Loss : 1.22393698272
for n_estimators = 1000 and max depth =  5
Log Loss : 1.20324843195
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2208543448
for n_estimators = 2000 and max depth =  5
Log Loss : 1.20321267377
for n_estimators = 2000 and max depth =  10
Log Loss : 1.22085424749
For values of best estimator =  2000 The train log loss is: 0.848245518291
For values of best estimator =  2000 The cross validation log loss is: 1.2032
1213927
For values of best estimator =  2000 The test log loss is: 1.21560766426
```

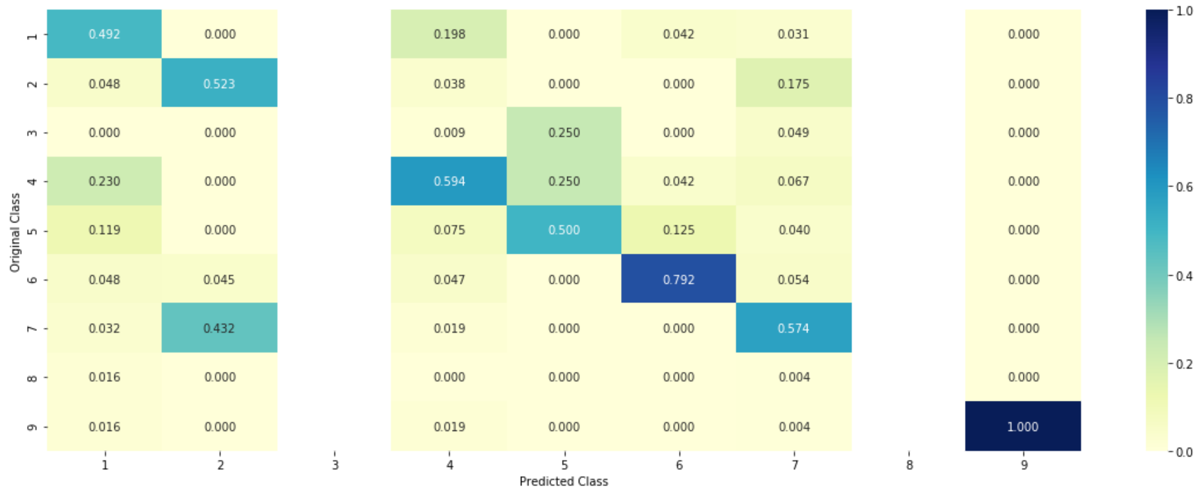
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [136]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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

clf = RandomForestClassifier(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_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

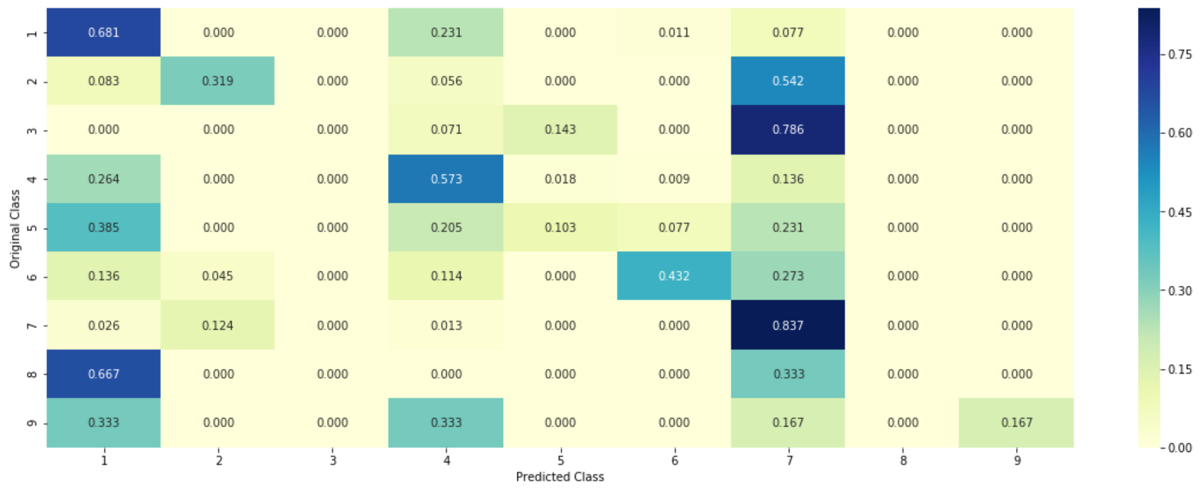
```
Log loss : 1.20321034972
Number of mis-classified points : 0.43609022556390975
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [138]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion=
'gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1397  0.1618  0.0237  0.0949  0.0528  0.0
502  0.4486  0.0161  0.0123]]
Actual Class : 7
--------------------------------------------------
2 Text feature [inhibitor] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
8 Text feature [inhibition] present in test data point [True]
9 Text feature [fully] present in test data point [True]
10 Text feature [oncogenes] present in test data point [True]
11 Text feature [none] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
14 Text feature [consistent] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
19 Text feature [function] present in test data point [True]
20 Text feature [constitutive] present in test data point [True]
21 Text feature [provides] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
25 Text feature [cells] present in test data point [True]
27 Text feature [grown] present in test data point [True]
29 Text feature [transforming] present in test data point [True]
31 Text feature [recently] present in test data point [True]
33 Text feature [activate] present in test data point [True]
35 Text feature [expressing] present in test data point [True]
36 Text feature [kinase] present in test data point [True]
40 Text feature [treated] present in test data point [True]
41 Text feature [therapeutic] present in test data point [True]
46 Text feature [monoclonal] present in test data point [True]
48 Text feature [advanced] present in test data point [True]
49 Text feature [stability] present in test data point [True]
51 Text feature [extent] present in test data point [True]
54 Text feature [occurs] present in test data point [True]
58 Text feature [prostate] present in test data point [True]
60 Text feature [effects] present in test data point [True]
61 Text feature [like] present in test data point [True]
62 Text feature [retained] present in test data point [True]
63 Text feature [patient] present in test data point [True]
64 Text feature [57] present in test data point [True]
66 Text feature [progressive] present in test data point [True]
71 Text feature [resistance] present in test data point [True]
72 Text feature [cell] present in test data point [True]
73 Text feature [membrane] present in test data point [True]
74 Text feature [clinical] present in test data point [True]
76 Text feature [predict] present in test data point [True]
81 Text feature [old] present in test data point [True]
82 Text feature [expressed] present in test data point [True]
84 Text feature [influence] present in test data point [True]
87 Text feature [response] present in test data point [True]
89 Text feature [far] present in test data point [True]
90 Text feature [survival] present in test data point [True]
92 Text feature [image] present in test data point [True]
93 Text feature [done] present in test data point [True]
94 Text feature [phosphate] present in test data point [True]
97 Text feature [map] present in test data point [True]
Out of the top  100  features  50 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [139]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1421  0.1904  0.0283  0.1787  0.0628  0.0
527  0.3275  0.0088  0.0088]]
Actuall Class : 2
--------------------------------------------------
0 Text feature [kidney] present in test data point [True]
1 Text feature [activating] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [suppressor] present in test data point [True]
11 Text feature [none] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [minutes] present in test data point [True]
14 Text feature [consistent] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
19 Text feature [function] present in test data point [True]
25 Text feature [cells] present in test data point [True]
27 Text feature [grown] present in test data point [True]
29 Text feature [transforming] present in test data point [True]
31 Text feature [recently] present in test data point [True]
33 Text feature [activate] present in test data point [True]
35 Text feature [expressing] present in test data point [True]
36 Text feature [kinase] present in test data point [True]
40 Text feature [treated] present in test data point [True]
43 Text feature [akt] present in test data point [True]
58 Text feature [prostate] present in test data point [True]
60 Text feature [effects] present in test data point [True]
61 Text feature [like] present in test data point [True]
63 Text feature [patient] present in test data point [True]
68 Text feature [functionally] present in test data point [True]
70 Text feature [nsclc] present in test data point [True]
72 Text feature [cell] present in test data point [True]
76 Text feature [predict] present in test data point [True]
78 Text feature [outside] present in test data point [True]
79 Text feature [transformation] present in test data point [True]
82 Text feature [expressed] present in test data point [True]
87 Text feature [response] present in test data point [True]
90 Text feature [survival] present in test data point [True]
94 Text feature [phosphate] present in test data point [True]
98 Text feature [importantly] present in test data point [True]
Out of the top  100  features  36 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [140]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, eps=1e-15))
```

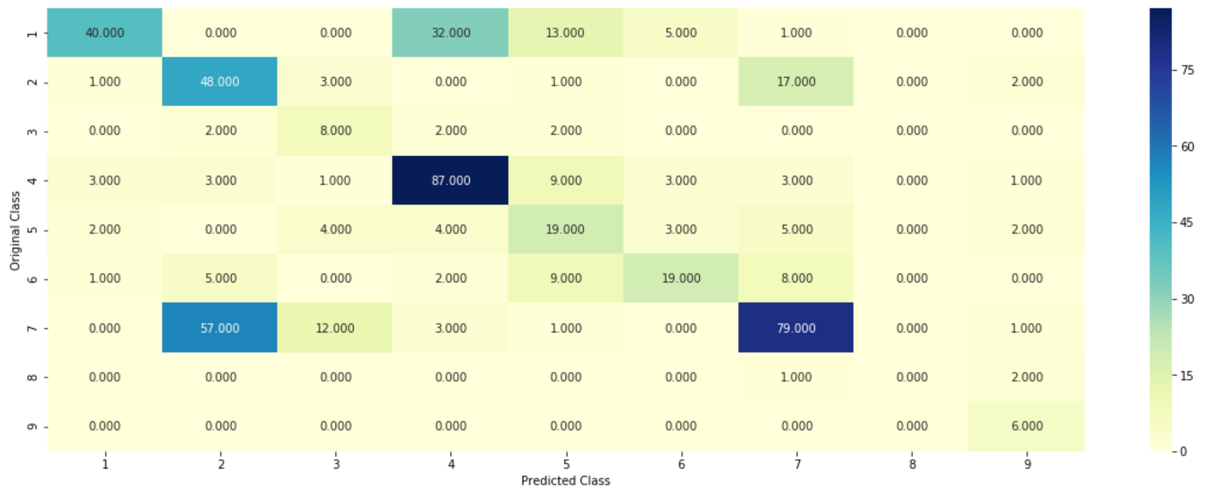
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],c
v_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 vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.09247743287
for n_estimators = 10 and max depth =  3
Log Loss : 1.63926487985
for n_estimators = 10 and max depth =  5
Log Loss : 1.32650726713
for n_estimators = 10 and max depth =  10
Log Loss : 2.21863156418
for n_estimators = 50 and max depth =  2
Log Loss : 1.73831714309
for n_estimators = 50 and max depth =  3
Log Loss : 1.4853287081
for n_estimators = 50 and max depth =  5
Log Loss : 1.34247068128
for n_estimators = 50 and max depth =  10
Log Loss : 1.79349067531
for n_estimators = 100 and max depth =  2
Log Loss : 1.56385224847
for n_estimators = 100 and max depth =  3
Log Loss : 1.52240231164
for n_estimators = 100 and max depth =  5
Log Loss : 1.28176514328
for n_estimators = 100 and max depth =  10
Log Loss : 1.74974739516
for n_estimators = 200 and max depth =  2
Log Loss : 1.59636985907
for n_estimators = 200 and max depth =  3
Log Loss : 1.50441233196
for n_estimators = 200 and max depth =  5
Log Loss : 1.34607106021
for n_estimators = 200 and max depth =  10
Log Loss : 1.71576815574
for n_estimators = 500 and max depth =  2
Log Loss : 1.64464486058
for n_estimators = 500 and max depth =  3
Log Loss : 1.55694476634
for n_estimators = 500 and max depth =  5
Log Loss : 1.4013577365
for n_estimators = 500 and max depth =  10
Log Loss : 1.76816155202
for n_estimators = 1000 and max depth =  2
Log Loss : 1.61798533792
for n_estimators = 1000 and max depth =  3
Log Loss : 1.54706123701
for n_estimators = 1000 and max depth =  5
Log Loss : 1.40239962753
for n_estimators = 1000 and max depth =  10
Log Loss : 1.73210973524
For values of best alpha =  100 The train log loss is: 0.0534800458838
For values of best alpha =  100 The cross validation log loss is: 1.281765143
28
For values of best alpha =  100 The test log loss is: 1.33408214454
```

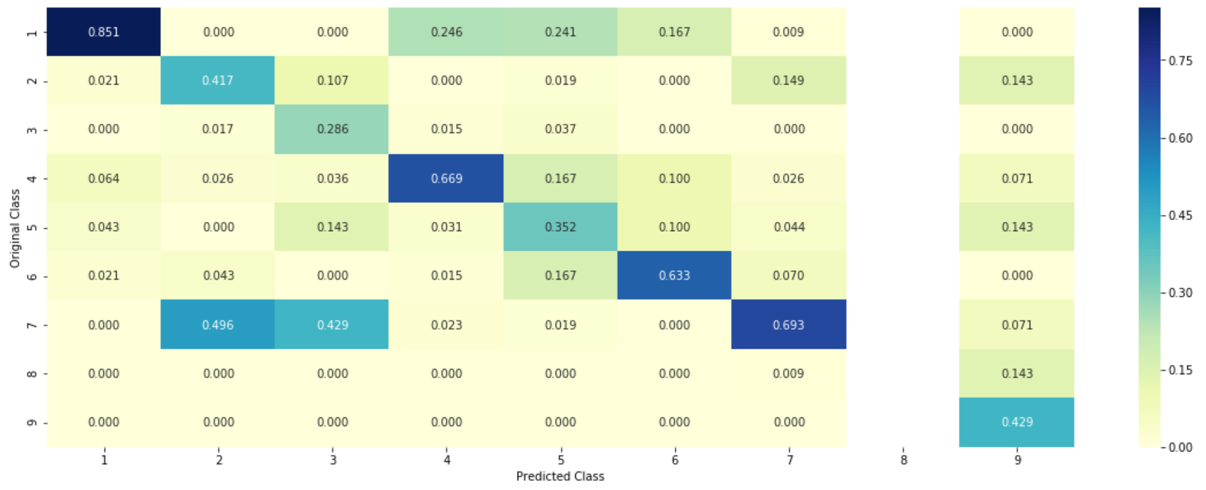
4.5.4. Testing model with best hyper parameters (Response Coding)

