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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

#from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",nam
es=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:
```
# Loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 201.99642683599996 seconds
```

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

Out[6]:

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

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

Out[7]:

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

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

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

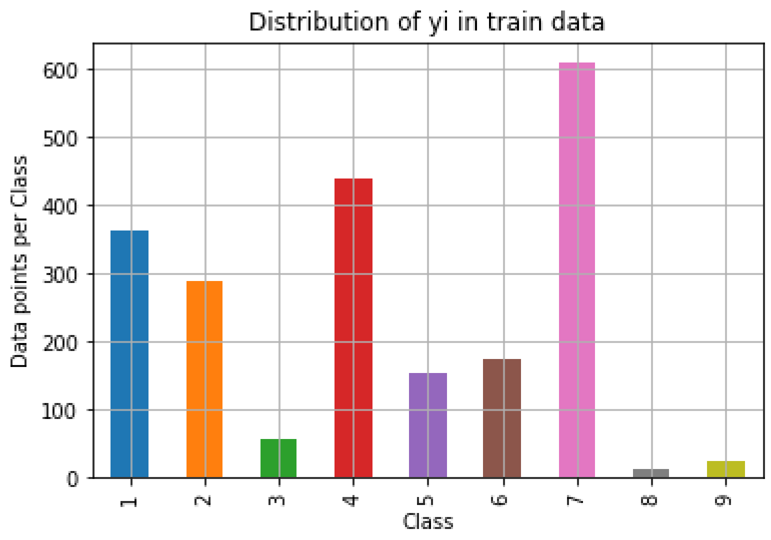
In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

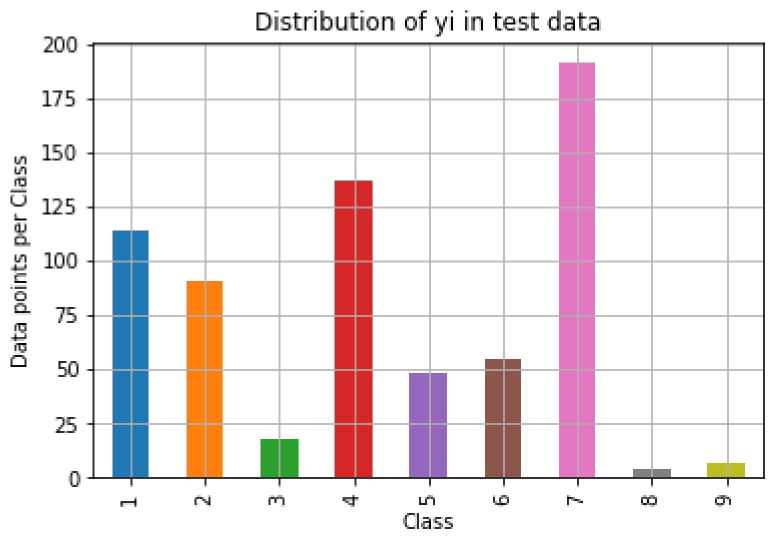
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:
```
# it returns a dict, keys as class labels and values as the number of data 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')
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')
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')
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
```

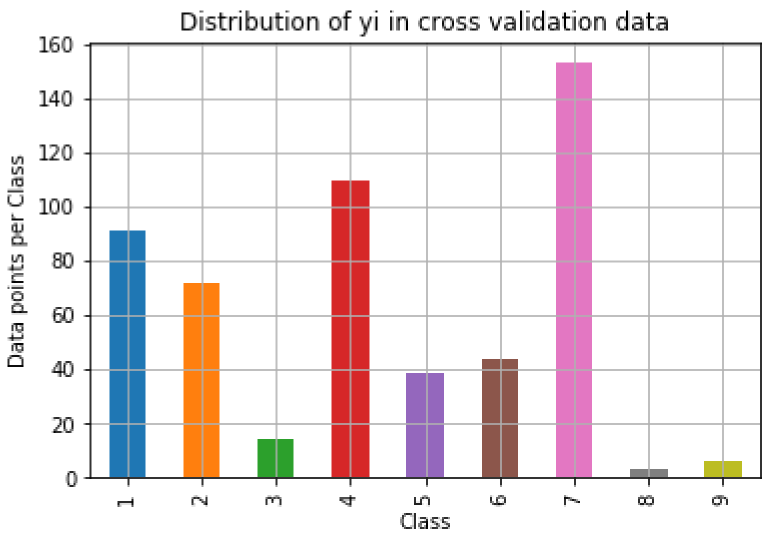
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```



```
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 %)
```