In [141]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimat
ors=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_sta
te=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_respons
eCoding,cv_y, clf)
```

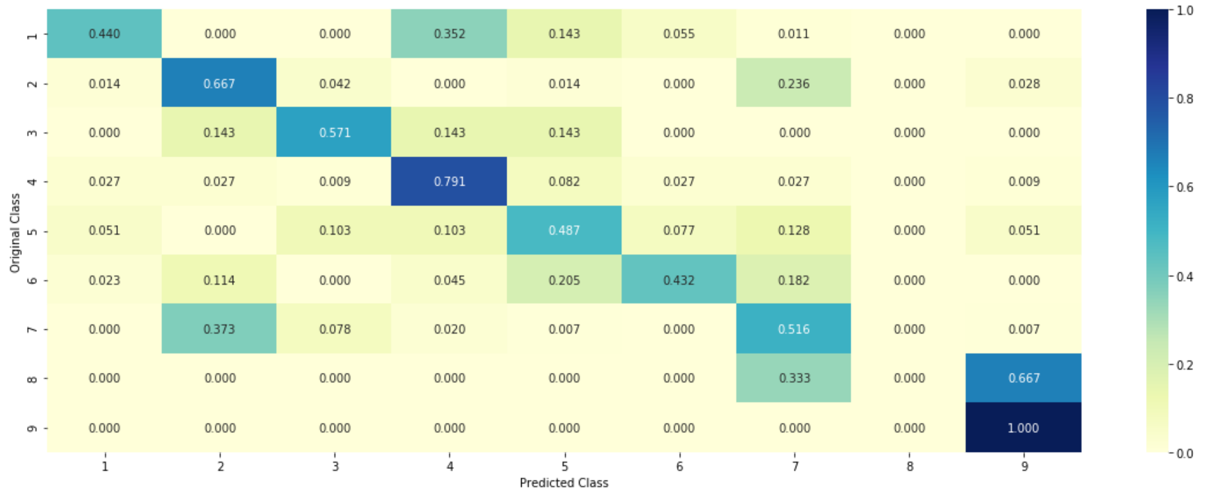
```
Log loss : 1.28176514328
Number of mis-classified points : 0.424812030075188
-------------------- Confusion matrix --------------------
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [145]:
```
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 = 10
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[  1.40000000e-03   5.61000000e-02   1.400000
00e-03   2.40000000e-03
    8.00000000e-04   5.90000000e-03   9.29000000e-01   1.50000000e-03
    1.40000000e-03]]
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
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [146]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0077  0.575   0.093   0.0188  0.0127  0.0
248  0.2207  0.0353  0.012 ]]
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
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text 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 [149]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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



# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.
html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
```

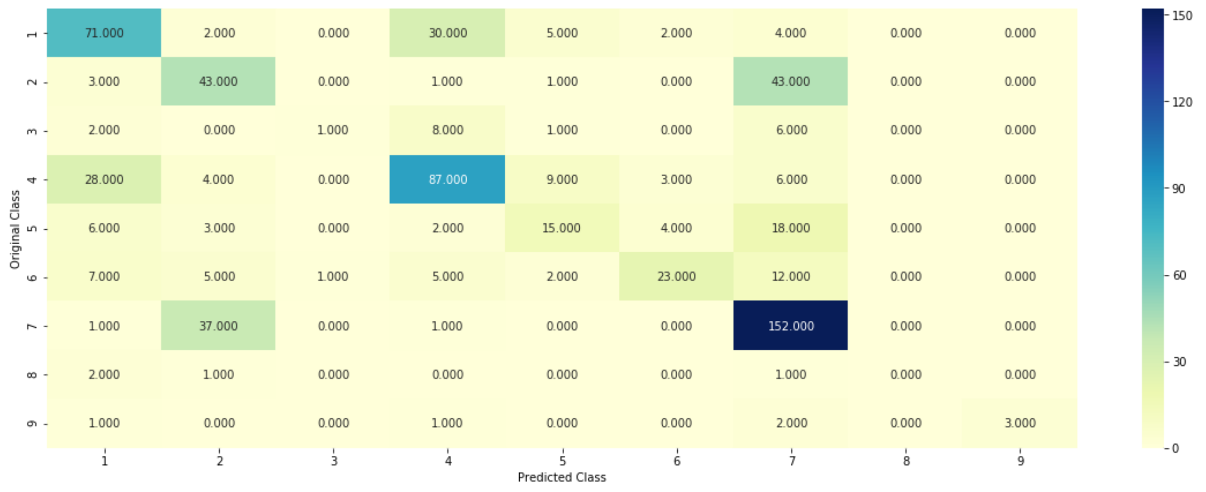
```
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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



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

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


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

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

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.88
Naive Bayes : Log Loss: 1.25
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.503
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.195
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.429
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.905
```

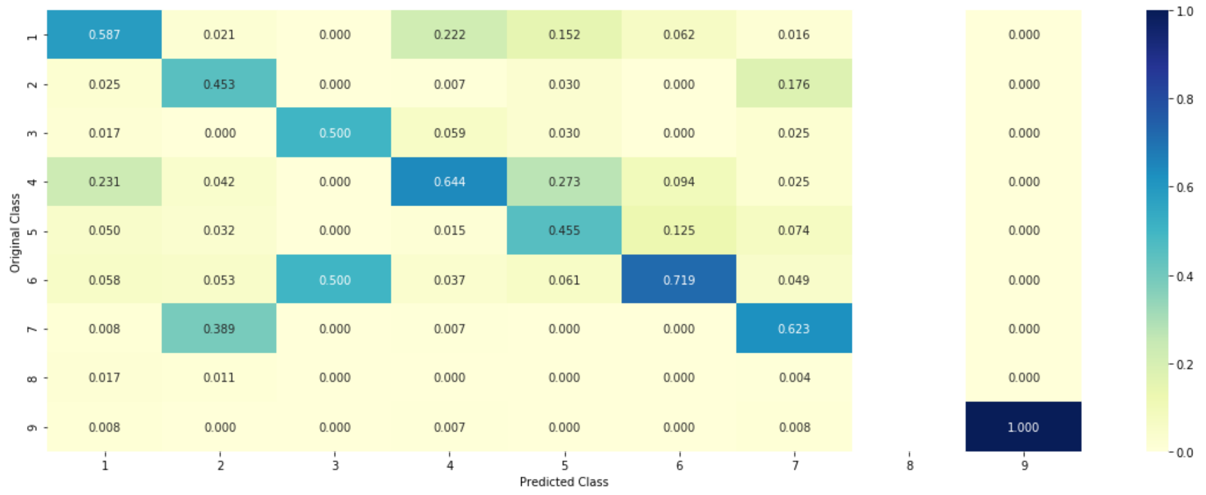
4.7.2 testing the model with the best hyper parameters

In [150]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_
x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCodin
g))
```

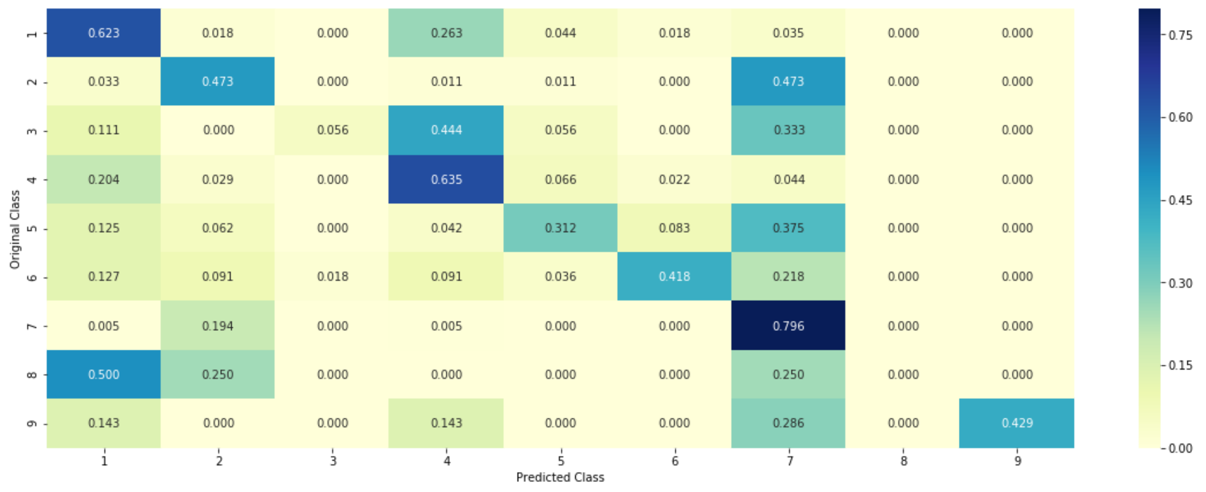
```
Log loss (train) on the stacking classifier : 0.569267606947
Log loss (CV) on the stacking classifier : 1.19522813528
Log loss (test) on the stacking classifier : 1.2412816008
Number of missclassified point : 0.40601503759398494
-------------------- Confusion matrix --------------------
```



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



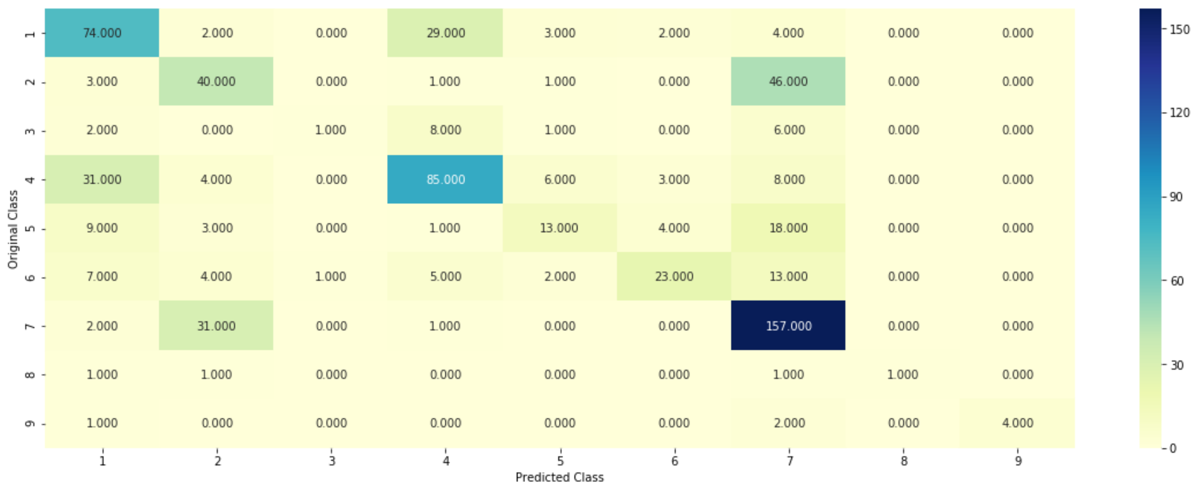
```
-------------------- Recall matrix (Row sum=1) --------------------
```



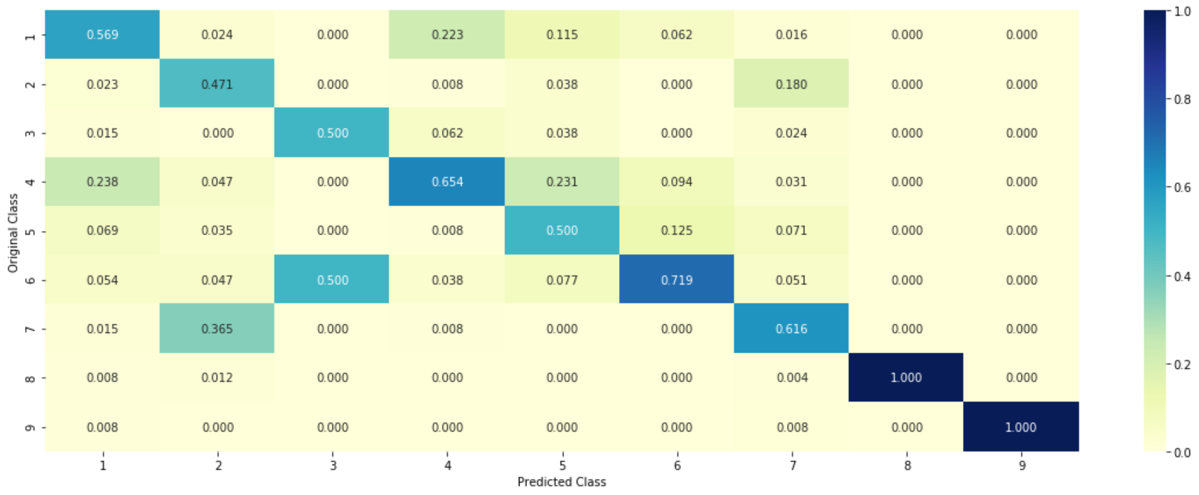
4.7.3 Maximum Voting classifier

In [151]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Votin
gClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf'
, sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.pre
dict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_p
roba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predi
ct_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_
x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCodin
g))
```

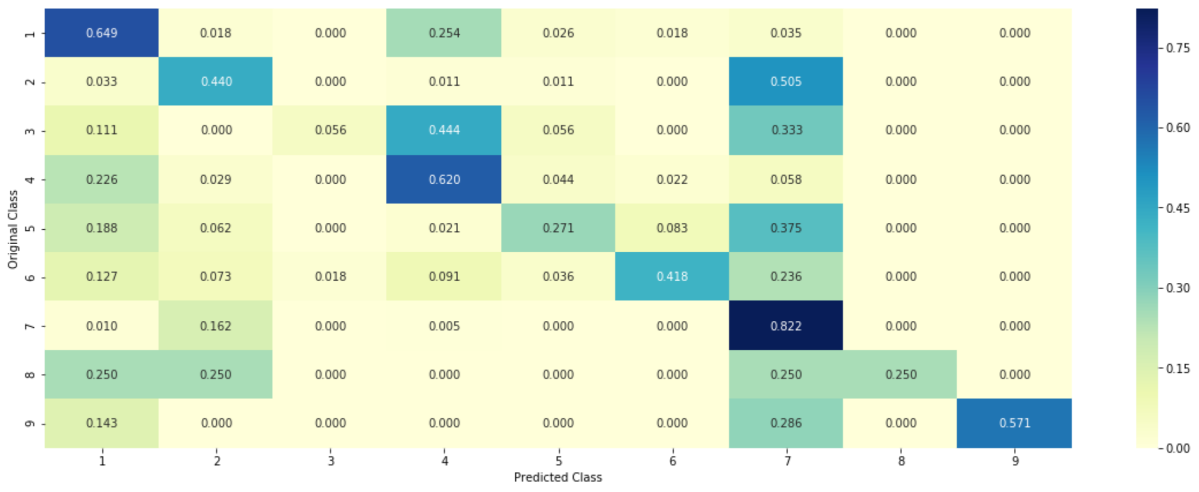
```
Log loss (train) on the VotingClassifier : 0.833815794932
Log loss (CV) on the VotingClassifier : 1.20990834871
Log loss (test) on the VotingClassifier : 1.24414041272
Number of missclassified point : 0.40150375939849625
-------------------- Confusion matrix --------------------
```



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



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



4.8 GBDT Classifier

4.8.1. With One hot Encoding

4.8.1.1. hyper parameter tuning

In [152]:
```
import xgboost as xgb
alpha = [100,200,500,1000]
max_depth = [3, 5]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = xgb.XGBClassifier(n_estimators=i, max_depth=j, random_state=42,
n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cla
sses_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
        print('-'*30)
```

```
for n_estimators = 100 and max depth =  3
Log Loss : 1.17751193357
------------------------------
for n_estimators = 100 and max depth =  5
Log Loss : 1.18027802804
------------------------------
for n_estimators = 200 and max depth =  3
Log Loss : 1.19009230388
------------------------------
for n_estimators = 200 and max depth =  5
Log Loss : 1.17777940669
------------------------------
for n_estimators = 500 and max depth =  3
Log Loss : 1.19270654639
------------------------------
for n_estimators = 500 and max depth =  5
Log Loss : 1.16815843073
------------------------------
for n_estimators = 1000 and max depth =  3
Log Loss : 1.19231925286
------------------------------
for n_estimators = 1000 and max depth =  5
Log Loss : 1.16936419207
------------------------------
```

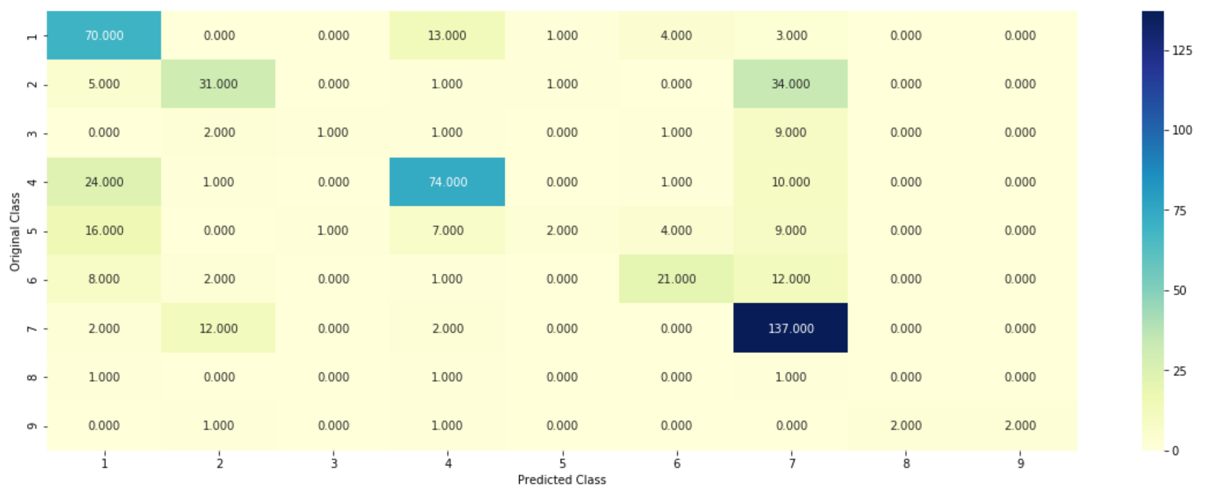
In [153]:
```
best_alpha = np.argmin(cv_log_error_array)
clf = xgb.XGBClassifier(n_estimators=alpha[int(best_alpha/2)], max_depth=max_d
epth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
For values of best estimator =  500 The train log loss is: 0.546358339995
For values of best estimator =  500 The cross validation log loss is: 1.16815
843073
For values of best estimator =  500 The test log loss is: 1.18960470622
```

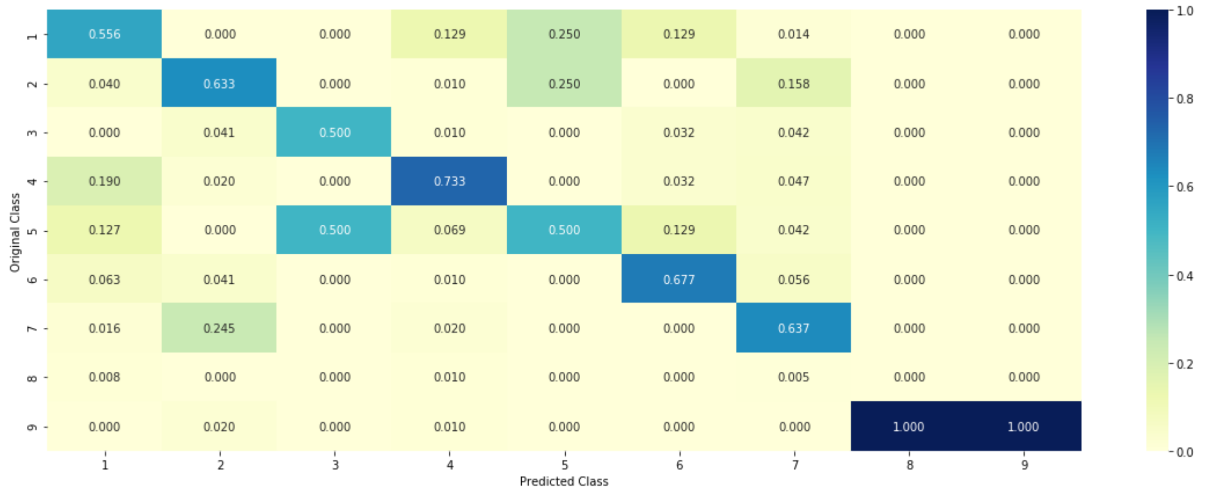
4.8.1.1. Testing model with best hyperparameter

In [154]:
```
clf = xgb.XGBClassifier(n_estimators=alpha[int(best_alpha/2)], max_depth=max_d
epth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

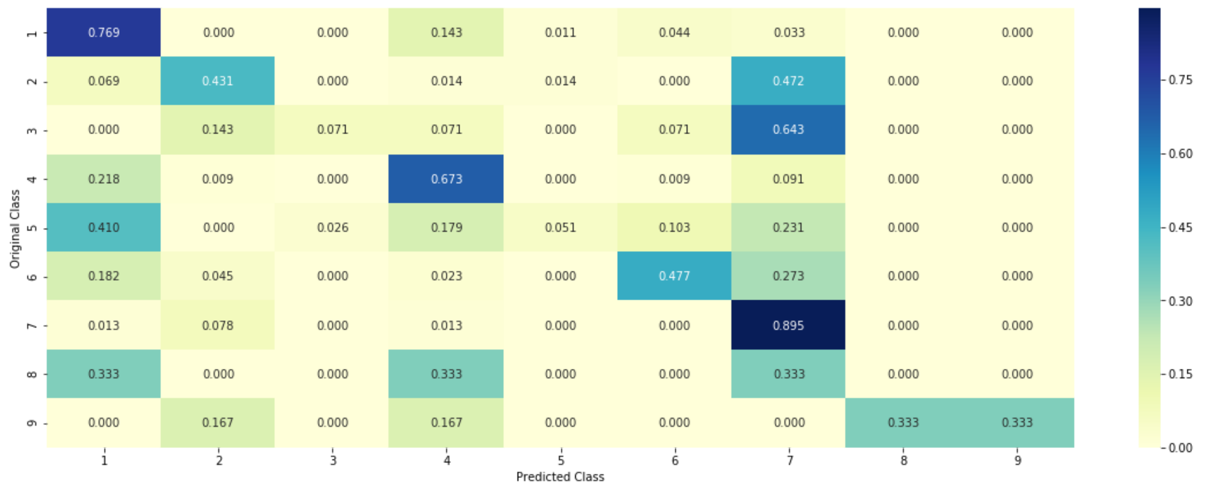
```
Log loss : 1.16815843073
Number of mis-classified points : 0.364661654135338355
-------------------- Confusion matrix --------------------
```



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



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



4.8.1.3. Feature Importance, Correctly classified point

In [155]:
```
# test_point_index = 10
clf = xgb.XGBClassifier(n_estimators=alpha[int(best_alpha/2)], max_depth=max_d
epth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0896  0.0948  0.0178  0.07    0.0435  0.0
343  0.6378  0.0064  0.0058]]
Actual Class : 7
--------------------------------------------------
5 Text feature [gel] present in test data point [True]
8 Text feature [nucleus] present in test data point [True]
13 Text feature [cells] present in test data point [True]
14 Text feature [sds] present in test data point [True]
16 Text feature [suggesting] present in test data point [True]
17 Text feature [prostate] present in test data point [True]
18 Text feature [21] present in test data point [True]
19 Text feature [residue] present in test data point [True]
24 Text feature [response] present in test data point [True]
25 Text feature [use] present in test data point [True]
26 Text feature [member] present in test data point [True]
28 Text feature [third] present in test data point [True]
29 Text feature [assays] present in test data point [True]
30 Text feature [studied] present in test data point [True]
31 Text feature [within] present in test data point [True]
32 Text feature [analysis] present in test data point [True]
34 Text feature [function] present in test data point [True]
35 Text feature [grown] present in test data point [True]
36 Text feature [amino] present in test data point [True]
38 Text feature [activated] present in test data point [True]
39 Text feature [cell] present in test data point [True]
40 Text feature [using] present in test data point [True]
42 Text feature [high] present in test data point [True]
43 Text feature [whereas] present in test data point [True]
44 Text feature [thus] present in test data point [True]
45 Text feature [along] present in test data point [True]
47 Text feature [whether] present in test data point [True]
48 Text feature [suggest] present in test data point [True]
49 Text feature [10] present in test data point [True]
50 Text feature [31] present in test data point [True]
51 Text feature [derived] present in test data point [True]
52 Text feature [activation] present in test data point [True]
54 Text feature [cases] present in test data point [True]
55 Text feature [expressing] present in test data point [True]
57 Text feature [normalized] present in test data point [True]
59 Text feature [suggested] present in test data point [True]
60 Text feature [pathways] present in test data point [True]
61 Text feature [11] present in test data point [True]
62 Text feature [oncogenes] present in test data point [True]
63 Text feature [completely] present in test data point [True]
64 Text feature [500] present in test data point [True]
66 Text feature [23] present in test data point [True]
67 Text feature [al] present in test data point [True]
68 Text feature [regulates] present in test data point [True]
70 Text feature [anti] present in test data point [True]
71 Text feature [marker] present in test data point [True]
72 Text feature [specific] present in test data point [True]
73 Text feature [100] present in test data point [True]
74 Text feature [leads] present in test data point [True]
75 Text feature [materials] present in test data point [True]
77 Text feature [amplification] present in test data point [True]
78 Text feature [involved] present in test data point [True]
```

```
82 Text feature [test] present in test data point [True]
83 Text feature [clinical] present in test data point [True]
84 Text feature [available] present in test data point [True]
86 Text feature [influence] present in test data point [True]
89 Text feature [subsequent] present in test data point [True]
91 Text feature [000] present in test data point [True]
92 Text feature [currently] present in test data point [True]
94 Text feature [encoding] present in test data point [True]
95 Text feature [role] present in test data point [True]
96 Text feature [edta] present in test data point [True]
97 Text feature [patients] present in test data point [True]
98 Text feature [six] present in test data point [True]
99 Text feature [measured] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

4.8.1.4. Feature Importanace, incorrectly classified point

In [157]:
```
test_point_index = 15
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[ 0.3774  0.0893  0.0197  0.2708  0.0494  0.0
337  0.1465  0.0069  0.0064]]
Actual Class : 4
--------------------------------------------------
4 Text feature [activating] present in test data point [True]
13 Text feature [cells] present in test data point [True]
14 Text feature [sds] present in test data point [True]
16 Text feature [suggesting] present in test data point [True]
18 Text feature [21] present in test data point [True]
29 Text feature [assays] present in test data point [True]
30 Text feature [studied] present in test data point [True]
31 Text feature [within] present in test data point [True]
32 Text feature [analysis] present in test data point [True]
34 Text feature [function] present in test data point [True]
35 Text feature [grown] present in test data point [True]
36 Text feature [amino] present in test data point [True]
39 Text feature [cell] present in test data point [True]
40 Text feature [using] present in test data point [True]
41 Text feature [phosphorylated] present in test data point [True]
43 Text feature [whereas] present in test data point [True]
46 Text feature [deletions] present in test data point [True]
47 Text feature [whether] present in test data point [True]
49 Text feature [10] present in test data point [True]
50 Text feature [31] present in test data point [True]
51 Text feature [derived] present in test data point [True]
52 Text feature [activation] present in test data point [True]
54 Text feature [cases] present in test data point [True]
55 Text feature [expressing] present in test data point [True]
61 Text feature [11] present in test data point [True]
63 Text feature [completely] present in test data point [True]
66 Text feature [23] present in test data point [True]
67 Text feature [al] present in test data point [True]
70 Text feature [anti] present in test data point [True]
72 Text feature [specific] present in test data point [True]
75 Text feature [materials] present in test data point [True]
79 Text feature [targets] present in test data point [True]
82 Text feature [test] present in test data point [True]
83 Text feature [clinical] present in test data point [True]
84 Text feature [available] present in test data point [True]
88 Text feature [hours] present in test data point [True]
91 Text feature [000] present in test data point [True]
94 Text feature [encoding] present in test data point [True]
95 Text feature [role] present in test data point [True]
98 Text feature [six] present in test data point [True]
Out of the top  100  features  40 are present in query point
```

4.8.2. With Response Coding

4.8.2.1. hyper parameter tuning

In [158]:
```
alpha = [10,50,100,200,500,1000]
max_depth = [3,5]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = xgb.XGBClassifier(n_estimators=i, 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.cla
sses_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
```

```
for n_estimators = 10 and max depth =  3
Log Loss : 1.92432409868
for n_estimators = 10 and max depth =  5
Log Loss : 1.92517287563
for n_estimators = 50 and max depth =  3
Log Loss : 1.85518702117
for n_estimators = 50 and max depth =  5
Log Loss : 1.86230793551
for n_estimators = 100 and max depth =  3
Log Loss : 1.55820521488
for n_estimators = 100 and max depth =  5
Log Loss : 1.61450846945
for n_estimators = 200 and max depth =  3
Log Loss : 1.4980030725
for n_estimators = 200 and max depth =  5
Log Loss : 1.53987032707
for n_estimators = 500 and max depth =  3
Log Loss : 1.48142838672
for n_estimators = 500 and max depth =  5
Log Loss : 1.51777098675
for n_estimators = 1000 and max depth =  3
Log Loss : 1.46837366778
for n_estimators = 1000 and max depth =  5
Log Loss : 1.51579066208
```

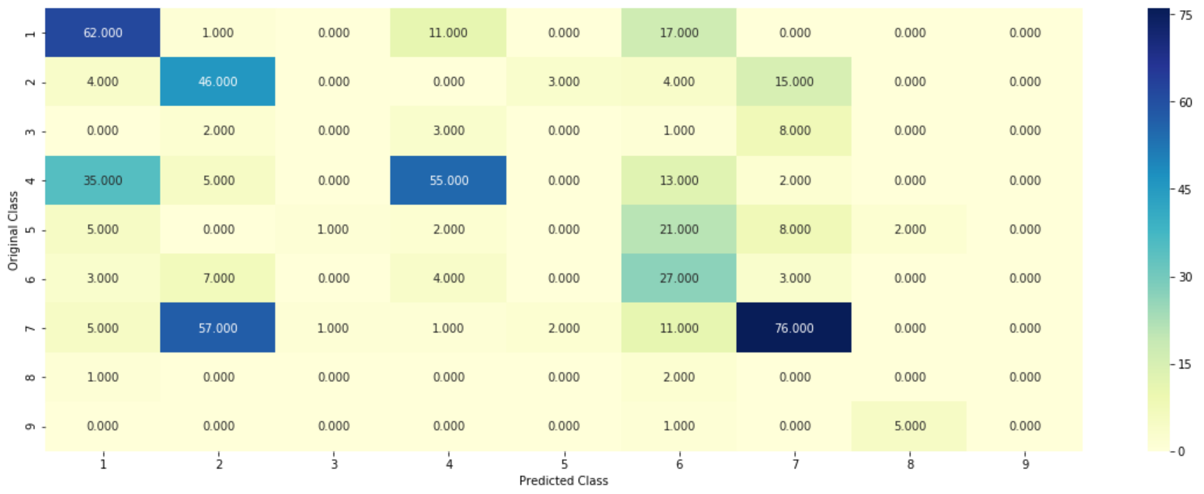
In [159]:
```
best_alpha = np.argmin(cv_log_error_array)
clf = xgb.XGBClassifier(n_estimators=alpha[int(best_alpha/2)], max_depth=max_d
epth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, 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 vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of best alpha =  100 The train log loss is: 0.0239723308605
For values of best alpha =  100 The cross validation log loss is: 1.468373667
78
For values of best alpha =  100 The test log loss is: 1.48467578073
```