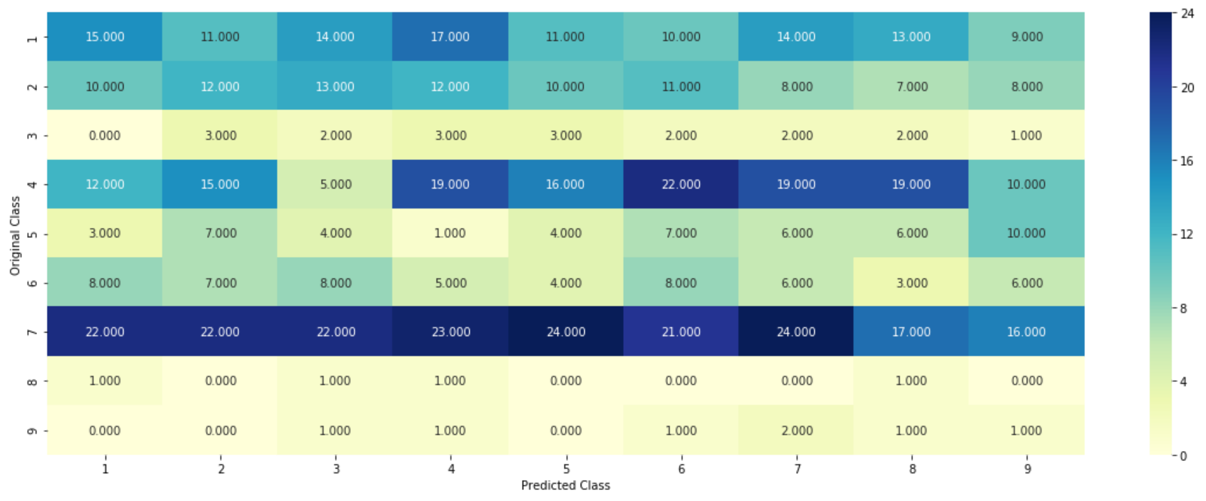
3.2 Prediction using a 'Random' Model

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

In [13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i 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()
```

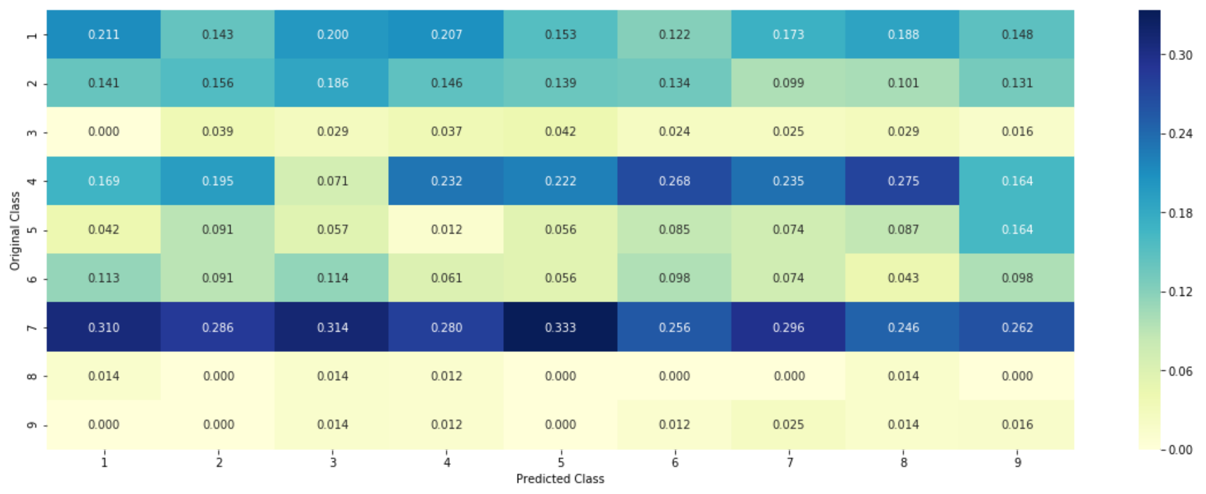
In [14]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
r sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.4806607268305183
Log loss on Test Data using Random Model 2.446522108208394
-------------------- Confusion matrix --------------------
```