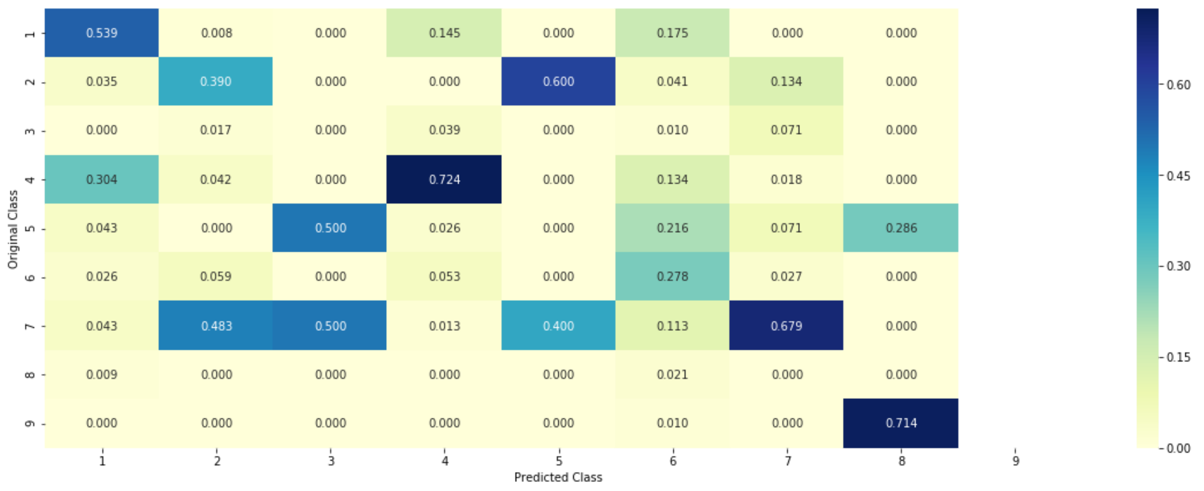
4.8.2.2. Testing model using best hyperparameter

In [160]:
```
clf = xgb.XGBClassifier(n_estimators=alpha[int(best_alpha/2)], max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

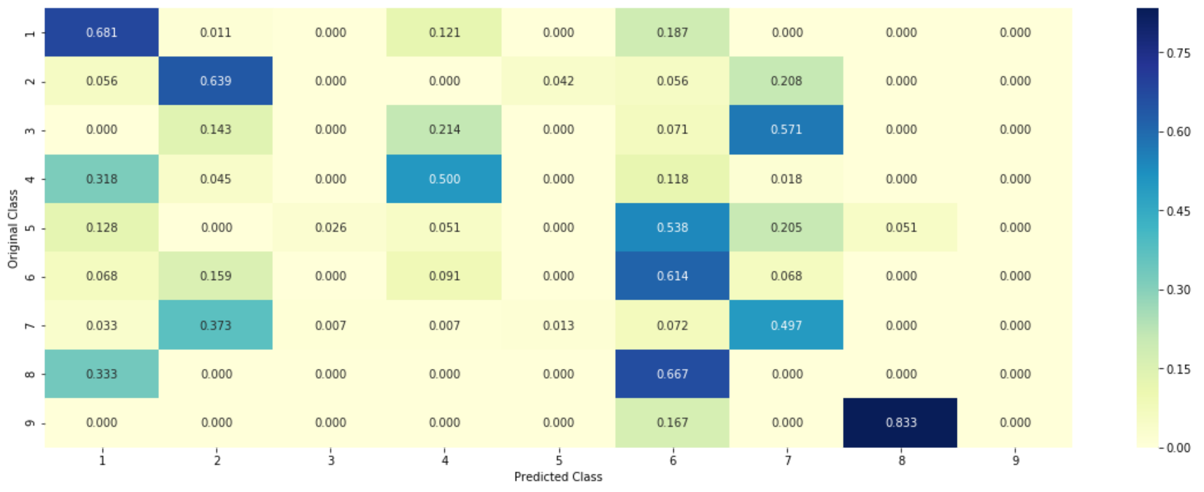
```
Log loss : 1.46837366778
Number of mis-classified points : 0.5
-------------------- Confusion matrix --------------------
```



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



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



Observation -

model is overfitting

4.9. Logistic Regression using feature engineered tfidf feature

Feature engineering on one hot encoded features

In [161]:
```
train_x_onehotCoding_FE=np.sqrt(train_x_onehotCoding)
test_x_onehotCoding_FE=np.sqrt(test_x_onehotCoding)
cv_x_onehotCoding_FE=np.sqrt(cv_x_onehotCoding)
```

In [162]:
```
print("One hot encoding features using feature engineering :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding_FE.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding_FE.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_onehotCoding_FE.shape)
```

```
One hot encoding features using feature engineering :
(number of data points * number of features) in train data =  (2124, 4200)
(number of data points * number of features) in test data =  (665, 4200)
(number of data points * number of features) in cross validation data = (532,
4200)
```

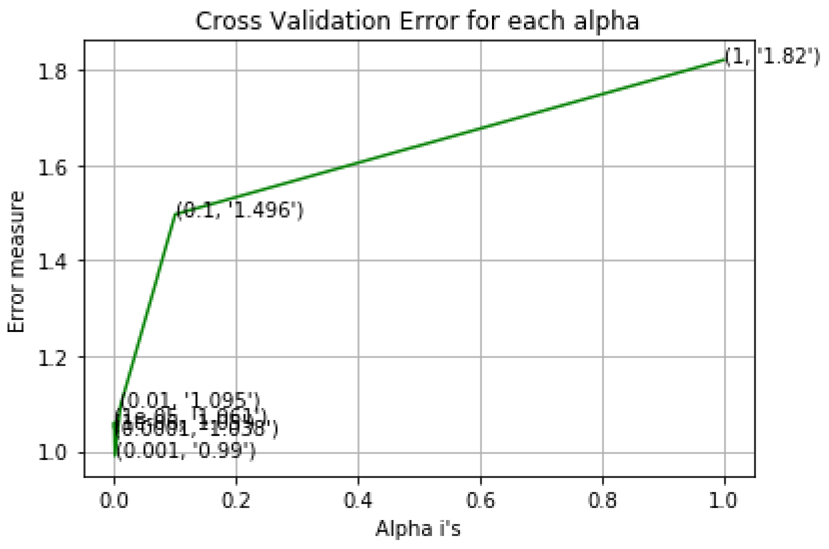
4.9.1. with class balancing

4.9.1.1. hyper parameter tuning

In [163]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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 = [0.005,0.05,0.5]
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, class_weight='balanced', penalty='l2', loss=
'log', random_state=42)
    clf.fit(train_x_onehotCoding_FE, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_FE, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_FE)
    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], class_weight='balanced', penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_FE, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_FE, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_FE)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_FE)
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_FE)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.05049698455
for alpha = 1e-05
Log Loss : 1.06078455344
for alpha = 0.0001
Log Loss : 1.03757613233
for alpha = 0.001
Log Loss : 0.990289328304
for alpha = 0.01
Log Loss : 1.09465118635
for alpha = 0.1
Log Loss : 1.49614912025
for alpha = 1
Log Loss : 1.82007290684
```

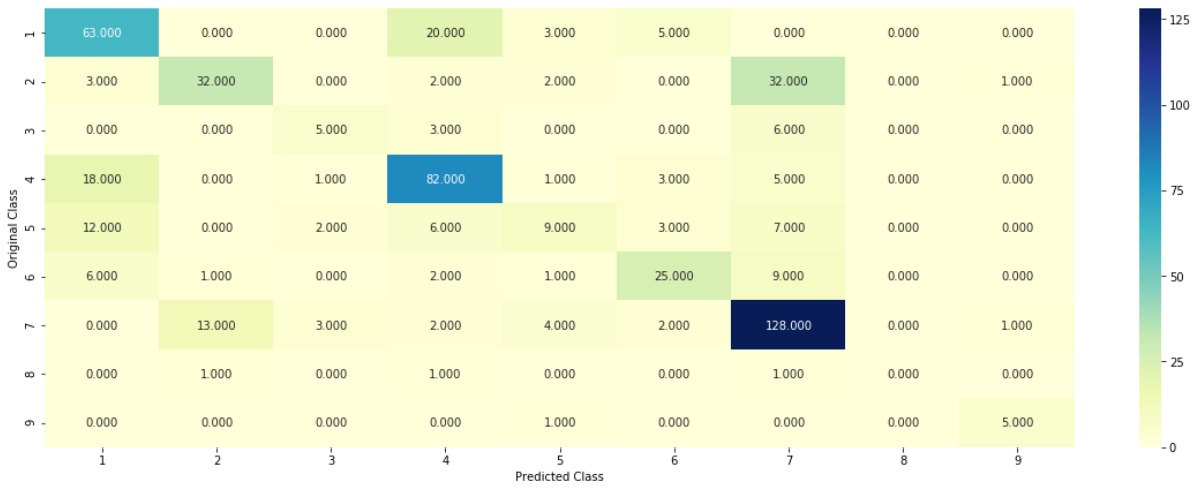


```
For values of best alpha =  0.001 The train log loss is: 0.580086893091
For values of best alpha =  0.001 The cross validation log loss is: 0.9902893
28304
For values of best alpha =  0.001 The test log loss is: 1.03455537011
```

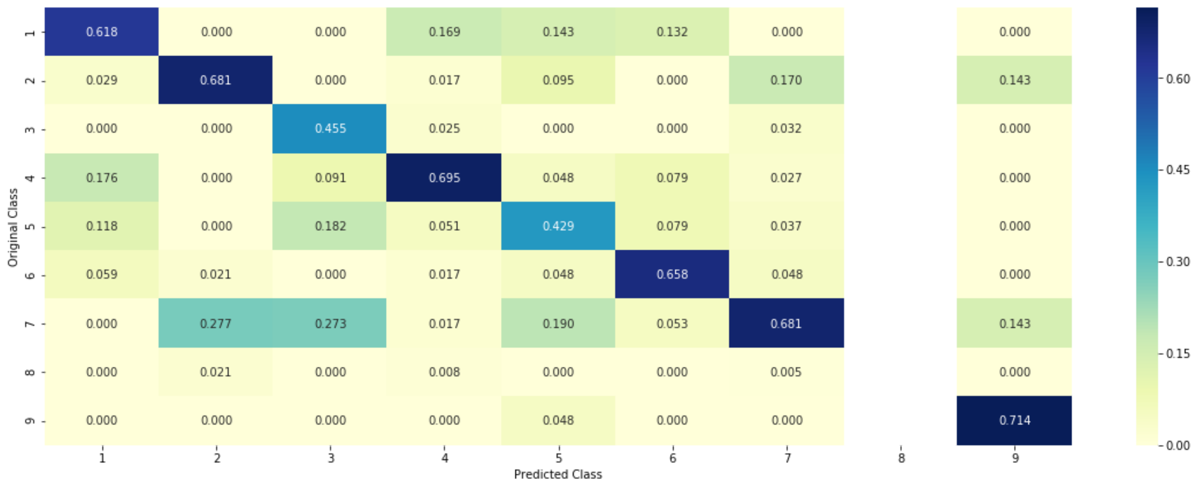
4.9.1.2. Testing the model using best hyperparameter