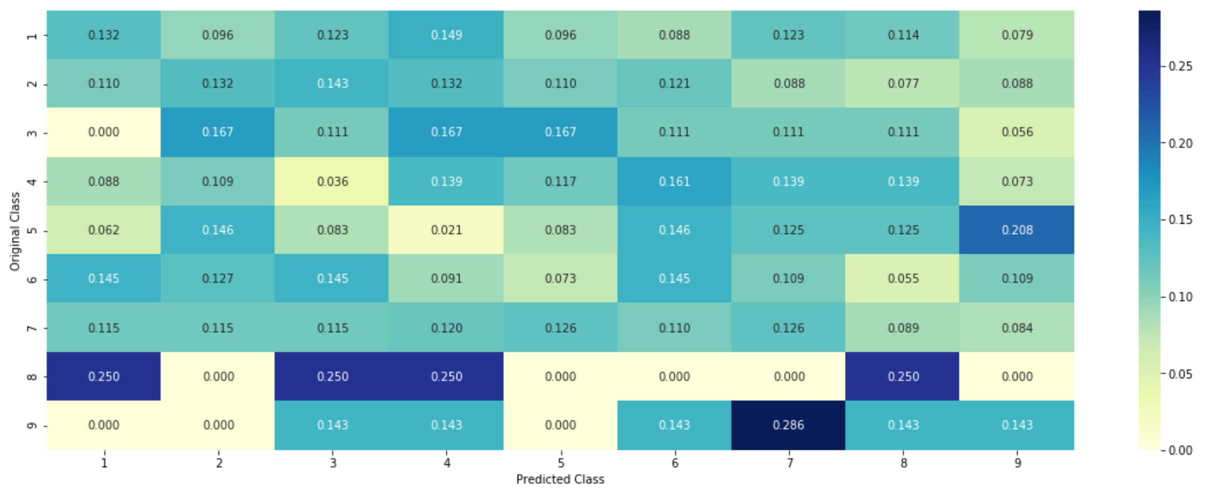
	1	2	3	4	5	6	7	8	9
1	15.000	11.000	14.000	17.000	11.000	10.000	14.000	13.000	9.000
2	10.000	12.000	13.000	12.000	10.000	11.000	8.000	7.000	8.000
3	0.000	3.000	2.000	3.000	3.000	2.000	2.000	2.000	1.000
4	12.000	15.000	5.000	19.000	16.000	22.000	19.000	19.000	10.000
5	3.000	7.000	4.000	1.000	4.000	7.000	6.000	6.000	10.000
6	8.000	7.000	8.000	5.000	4.000	8.000	6.000	3.000	6.000
7	22.000	22.000	22.000	23.000	24.000	21.000	24.000	17.000	16.000
8	1.000	0.000	1.000	1.000	0.000	0.000	0.000	1.000	0.000
9	0.000	0.000	1.000	1.000	0.000	1.000	2.000	1.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.211	0.143	0.200	0.207	0.153	0.122	0.173	0.188	0.148
2	0.141	0.156	0.186	0.146	0.139	0.134	0.099	0.101	0.131
3	0.000	0.039	0.029	0.037	0.042	0.024	0.025	0.029	0.016
4	0.169	0.195	0.071	0.232	0.222	0.268	0.235	0.275	0.164
5	0.042	0.091	0.057	0.012	0.056	0.085	0.074	0.087	0.164
6	0.113	0.091	0.114	0.061	0.056	0.098	0.074	0.043	0.098
7	0.310	0.286	0.314	0.280	0.333	0.256	0.296	0.246	0.262
8	0.014	0.000	0.014	0.012	0.000	0.000	0.000	0.014	0.000
9	0.000	0.000	0.014	0.012	0.000	0.012	0.025	0.014	0.016

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



	1	2	3	4	5	6	7	8	9
1	0.132	0.096	0.123	0.149	0.096	0.088	0.123	0.114	0.079
2	0.110	0.132	0.143	0.132	0.110	0.121	0.088	0.077	0.088
3	0.000	0.167	0.111	0.167	0.167	0.111	0.111	0.111	0.056
4	0.088	0.109	0.036	0.139	0.117	0.161	0.139	0.139	0.073
5	0.062	0.146	0.083	0.021	0.083	0.146	0.125	0.125	0.208
6	0.145	0.127	0.145	0.091	0.073	0.145	0.109	0.055	0.109
7	0.115	0.115	0.115	0.120	0.126	0.110	0.126	0.089	0.084
8	0.250	0.000	0.250	0.250	0.000	0.000	0.000	0.250	0.000
9	0.000	0.000	0.143	0.143	0.000	0.143	0.286	0.143	0.143

3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in
 train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in 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?

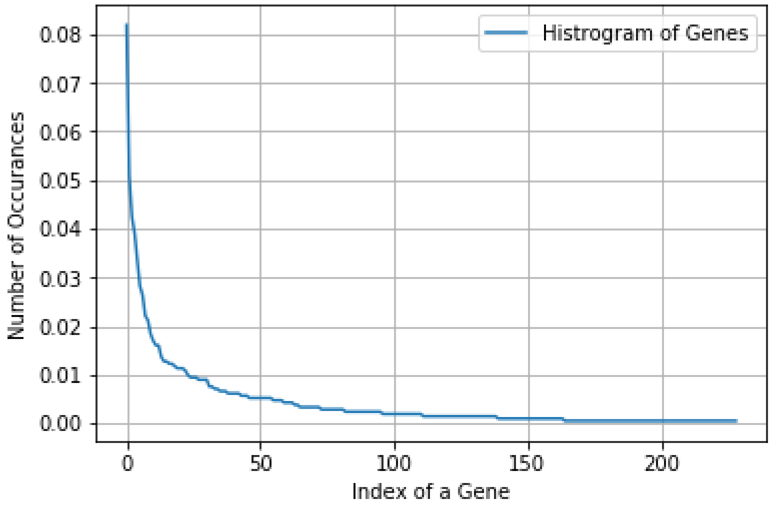
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 229
BRCA1     174
TP53      108
EGFR       90
BRCA2      83
PTEN       71
KIT        60
BRAF       56
ALK        47
ERBB2      45
PDGFRA     39
Name: Gene, dtype: int64
```

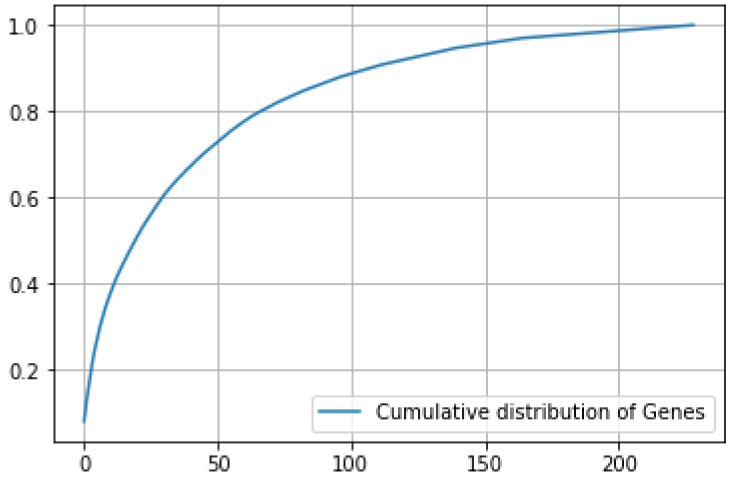
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

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

In [18]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", tra
in_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test
_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df
))
```

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

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = 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 [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
714     ERBB2
184      EGFR
2416    PTPRD
358     EP300
491      TP53
Name: Gene, dtype: object
```

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

Out[24]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt1',
 'flt3',
 'foxa1',
 'foxl2',
 'foxp1',
 'fubp1',
 'gata3',
 'gli1',
 'gna11',
 'gnas',
 'h3f3a',
 'hist1h1c',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'ikzf1',
 'il7r',
 'inpp4b',
 'jak1',
 'jak2',
 'jun',
 'kdm5a',
 'kdm5c',
 'kdr',
 'keap1',
 'kit',
 'klf4',
 'kmt2a',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats2',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm4',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'ncor1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad51b',
 'rad51c',
 'rad51d',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rhoa',
 'rit1',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'sdhc',
 'sf3b1',
 'shoc2',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'xpo1',
 'xrcc2',
 'yap1']
```