In [164]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

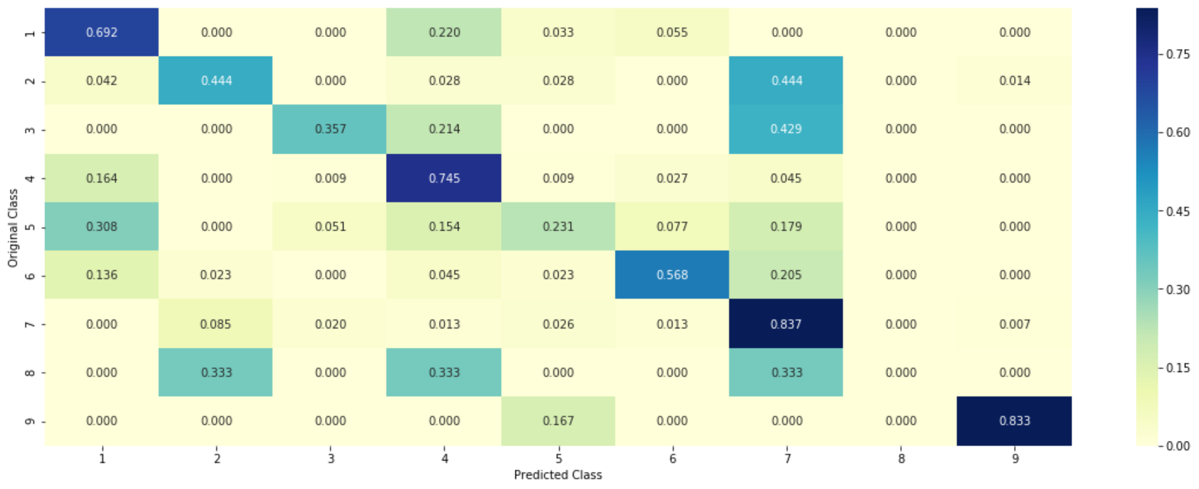
```
Log loss : 0.990289328304
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```



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



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



4.9.1.3. Feature Importance, correctly classifed point

In [166]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_FE,train_y)
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_FE[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_FE[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[  1.48000000e-02   3.45100000e-01   5.200000
00e-03   1.17000000e-02
    1.47000000e-02   4.59000000e-02   5.58400000e-01   3.90000000e-03
    3.00000000e-04]]
Actual Class : 7
--------------------------------------------------
0 Text feature [intracellular] present in test data point [True]
3 Text feature [consistent] present in test data point [True]
4 Text feature [remain] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [activate] present in test data point [True]
9 Text feature [slightly] present in test data point [True]
11 Text feature [institute] present in test data point [True]
15 Text feature [finally] present in test data point [True]
18 Text feature [technology] present in test data point [True]
20 Text feature [3b] present in test data point [True]
21 Text feature [proteolysis] present in test data point [True]
24 Text feature [corresponding] present in test data point [True]
30 Text feature [account] present in test data point [True]
32 Text feature [elements] present in test data point [True]
33 Text feature [folding] present in test data point [True]
34 Text feature [us] present in test data point [True]
35 Text feature [activation] present in test data point [True]
37 Text feature [extracted] present in test data point [True]
38 Text feature [onto] present in test data point [True]
39 Text feature [transforming] present in test data point [True]
40 Text feature [constitutive] present in test data point [True]
42 Text feature [lead] present in test data point [True]
43 Text feature [produce] present in test data point [True]
44 Text feature [transient] present in test data point [True]
45 Text feature [others] present in test data point [True]
48 Text feature [106] present in test data point [True]
49 Text feature [clearly] present in test data point [True]
50 Text feature [contrast] present in test data point [True]
51 Text feature [dose] present in test data point [True]
54 Text feature [signaling] present in test data point [True]
56 Text feature [next] present in test data point [True]
57 Text feature [respect] present in test data point [True]
60 Text feature [culture] present in test data point [True]
62 Text feature [work] present in test data point [True]
65 Text feature [capacity] present in test data point [True]
66 Text feature [new] present in test data point [True]
68 Text feature [directly] present in test data point [True]
70 Text feature [life] present in test data point [True]
71 Text feature [map] present in test data point [True]
72 Text feature [2b] present in test data point [True]
73 Text feature [versus] present in test data point [True]
75 Text feature [94] present in test data point [True]
76 Text feature [position] present in test data point [True]
78 Text feature [appear] present in test data point [True]
81 Text feature [exposure] present in test data point [True]
85 Text feature [mass] present in test data point [True]
88 Text feature [multiple] present in test data point [True]
91 Text feature [promega] present in test data point [True]
93 Text feature [positive] present in test data point [True]
94 Text feature [clone] present in test data point [True]
```

```
95 Text feature [dr] present in test data point [True]
96 Text feature [day] present in test data point [True]
97 Text feature [initially] present in test data point [True]
98 Text feature [expressed] present in test data point [True]
100 Text feature [wt] present in test data point [True]
102 Text feature [105] present in test data point [True]
106 Text feature [along] present in test data point [True]
107 Text feature [initial] present in test data point [True]
109 Text feature [pathway] present in test data point [True]
111 Text feature [old] present in test data point [True]
112 Text feature [signals] present in test data point [True]
113 Text feature [even] present in test data point [True]
114 Text feature [solution] present in test data point [True]
116 Text feature [regulatory] present in test data point [True]
117 Text feature [500] present in test data point [True]
120 Text feature [receptor] present in test data point [True]
124 Text feature [profiling] present in test data point [True]
126 Text feature [phosphate] present in test data point [True]
127 Text feature [cyclin] present in test data point [True]
131 Text feature [human] present in test data point [True]
134 Text feature [ligand] present in test data point [True]
135 Text feature [observation] present in test data point [True]
136 Text feature [center] present in test data point [True]
138 Text feature [equal] present in test data point [True]
140 Text feature [59] present in test data point [True]
141 Text feature [86] present in test data point [True]
144 Text feature [doses] present in test data point [True]
145 Text feature [animals] present in test data point [True]
147 Text feature [depletion] present in test data point [True]
148 Text feature [portion] present in test data point [True]
151 Text feature [leading] present in test data point [True]
153 Text feature [disease] present in test data point [True]
154 Text feature [conclusion] present in test data point [True]
156 Text feature [certain] present in test data point [True]
157 Text feature [bearing] present in test data point [True]
162 Text feature [bars] present in test data point [True]
163 Text feature [nuclear] present in test data point [True]
165 Text feature [progressive] present in test data point [True]
167 Text feature [fisher] present in test data point [True]
168 Text feature [exogenous] present in test data point [True]
169 Text feature [codon] present in test data point [True]
171 Text feature [threonine] present in test data point [True]
178 Text feature [51] present in test data point [True]
179 Text feature [provide] present in test data point [True]
184 Text feature [transiently] present in test data point [True]
185 Text feature [collection] present in test data point [True]
186 Text feature [76] present in test data point [True]
189 Text feature [collected] present in test data point [True]
193 Text feature [examine] present in test data point [True]
195 Text feature [relative] present in test data point [True]
196 Text feature [colonies] present in test data point [True]
200 Text feature [latter] present in test data point [True]
202 Text feature [therapeutic] present in test data point [True]
205 Text feature [inhibition] present in test data point [True]
208 Text feature [interpretation] present in test data point [True]
209 Text feature [diagnosis] present in test data point [True]
210 Text feature [factor] present in test data point [True]
```

```
214 Text feature [synthesis] present in test data point [True]
215 Text feature [markers] present in test data point [True]
217 Text feature [either] present in test data point [True]
220 Text feature [independent] present in test data point [True]
222 Text feature [compound] present in test data point [True]
223 Text feature [basal] present in test data point [True]
228 Text feature [described] present in test data point [True]
229 Text feature [suggest] present in test data point [True]
230 Text feature [cross] present in test data point [True]
231 Text feature [3a] present in test data point [True]
232 Text feature [4d] present in test data point [True]
234 Text feature [profiles] present in test data point [True]
235 Text feature [absence] present in test data point [True]
238 Text feature [kb] present in test data point [True]
239 Text feature [xenograft] present in test data point [True]
240 Text feature [hormone] present in test data point [True]
242 Text feature [sites] present in test data point [True]
244 Text feature [made] present in test data point [True]
247 Text feature [santa] present in test data point [True]
248 Text feature [seeded] present in test data point [True]
250 Text feature [96] present in test data point [True]
253 Text feature [nacl] present in test data point [True]
255 Text feature [promoting] present in test data point [True]
257 Text feature [loading] present in test data point [True]
259 Text feature [length] present in test data point [True]
260 Text feature [fraction] present in test data point [True]
262 Text feature [peptides] present in test data point [True]
263 Text feature [following] present in test data point [True]
265 Text feature [encoding] present in test data point [True]
268 Text feature [standard] present in test data point [True]
269 Text feature [real] present in test data point [True]
271 Text feature [blood] present in test data point [True]
273 Text feature [state] present in test data point [True]
274 Text feature [perhaps] present in test data point [True]
275 Text feature [49] present in test data point [True]
276 Text feature [affecting] present in test data point [True]
277 Text feature [therapies] present in test data point [True]
278 Text feature [strategies] present in test data point [True]
279 Text feature [compare] present in test data point [True]
281 Text feature [toward] present in test data point [True]
283 Text feature [area] present in test data point [True]
284 Text feature [explain] present in test data point [True]
286 Text feature [approximately] present in test data point [True]
287 Text feature [downstream] present in test data point [True]
289 Text feature [71] present in test data point [True]
290 Text feature [cysteine] present in test data point [True]
291 Text feature [presented] present in test data point [True]
292 Text feature [charge] present in test data point [True]
293 Text feature [stably] present in test data point [True]
295 Text feature [independently] present in test data point [True]
296 Text feature [reference] present in test data point [True]
298 Text feature [higher] present in test data point [True]
300 Text feature [vector] present in test data point [True]
301 Text feature [inhibits] present in test data point [True]
303 Text feature [metastasis] present in test data point [True]
304 Text feature [spectrum] present in test data point [True]
305 Text feature [together] present in test data point [True]
```

```
306 Text feature [prior] present in test data point [True]
307 Text feature [ratios] present in test data point [True]
308 Text feature [therapy] present in test data point [True]
309 Text feature [note] present in test data point [True]
310 Text feature [formed] present in test data point [True]
313 Text feature [immunoblot] present in test data point [True]
315 Text feature [amplification] present in test data point [True]
316 Text feature [cycles] present in test data point [True]
321 Text feature [amplified] present in test data point [True]
325 Text feature [already] present in test data point [True]
326 Text feature [adult] present in test data point [True]
328 Text feature [regulated] present in test data point [True]
329 Text feature [dd] present in test data point [True]
330 Text feature [order] present in test data point [True]
331 Text feature [considered] present in test data point [True]
334 Text feature [produced] present in test data point [True]
335 Text feature [residue] present in test data point [True]
336 Text feature [cytoplasmic] present in test data point [True]
337 Text feature [81] present in test data point [True]
338 Text feature [mutants] present in test data point [True]
341 Text feature [double] present in test data point [True]
342 Text feature [alteration] present in test data point [True]
343 Text feature [000] present in test data point [True]
345 Text feature [without] present in test data point [True]
346 Text feature [xenografts] present in test data point [True]
347 Text feature [immunoblotting] present in test data point [True]
348 Text feature [variety] present in test data point [True]
349 Text feature [overall] present in test data point [True]
350 Text feature [drugs] present in test data point [True]
351 Text feature [id] present in test data point [True]
352 Text feature [accumulation] present in test data point [True]
354 Text feature [carried] present in test data point [True]
357 Text feature [reverse] present in test data point [True]
358 Text feature [increased] present in test data point [True]
360 Text feature [approach] present in test data point [True]
361 Text feature [2d] present in test data point [True]
363 Text feature [oncogenes] present in test data point [True]
364 Text feature [exposed] present in test data point [True]
366 Text feature [carcinoma] present in test data point [True]
367 Text feature [empty] present in test data point [True]
368 Text feature [probably] present in test data point [True]
369 Text feature [exhibited] present in test data point [True]
370 Text feature [effect] present in test data point [True]
371 Text feature [size] present in test data point [True]
373 Text feature [sections] present in test data point [True]
374 Text feature [specimens] present in test data point [True]
375 Text feature [2a] present in test data point [True]
377 Text feature [cultured] present in test data point [True]
382 Text feature [experiments] present in test data point [True]
383 Text feature [induction] present in test data point [True]
386 Text feature [subsequent] present in test data point [True]
387 Text feature [develop] present in test data point [True]
388 Text feature [outcome] present in test data point [True]
389 Text feature [primer] present in test data point [True]
390 Text feature [2003] present in test data point [True]
391 Text feature [57] present in test data point [True]
393 Text feature [long] present in test data point [True]
```

```
400 Text feature [frequently] present in test data point [True]
401 Text feature [cohort] present in test data point [True]
402 Text feature [estimated] present in test data point [True]
403 Text feature [patterns] present in test data point [True]
405 Text feature [2002] present in test data point [True]
406 Text feature [high] present in test data point [True]
408 Text feature [survival] present in test data point [True]
410 Text feature [relationship] present in test data point [True]
412 Text feature [agents] present in test data point [True]
415 Text feature [shift] present in test data point [True]
417 Text feature [induced] present in test data point [True]
418 Text feature [refractory] present in test data point [True]
419 Text feature [benefit] present in test data point [True]
421 Text feature [added] present in test data point [True]
422 Text feature [density] present in test data point [True]
423 Text feature [mechanism] present in test data point [True]
424 Text feature [available] present in test data point [True]
426 Text feature [cancers] present in test data point [True]
427 Text feature [alterations] present in test data point [True]
431 Text feature [proportion] present in test data point [True]
436 Text feature [whose] present in test data point [True]
437 Text feature [properties] present in test data point [True]
441 Text feature [incidence] present in test data point [True]
444 Text feature [contain] present in test data point [True]
449 Text feature [primary] present in test data point [True]
453 Text feature [examined] present in test data point [True]
454 Text feature [vectors] present in test data point [True]
457 Text feature [assessed] present in test data point [True]
464 Text feature [plasmid] present in test data point [True]
473 Text feature [injected] present in test data point [True]
476 Text feature [despite] present in test data point [True]
478 Text feature [carcinomas] present in test data point [True]
480 Text feature [antigen] present in test data point [True]
483 Text feature [neither] present in test data point [True]
487 Text feature [material] present in test data point [True]
488 Text feature [fig] present in test data point [True]
492 Text feature [genomic] present in test data point [True]
493 Text feature [driven] present in test data point [True]
494 Text feature [colony] present in test data point [True]
498 Text feature [media] present in test data point [True]
499 Text feature [investigate] present in test data point [True]
Out of the top  500  features  262 are present in query point
```

4.9.1.4. Feature Importance, InCorrectly classifed point

In [168]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_FE[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_FE[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0241  0.3203  0.0194  0.3774  0.0217  0.1
924  0.0391  0.0043  0.0014]]
Actual Class : 2
--------------------------------------------------
3 Text feature [suppressor] present in test data point [True]
4 Text feature [none] present in test data point [True]
7 Text feature [ligase] present in test data point [True]
9 Text feature [laboratory] present in test data point [True]
20 Text feature [importantly] present in test data point [True]
23 Text feature [del] present in test data point [True]
27 Text feature [western] present in test data point [True]
35 Text feature [numbers] present in test data point [True]
41 Text feature [e2] present in test data point [True]
44 Text feature [expected] present in test data point [True]
46 Text feature [low] present in test data point [True]
51 Text feature [catalytic] present in test data point [True]
52 Text feature [transferred] present in test data point [True]
53 Text feature [received] present in test data point [True]
54 Text feature [contribute] present in test data point [True]
57 Text feature [visualized] present in test data point [True]
63 Text feature [bind] present in test data point [True]
75 Text feature [affected] present in test data point [True]
77 Text feature [third] present in test data point [True]
79 Text feature [wt] present in test data point [True]
82 Text feature [ca] present in test data point [True]
92 Text feature [dna] present in test data point [True]
93 Text feature [respect] present in test data point [True]
94 Text feature [tagged] present in test data point [True]
95 Text feature [separated] present in test data point [True]
96 Text feature [may] present in test data point [True]
97 Text feature [frequency] present in test data point [True]
98 Text feature [minutes] present in test data point [True]
101 Text feature [lysed] present in test data point [True]
102 Text feature [loss] present in test data point [True]
109 Text feature [remaining] present in test data point [True]
111 Text feature [tris] present in test data point [True]
113 Text feature [ubiquitin] present in test data point [True]
114 Text feature [washed] present in test data point [True]
120 Text feature [cross] present in test data point [True]
121 Text feature [sigma] present in test data point [True]
123 Text feature [majority] present in test data point [True]
124 Text feature [nsclc] present in test data point [True]
127 Text feature [one] present in test data point [True]
130 Text feature [possible] present in test data point [True]
131 Text feature [tumorigenesis] present in test data point [True]
139 Text feature [substrate] present in test data point [True]
143 Text feature [present] present in test data point [True]
147 Text feature [gene] present in test data point [True]
153 Text feature [currently] present in test data point [True]
155 Text feature [suggesting] present in test data point [True]
157 Text feature [among] present in test data point [True]
158 Text feature [enzyme] present in test data point [True]
159 Text feature [hypothesis] present in test data point [True]
164 Text feature [santa] present in test data point [True]
167 Text feature [ii] present in test data point [True]
168 Text feature [ha] present in test data point [True]
```

```
169 Text feature [produced] present in test data point [True]
174 Text feature [missense] present in test data point [True]
178 Text feature [substrates] present in test data point [True]
180 Text feature [cytokine] present in test data point [True]
183 Text feature [supplemented] present in test data point [True]
186 Text feature [difference] present in test data point [True]
188 Text feature [age] present in test data point [True]
189 Text feature [liver] present in test data point [True]
190 Text feature [since] present in test data point [True]
192 Text feature [prediction] present in test data point [True]
197 Text feature [parental] present in test data point [True]
200 Text feature [domains] present in test data point [True]
209 Text feature [institutional] present in test data point [True]
212 Text feature [alone] present in test data point [True]
213 Text feature [dual] present in test data point [True]
214 Text feature [terms] present in test data point [True]
215 Text feature [plus] present in test data point [True]
222 Text feature [buffer] present in test data point [True]
223 Text feature [regulating] present in test data point [True]
225 Text feature [function] present in test data point [True]
226 Text feature [relatively] present in test data point [True]
231 Text feature [38] present in test data point [True]
232 Text feature [five] present in test data point [True]
236 Text feature [nucleotide] present in test data point [True]
247 Text feature [mm] present in test data point [True]
259 Text feature [construct] present in test data point [True]
268 Text feature [regulation] present in test data point [True]
269 Text feature [29] present in test data point [True]
272 Text feature [reviewed] present in test data point [True]
273 Text feature [whether] present in test data point [True]
276 Text feature [phase] present in test data point [True]
281 Text feature [induced] present in test data point [True]
282 Text feature [modified] present in test data point [True]
296 Text feature [agreement] present in test data point [True]
307 Text feature [reduced] present in test data point [True]
315 Text feature [university] present in test data point [True]
316 Text feature [receptors] present in test data point [True]
317 Text feature [temperature] present in test data point [True]
319 Text feature [characterized] present in test data point [True]
320 Text feature [background] present in test data point [True]
330 Text feature [2007] present in test data point [True]
334 Text feature [correlated] present in test data point [True]
336 Text feature [gain] present in test data point [True]
341 Text feature [26] present in test data point [True]
349 Text feature [analyses] present in test data point [True]
357 Text feature [polymerase] present in test data point [True]
363 Text feature [located] present in test data point [True]
372 Text feature [standard] present in test data point [True]
374 Text feature [could] present in test data point [True]
375 Text feature [co] present in test data point [True]
378 Text feature [us] present in test data point [True]
381 Text feature [transfected] present in test data point [True]
392 Text feature [paired] present in test data point [True]
396 Text feature [thus] present in test data point [True]
404 Text feature [kit] present in test data point [True]
405 Text feature [molecules] present in test data point [True]
407 Text feature [inhibited] present in test data point [True]
```

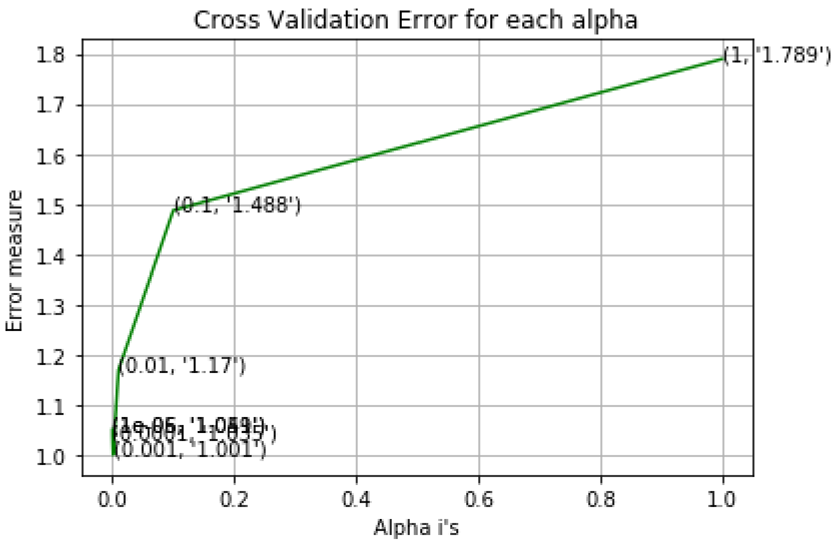
```
430 Text feature [conjugated] present in test data point [True]
436 Text feature [37] present in test data point [True]
437 Text feature [rates] present in test data point [True]
439 Text feature [allelic] present in test data point [True]
446 Text feature [bovine] present in test data point [True]
451 Text feature [intact] present in test data point [True]
452 Text feature [apoptosis] present in test data point [True]
461 Text feature [levels] present in test data point [True]
469 Text feature [either] present in test data point [True]
476 Text feature [demonstrated] present in test data point [True]
480 Text feature [decreased] present in test data point [True]
483 Text feature [30] present in test data point [True]
494 Text feature [markers] present in test data point [True]
Out of the top  500  features  122 are present in query point
```

4.9.2. without balanced class

4.9.2.1. hyper parameter tuning

In [169]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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 = [0.005,0.05,0.5]
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_FE, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_FE, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_FE)
    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_FE, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_FE, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_FE)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_FE)
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_FE)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.05101778568
for alpha = 1e-05
Log Loss : 1.04851582927
for alpha = 0.0001
Log Loss : 1.03541789132
for alpha = 0.001
Log Loss : 1.00085322837
for alpha = 0.01
Log Loss : 1.16994252674
for alpha = 0.1
Log Loss : 1.48756879585
for alpha = 1
Log Loss : 1.78933990513
```

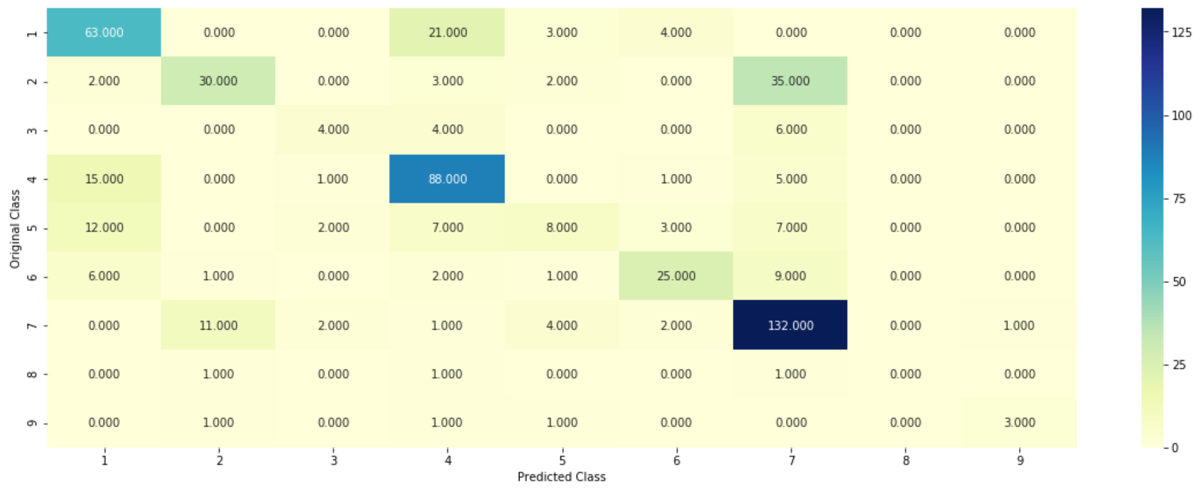


```
For values of best alpha =  0.001 The train log loss is: 0.578572265297
For values of best alpha =  0.001 The cross validation log loss is: 1.0008532
2837
For values of best alpha =  0.001 The test log loss is: 1.05357299175
```

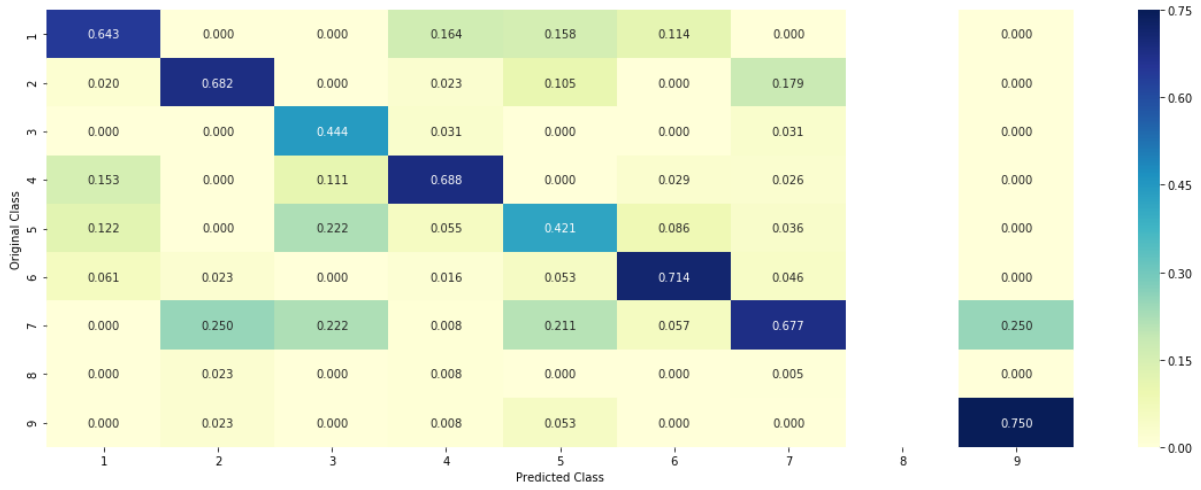
4.9.2.2. Testing the model with best hyperparameter

In [170]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
lessons/geometric-intuition-1/
#------------------------------
clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random
_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_FE, train_y, cv_x_oneho
tCoding_FE, cv_y, clf)
```

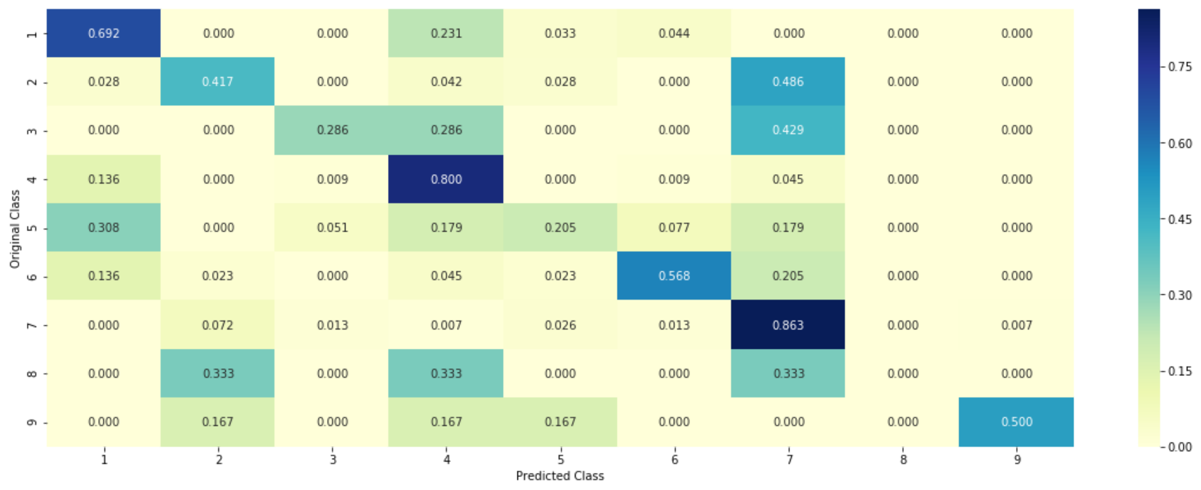
```
Log loss : 1.00085322837
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```



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



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



4.9.2.3. Feature Importance, Correctly classified point

In [172]:
```
clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random
_state=42)
clf.fit(train_x_onehotCoding_FE,train_y)
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_FE[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_FE[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[  1.38000000e-02   3.44300000e-01   4.000000
00e-03   1.13000000e-02
    1.15000000e-02   4.03000000e-02   5.69800000e-01   4.70000000e-03
    2.00000000e-04]]
Actual Class : 7
--------------------------------------------------
1 Text feature [intracellular] present in test data point [True]
3 Text feature [remain] present in test data point [True]
4 Text feature [consistent] present in test data point [True]
6 Text feature [activate] present in test data point [True]
7 Text feature [institute] present in test data point [True]
11 Text feature [activated] present in test data point [True]
12 Text feature [finally] present in test data point [True]
16 Text feature [slightly] present in test data point [True]
18 Text feature [proteolysis] present in test data point [True]
19 Text feature [technology] present in test data point [True]
25 Text feature [3b] present in test data point [True]
28 Text feature [account] present in test data point [True]
32 Text feature [corresponding] present in test data point [True]
33 Text feature [folding] present in test data point [True]
34 Text feature [elements] present in test data point [True]
35 Text feature [transforming] present in test data point [True]
36 Text feature [onto] present in test data point [True]
37 Text feature [contrast] present in test data point [True]
40 Text feature [clearly] present in test data point [True]
41 Text feature [us] present in test data point [True]
44 Text feature [capacity] present in test data point [True]
45 Text feature [106] present in test data point [True]
46 Text feature [constitutive] present in test data point [True]
48 Text feature [activation] present in test data point [True]
49 Text feature [new] present in test data point [True]
52 Text feature [94] present in test data point [True]
53 Text feature [positive] present in test data point [True]
54 Text feature [respect] present in test data point [True]
56 Text feature [produce] present in test data point [True]
57 Text feature [initial] present in test data point [True]
58 Text feature [culture] present in test data point [True]
59 Text feature [lead] present in test data point [True]
61 Text feature [initially] present in test data point [True]
62 Text feature [2b] present in test data point [True]
63 Text feature [directly] present in test data point [True]
64 Text feature [wt] present in test data point [True]
65 Text feature [transient] present in test data point [True]
66 Text feature [signaling] present in test data point [True]
67 Text feature [expressed] present in test data point [True]
68 Text feature [extracted] present in test data point [True]
69 Text feature [dose] present in test data point [True]
71 Text feature [fisher] present in test data point [True]
72 Text feature [others] present in test data point [True]
76 Text feature [clone] present in test data point [True]
77 Text feature [exposure] present in test data point [True]
80 Text feature [depletion] present in test data point [True]
81 Text feature [appear] present in test data point [True]
85 Text feature [promega] present in test data point [True]
87 Text feature [life] present in test data point [True]
88 Text feature [next] present in test data point [True]
```

```
89 Text feature [map] present in test data point [True]
91 Text feature [position] present in test data point [True]
93 Text feature [versus] present in test data point [True]
95 Text feature [signals] present in test data point [True]
98 Text feature [progressive] present in test data point [True]
101 Text feature [mass] present in test data point [True]
104 Text feature [observation] present in test data point [True]
108 Text feature [86] present in test data point [True]
112 Text feature [even] present in test data point [True]
117 Text feature [along] present in test data point [True]
118 Text feature [day] present in test data point [True]
120 Text feature [either] present in test data point [True]
122 Text feature [sites] present in test data point [True]
123 Text feature [regulatory] present in test data point [True]
126 Text feature [standard] present in test data point [True]
129 Text feature [latter] present in test data point [True]
130 Text feature [certain] present in test data point [True]
131 Text feature [examine] present in test data point [True]
133 Text feature [doses] present in test data point [True]
134 Text feature [considered] present in test data point [True]
138 Text feature [solution] present in test data point [True]
139 Text feature [500] present in test data point [True]
141 Text feature [inhibition] present in test data point [True]
142 Text feature [suggest] present in test data point [True]
143 Text feature [exogenous] present in test data point [True]
144 Text feature [leading] present in test data point [True]
147 Text feature [59] present in test data point [True]
148 Text feature [collected] present in test data point [True]
149 Text feature [human] present in test data point [True]
150 Text feature [profiling] present in test data point [True]
153 Text feature [profiles] present in test data point [True]
155 Text feature [portion] present in test data point [True]
159 Text feature [diagnosis] present in test data point [True]
164 Text feature [cyclin] present in test data point [True]
166 Text feature [nacl] present in test data point [True]
168 Text feature [work] present in test data point [True]
169 Text feature [105] present in test data point [True]
171 Text feature [interpretation] present in test data point [True]
175 Text feature [ligand] present in test data point [True]
177 Text feature [compare] present in test data point [True]
179 Text feature [3a] present in test data point [True]
180 Text feature [colonies] present in test data point [True]
181 Text feature [downstream] present in test data point [True]
182 Text feature [kb] present in test data point [True]
183 Text feature [described] present in test data point [True]
184 Text feature [codon] present in test data point [True]
185 Text feature [spectrum] present in test data point [True]
192 Text feature [51] present in test data point [True]
196 Text feature [disease] present in test data point [True]
202 Text feature [phosphate] present in test data point [True]
203 Text feature [bars] present in test data point [True]
204 Text feature [dr] present in test data point [True]
205 Text feature [equal] present in test data point [True]
208 Text feature [perhaps] present in test data point [True]
209 Text feature [made] present in test data point [True]
210 Text feature [order] present in test data point [True]
212 Text feature [independently] present in test data point [True]
```

```
214 Text feature [provide] present in test data point [True]
216 Text feature [96] present in test data point [True]
217 Text feature [markers] present in test data point [True]
220 Text feature [stably] present in test data point [True]
223 Text feature [vector] present in test data point [True]
224 Text feature [produced] present in test data point [True]
226 Text feature [affecting] present in test data point [True]
227 Text feature [76] present in test data point [True]
228 Text feature [higher] present in test data point [True]
229 Text feature [center] present in test data point [True]
234 Text feature [bearing] present in test data point [True]
235 Text feature [old] present in test data point [True]
241 Text feature [transiently] present in test data point [True]
243 Text feature [hormone] present in test data point [True]
244 Text feature [relative] present in test data point [True]
249 Text feature [effect] present in test data point [True]
250 Text feature [receptor] present in test data point [True]
254 Text feature [pathway] present in test data point [True]
255 Text feature [xenograft] present in test data point [True]
261 Text feature [mutants] present in test data point [True]
262 Text feature [outcome] present in test data point [True]
263 Text feature [multiple] present in test data point [True]
264 Text feature [carcinoma] present in test data point [True]
270 Text feature [threonine] present in test data point [True]
271 Text feature [animals] present in test data point [True]
274 Text feature [collection] present in test data point [True]
275 Text feature [overall] present in test data point [True]
278 Text feature [approximately] present in test data point [True]
279 Text feature [immunoblotting] present in test data point [True]
281 Text feature [cycles] present in test data point [True]
282 Text feature [compound] present in test data point [True]
284 Text feature [without] present in test data point [True]
288 Text feature [49] present in test data point [True]
289 Text feature [conclusion] present in test data point [True]
290 Text feature [reverse] present in test data point [True]
294 Text feature [seeded] present in test data point [True]
297 Text feature [synthesis] present in test data point [True]
298 Text feature [residue] present in test data point [True]
299 Text feature [empty] present in test data point [True]
305 Text feature [2a] present in test data point [True]
309 Text feature [high] present in test data point [True]
310 Text feature [fraction] present in test data point [True]
311 Text feature [id] present in test data point [True]
312 Text feature [dd] present in test data point [True]
313 Text feature [blood] present in test data point [True]
315 Text feature [factor] present in test data point [True]
316 Text feature [absence] present in test data point [True]
320 Text feature [santa] present in test data point [True]
321 Text feature [nuclear] present in test data point [True]
322 Text feature [cross] present in test data point [True]
323 Text feature [state] present in test data point [True]
325 Text feature [following] present in test data point [True]
328 Text feature [therapeutic] present in test data point [True]
330 Text feature [approach] present in test data point [True]
335 Text feature [promoting] present in test data point [True]
336 Text feature [induction] present in test data point [True]
337 Text feature [71] present in test data point [True]
```

```
338 Text feature [loading] present in test data point [True]
339 Text feature [length] present in test data point [True]
340 Text feature [therapies] present in test data point [True]
342 Text feature [regulated] present in test data point [True]
343 Text feature [peptides] present in test data point [True]
346 Text feature [cytoplasmic] present in test data point [True]
347 Text feature [4d] present in test data point [True]
348 Text feature [charge] present in test data point [True]
352 Text feature [already] present in test data point [True]
355 Text feature [basal] present in test data point [True]
356 Text feature [amplified] present in test data point [True]
360 Text feature [ratios] present in test data point [True]
362 Text feature [vs] present in test data point [True]
366 Text feature [together] present in test data point [True]
372 Text feature [accumulation] present in test data point [True]
373 Text feature [immunoblot] present in test data point [True]
374 Text feature [antigen] present in test data point [True]
375 Text feature [drugs] present in test data point [True]
377 Text feature [translation] present in test data point [True]
380 Text feature [proportion] present in test data point [True]
382 Text feature [maintained] present in test data point [True]
386 Text feature [oncogenes] present in test data point [True]
391 Text feature [presented] present in test data point [True]
393 Text feature [exhibited] present in test data point [True]
394 Text feature [subsequent] present in test data point [True]
395 Text feature [formed] present in test data point [True]
401 Text feature [mechanism] present in test data point [True]
404 Text feature [primary] present in test data point [True]
409 Text feature [reference] present in test data point [True]
415 Text feature [probably] present in test data point [True]
416 Text feature [pa] present in test data point [True]
419 Text feature [toward] present in test data point [True]
420 Text feature [real] present in test data point [True]
426 Text feature [investigate] present in test data point [True]
428 Text feature [chromosome] present in test data point [True]
432 Text feature [000] present in test data point [True]
433 Text feature [genomic] present in test data point [True]
436 Text feature [double] present in test data point [True]
437 Text feature [independent] present in test data point [True]
438 Text feature [vectors] present in test data point [True]
440 Text feature [cultured] present in test data point [True]
447 Text feature [explain] present in test data point [True]
449 Text feature [morphology] present in test data point [True]
451 Text feature [inhibits] present in test data point [True]
454 Text feature [2002] present in test data point [True]
455 Text feature [carried] present in test data point [True]
462 Text feature [develop] present in test data point [True]
463 Text feature [refractory] present in test data point [True]
467 Text feature [online] present in test data point [True]
471 Text feature [colony] present in test data point [True]
472 Text feature [long] present in test data point [True]
474 Text feature [direct] present in test data point [True]
475 Text feature [various] present in test data point [True]
476 Text feature [57] present in test data point [True]
477 Text feature [cysteine] present in test data point [True]
480 Text feature [demonstrate] present in test data point [True]
481 Text feature [active] present in test data point [True]
```

```
482 Text feature [relationship] present in test data point [True]
483 Text feature [estimated] present in test data point [True]
487 Text feature [carcinomas] present in test data point [True]
495 Text feature [benefit] present in test data point [True]
Out of the top  500  features  225 are present in query point
```

4.9.2.4. Feature Importance, Incorrectly classified point

In [173]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_FE[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_FE[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0278  0.3452  0.0159  0.3906  0.026   0.1
543  0.0341  0.0053  0.0008]]
Actual Class : 2
--------------------------------------------------
3 Text feature [suppressor] present in test data point [True]
6 Text feature [none] present in test data point [True]
7 Text feature [ligase] present in test data point [True]
9 Text feature [del] present in test data point [True]
10 Text feature [laboratory] present in test data point [True]
14 Text feature [importantly] present in test data point [True]
15 Text feature [low] present in test data point [True]
22 Text feature [numbers] present in test data point [True]
23 Text feature [western] present in test data point [True]
32 Text feature [expected] present in test data point [True]
37 Text feature [contribute] present in test data point [True]
38 Text feature [e2] present in test data point [True]
41 Text feature [received] present in test data point [True]
46 Text feature [bind] present in test data point [True]
48 Text feature [third] present in test data point [True]
50 Text feature [transferred] present in test data point [True]
51 Text feature [catalytic] present in test data point [True]
67 Text feature [separated] present in test data point [True]
80 Text feature [tagged] present in test data point [True]
81 Text feature [visualized] present in test data point [True]
86 Text feature [affected] present in test data point [True]
90 Text feature [respect] present in test data point [True]
93 Text feature [minutes] present in test data point [True]
96 Text feature [cross] present in test data point [True]
97 Text feature [remaining] present in test data point [True]
101 Text feature [ca] present in test data point [True]
104 Text feature [loss] present in test data point [True]
107 Text feature [ha] present in test data point [True]
108 Text feature [among] present in test data point [True]
109 Text feature [ubiquitin] present in test data point [True]
113 Text feature [frequency] present in test data point [True]
116 Text feature [dna] present in test data point [True]
128 Text feature [majority] present in test data point [True]
131 Text feature [mm] present in test data point [True]
134 Text feature [present] present in test data point [True]
137 Text feature [currently] present in test data point [True]
138 Text feature [wt] present in test data point [True]
139 Text feature [suggesting] present in test data point [True]
144 Text feature [substrate] present in test data point [True]
147 Text feature [washed] present in test data point [True]
148 Text feature [tumorigenesis] present in test data point [True]
150 Text feature [may] present in test data point [True]
157 Text feature [enzyme] present in test data point [True]
167 Text feature [santa] present in test data point [True]
171 Text feature [prediction] present in test data point [True]
173 Text feature [plus] present in test data point [True]
175 Text feature [one] present in test data point [True]
182 Text feature [lysed] present in test data point [True]
193 Text feature [hypothesis] present in test data point [True]
198 Text feature [dual] present in test data point [True]
199 Text feature [difference] present in test data point [True]
202 Text feature [thus] present in test data point [True]
```

```
206 Text feature [age] present in test data point [True]
211 Text feature [reviewed] present in test data point [True]
213 Text feature [reduced] present in test data point [True]
217 Text feature [produced] present in test data point [True]
220 Text feature [institutional] present in test data point [True]
224 Text feature [parental] present in test data point [True]
225 Text feature [possible] present in test data point [True]
232 Text feature [characterized] present in test data point [True]
233 Text feature [38] present in test data point [True]
234 Text feature [missense] present in test data point [True]
239 Text feature [supplemented] present in test data point [True]
243 Text feature [nsclc] present in test data point [True]
247 Text feature [since] present in test data point [True]
249 Text feature [26] present in test data point [True]
252 Text feature [tris] present in test data point [True]
253 Text feature [gene] present in test data point [True]
258 Text feature [sigma] present in test data point [True]
259 Text feature [cytokine] present in test data point [True]
265 Text feature [2007] present in test data point [True]
268 Text feature [ii] present in test data point [True]
273 Text feature [domains] present in test data point [True]
275 Text feature [alone] present in test data point [True]
276 Text feature [function] present in test data point [True]
277 Text feature [modified] present in test data point [True]
281 Text feature [induced] present in test data point [True]
283 Text feature [agreement] present in test data point [True]
284 Text feature [relatively] present in test data point [True]
287 Text feature [whether] present in test data point [True]
300 Text feature [could] present in test data point [True]
301 Text feature [regulation] present in test data point [True]
305 Text feature [co] present in test data point [True]
309 Text feature [liver] present in test data point [True]
313 Text feature [terms] present in test data point [True]
320 Text feature [construct] present in test data point [True]
325 Text feature [tyrosine] present in test data point [True]
327 Text feature [transfected] present in test data point [True]
333 Text feature [29] present in test data point [True]
336 Text feature [year] present in test data point [True]
344 Text feature [regulating] present in test data point [True]
346 Text feature [analyses] present in test data point [True]
348 Text feature [apoptosis] present in test data point [True]
349 Text feature [nucleotide] present in test data point [True]
351 Text feature [substrates] present in test data point [True]
371 Text feature [receptors] present in test data point [True]
372 Text feature [demonstrated] present in test data point [True]
377 Text feature [followed] present in test data point [True]
395 Text feature [temperature] present in test data point [True]
397 Text feature [buffer] present in test data point [True]
399 Text feature [molecules] present in test data point [True]
404 Text feature [gain] present in test data point [True]
411 Text feature [polymerase] present in test data point [True]
413 Text feature [background] present in test data point [True]
421 Text feature [blue] present in test data point [True]
423 Text feature [inhibited] present in test data point [True]
426 Text feature [kit] present in test data point [True]
431 Text feature [us] present in test data point [True]
432 Text feature [either] present in test data point [True]
```

```
442 Text feature [allelic] present in test data point [True]
445 Text feature [phase] present in test data point [True]
452 Text feature [kinase] present in test data point [True]
453 Text feature [lysis] present in test data point [True]
465 Text feature [paired] present in test data point [True]
469 Text feature [however] present in test data point [True]
473 Text feature [detected] present in test data point [True]
474 Text feature [located] present in test data point [True]
477 Text feature [intact] present in test data point [True]
478 Text feature [293t] present in test data point [True]
479 Text feature [markers] present in test data point [True]
484 Text feature [medium] present in test data point [True]
490 Text feature [sds] present in test data point [True]
491 Text feature [five] present in test data point [True]
Out of the top  500  features  123 are present in query point
```

4.10. Logistic Regression using bag of words featurization

In [174]:
```
def get_impfeature_names_count(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3, ngram_range=(1,2))

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

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

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

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

In [175]:
```
text_vectorizer = CountVectorizer(min_df=3, ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])

# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=0)

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

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

Stacking the three type of features

In [176]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,t
rain_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,tes
t_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_vari
ation_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_fea
ture_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_r
esponseCoding))
```

In [177]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_onehotCoding.shape)
```

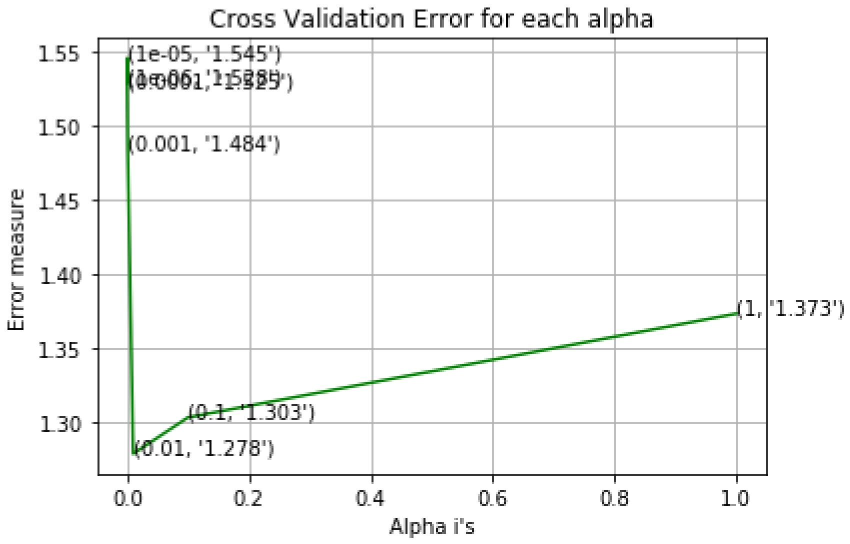
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 769675)
(number of data points * number of features) in test data =  (665, 769675)
(number of data points * number of features) in cross validation data = (532,
769675)
```

4.10.1. without class balancing

4.10.1.1. hyperparameter tuning

In [178]:
```
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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.52844752065
for alpha = 1e-05
Log Loss : 1.54519654311
for alpha = 0.0001
Log Loss : 1.52519044026
for alpha = 0.001
Log Loss : 1.48389768926
for alpha = 0.01
Log Loss : 1.2780698001
for alpha = 0.1
Log Loss : 1.30268551016
for alpha = 1
Log Loss : 1.37278422829
```

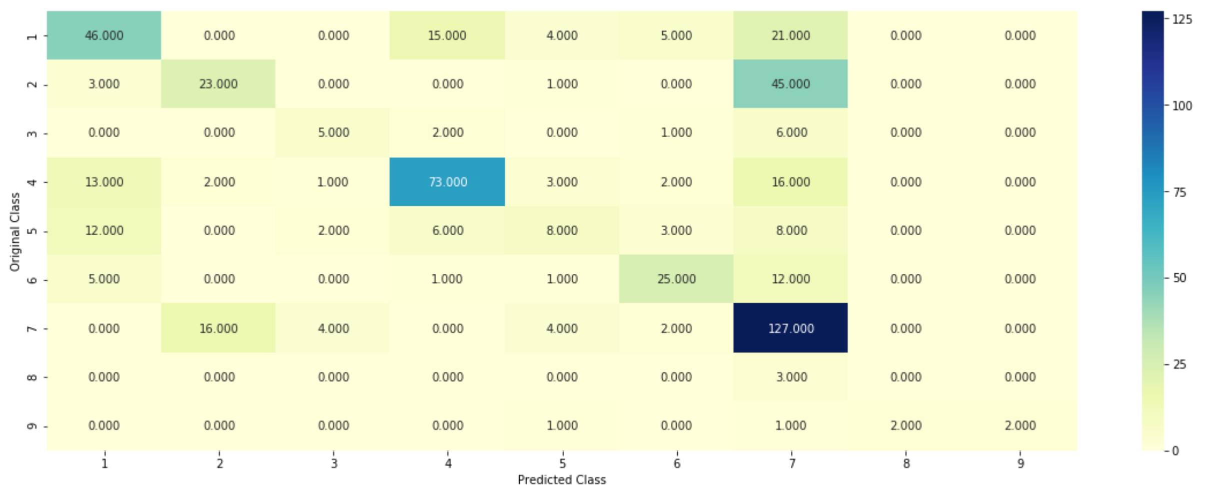


```
For values of best alpha =  0.01 The train log loss is: 0.814857601342
For values of best alpha =  0.01 The cross validation log loss is: 1.27806980
01
For values of best alpha =  0.01 The test log loss is: 1.2579087198
```

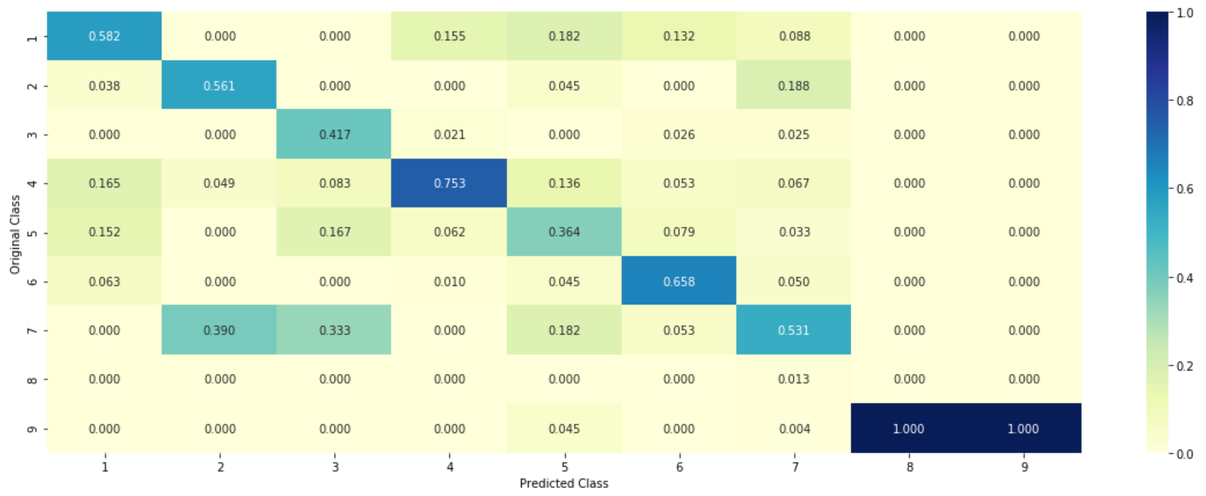
4.10.1.2. Testing model using best hyperparameter

In [179]:
```
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_onehotCo
ding, cv_y, clf)
```

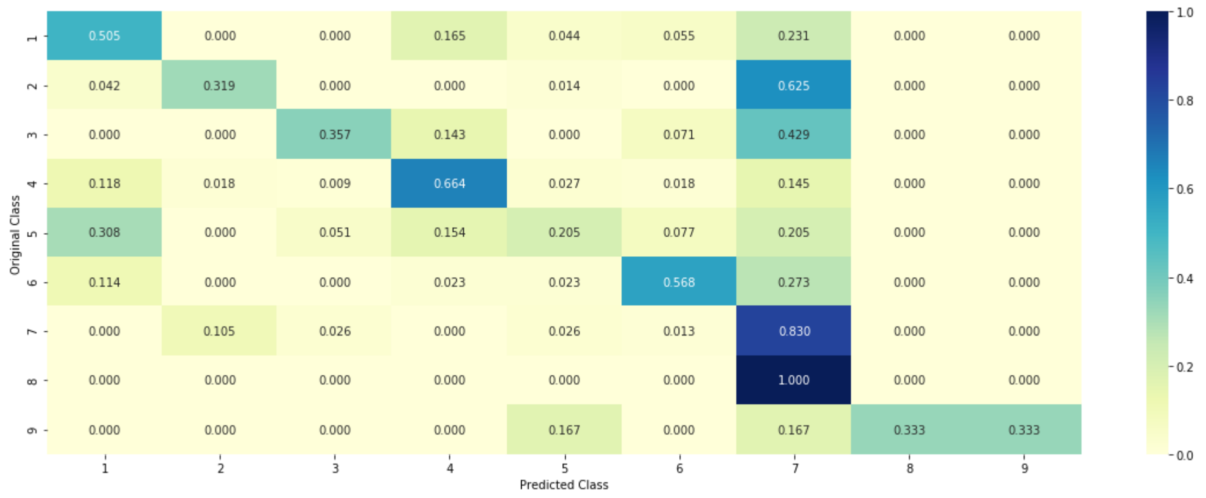
```
Log loss : 1.2780698001
Number of mis-classified points : 0.4191729323308271
-------------------- Confusion matrix --------------------
```



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



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



4.10.1.3. Feature importance, correctly classified point

In [181]:
```
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_count(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_in
dex], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0879  0.3871  0.0103  0.0275  0.0128  0.3
522  0.1185  0.0028  0.0009]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.10.1.4. Feature importance, Incorrectly classified point

In [183]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_count(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_in
dex], no_feature)
```

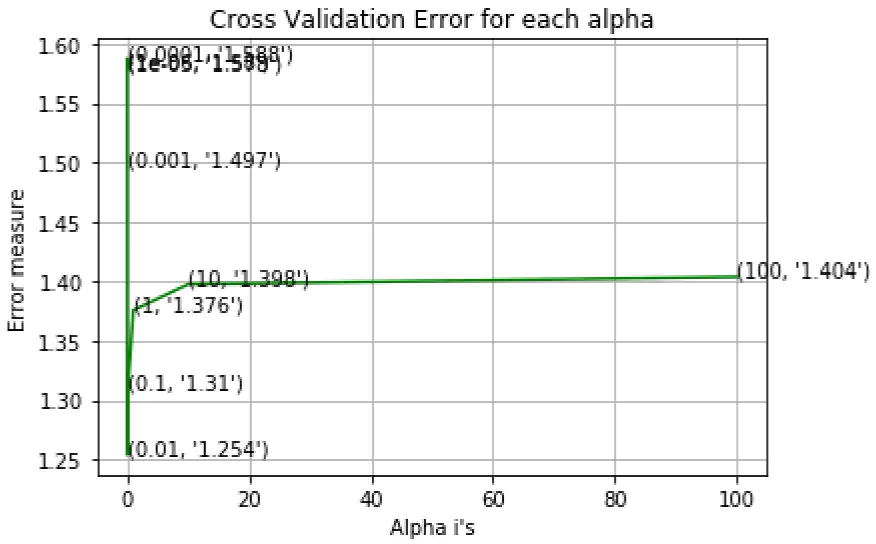
```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.1013  0.1079  0.0188  0.5049  0.0381  0.0
358  0.1843  0.0053  0.0036]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.10.2. With class balancing

4.10.2.1. hyperparameter tuning

In [185]:
```
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-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', 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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.58020458507
for alpha = 1e-05
Log Loss : 1.5783738941
for alpha = 0.0001
Log Loss : 1.58770255162
for alpha = 0.001
Log Loss : 1.49673815693
for alpha = 0.01
Log Loss : 1.25391998573
for alpha = 0.1
Log Loss : 1.30977081741
for alpha = 1
Log Loss : 1.37606047145
for alpha = 10
Log Loss : 1.3979083669
for alpha = 100
Log Loss : 1.4040956803
```

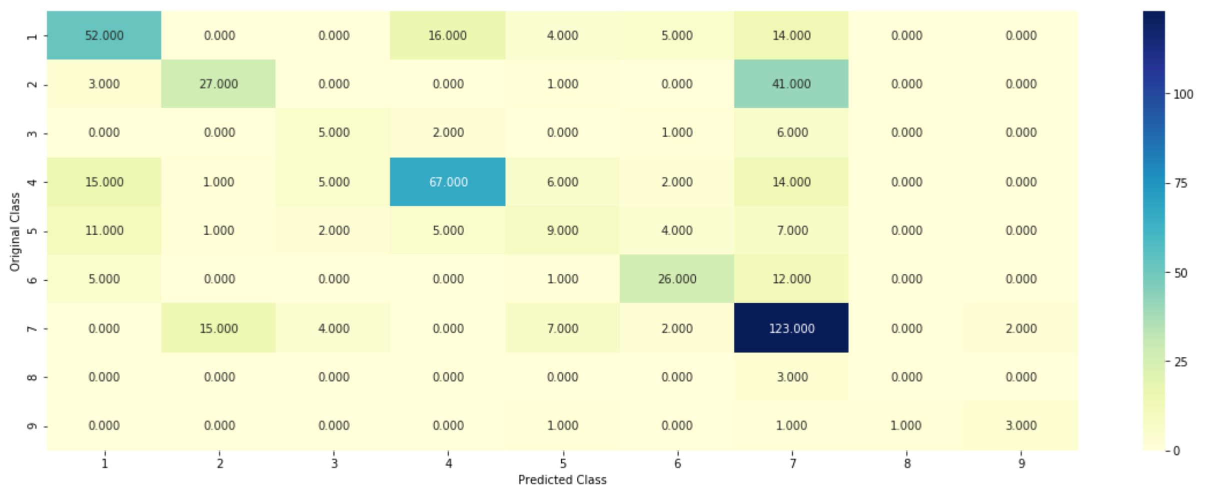


```
For values of best alpha =  0.01 The train log loss is: 0.815007815167
For values of best alpha =  0.01 The cross validation log loss is: 1.25391998
573
For values of best alpha =  0.01 The test log loss is: 1.25149603602
```

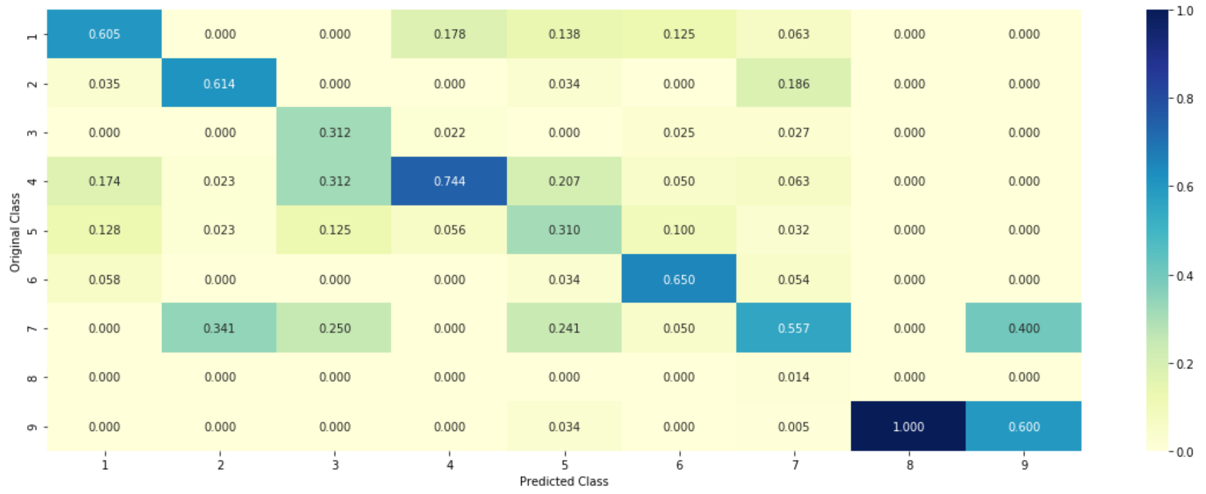
4.10.2.2. Testing model using best hyperparameter

In [186]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCo
ding, cv_y, clf)
```

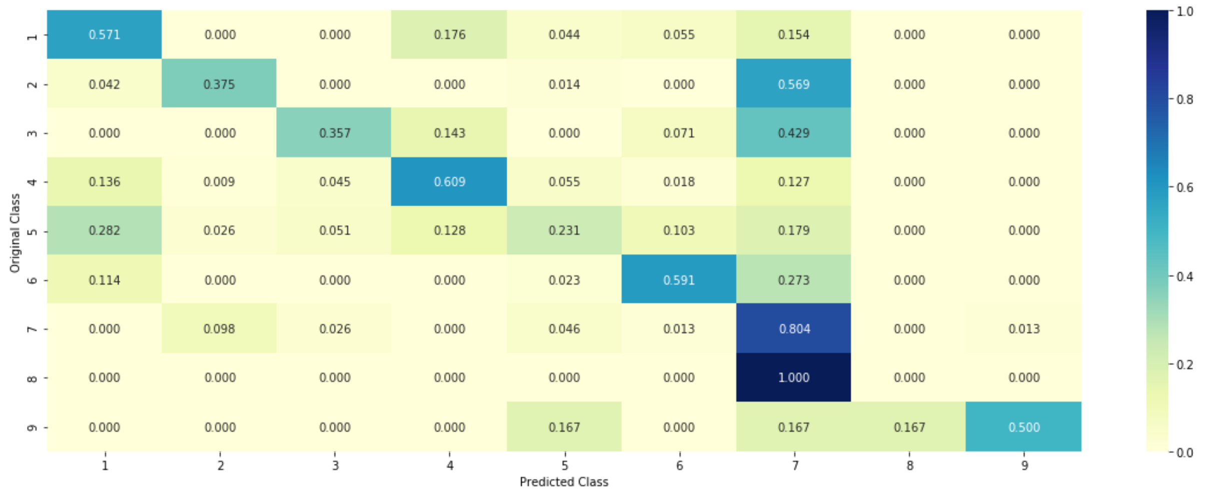
```
Log loss : 1.25391998573
Number of mis-classified points : 0.413533834586466l4
-------------------- Confusion matrix --------------------
```



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



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



4.10.2.3. Feature importance , correctly classified point

In [187]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_count(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_in
dex], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0802  0.3774  0.013   0.0482  0.0167  0.3
576  0.0958  0.0046  0.0065]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.10.2.4. Feature importance , correctly classified point

In [189]:
```
test_point_index = 10
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_count(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_in
dex], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0504  0.0858  0.0086  0.0328  0.0173  0.0
029  0.7782  0.0128  0.0112]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

sr. no.	Model	Featurization	Train error	CV error	Test error	Misclassified points
1	Naive bayes + one-hot coding	Tfidf	0.5675	1.2543	1.2847	0.4060
2	K-NN + one-hot coding	Tfidf	0.6730	1.0140	1.1210	0.3759
3	Logistic Regression + one-hot coding + with class balancing	Tfidf	0.4163	1.0278	1.0624	0.3759
4	Logistic Regression + one-hot coding + without class balancing	Tfidf	0.4104	1.0564	1.0967	0.3627
5	Linear SVM + one-hot coding	Tfidf	0.4849	1.0565	1.1049	0.3609
6	Random forest + one-hot coding	Tfidf	0.8482	1.2032	1.2156	0.4360
7	Random forest + Response coding	Tfidf	0.0534	1.2817	1.3340	0.4360
8	Stacking classifier + one-hot coding	Tfidf	0.5692	1.1952	1.2412	0.4060
9	Maximum voting classifier + one-hot coding	Tfidf	0.8338	1.2099	1.2441	0.4015
10	GBDT Clasifier + one-hot coding	Tfidf	0.5463	1.1681	1.1896	0.3646
11	GBDT Clasifier + Response coding	Tfidf	0.0239	1.4683	1.4846	0.5000
12	Logistic Regression + sqrt feature engineering + with class balancing	Tfidf	0.5800	0.9902	1.0345	0.3439
13	Logistic Regression + sqrt feature engineering + without class balancing	Tfidf	0.5785	1.0008	1.0535	0.3364
14	Logistic Regression + unigram and bigram + without class balancing	bag of words	0.8148	1.2780	1.2579	0.4191
15	Logistic Regression + unigram and bigram + with class balancing	bag of words	0.8150	1.2539	1.2514	0.4135

Observation

- We use Unigram in all tfidf featurization.
- Logistic Regression and linear SVM work well in this data set.
- Random forest and GBDT overfit the model with Response coding i.e difference between there Train error and CV error are high.
- Using "Logistic Regression + sqrt feature engineering + with class balancing" , we get very less log loss for cv and test data which is the lowest in all.