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding m
ethod. The shape of gene feature: (2124, 229)
```

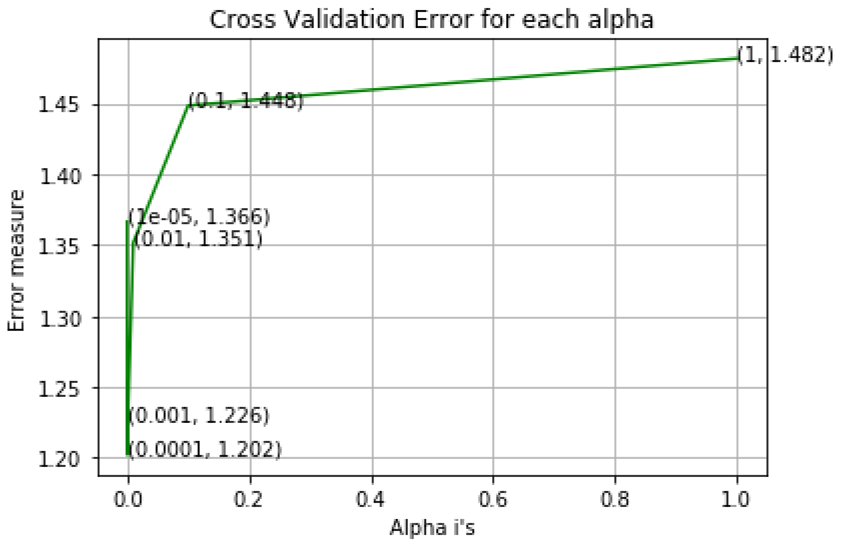
Q4. How good is this gene feature in predicting y_i?

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

In [26]:
```
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: 1.3664379973709315
For values of alpha =  0.0001 The log loss is: 1.2020958198964322
For values of alpha =  0.001 The log loss is: 1.2259686905568286
For values of alpha =  0.01 The log loss is: 1.3514145526383885
For values of alpha =  0.1 The log loss is: 1.448424907046388
For values of alpha =  1 The log loss is: 1.4816841487104513
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0339971702522932
For values of best alpha =  0.0001 The cross validation log loss is: 1.202095
8198964322
For values of best alpha =  0.0001 The test log loss is: 1.208616725476929
```

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

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

In [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", 
unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  229  gen
es in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
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?

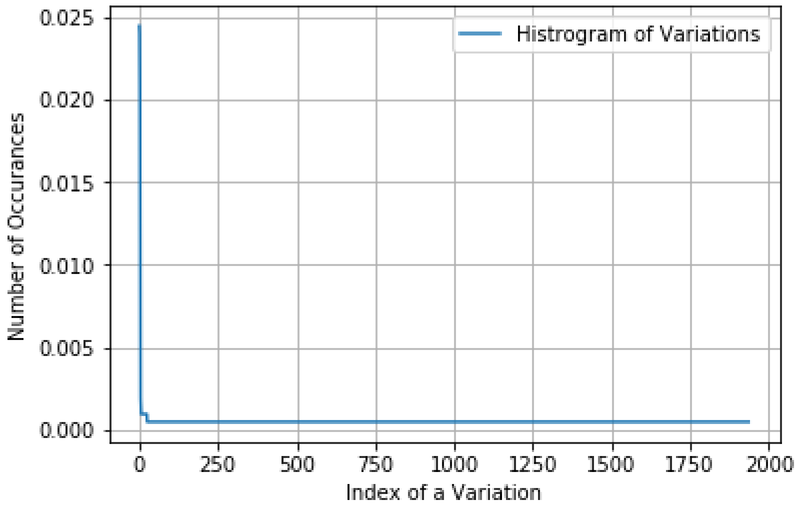
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1937
Truncating_Mutations    52
Amplification           51
Deletion                44
Fusions                 20
Overexpression           4
E17K                     3
Q61R                     2
V321M                    2
ETV6-NTRK3_Fusion        2
Y42C                     2
Name: Variation, dtype: int64
```

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

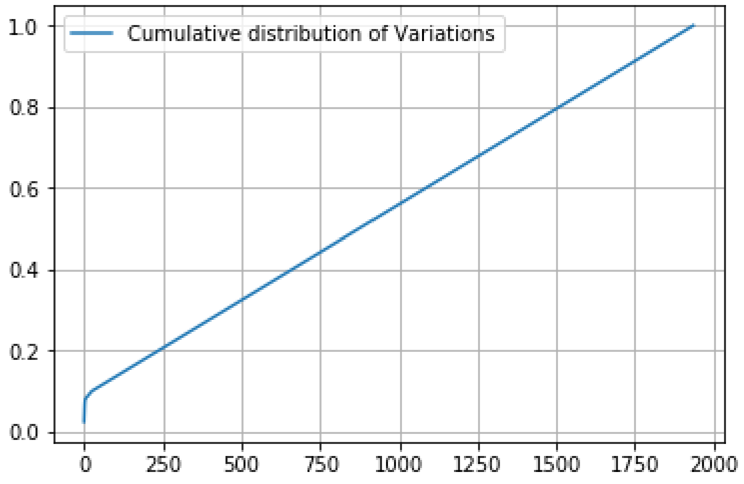
```
Ans: There are 1937 different categories of variations in the train data, and
they are distibuted as follows
```

In [30]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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

```
[0.02448211 0.04849341 0.06920904 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "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 [33]:
```
print("train_variation_feature_responseCoding is a converted feature using the
response coding method. The shape of Variation feature:", train_variation_feat
ure_responseCoding.shape)
```

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

In [34]:
```
# 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 [35]:
```
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)
```

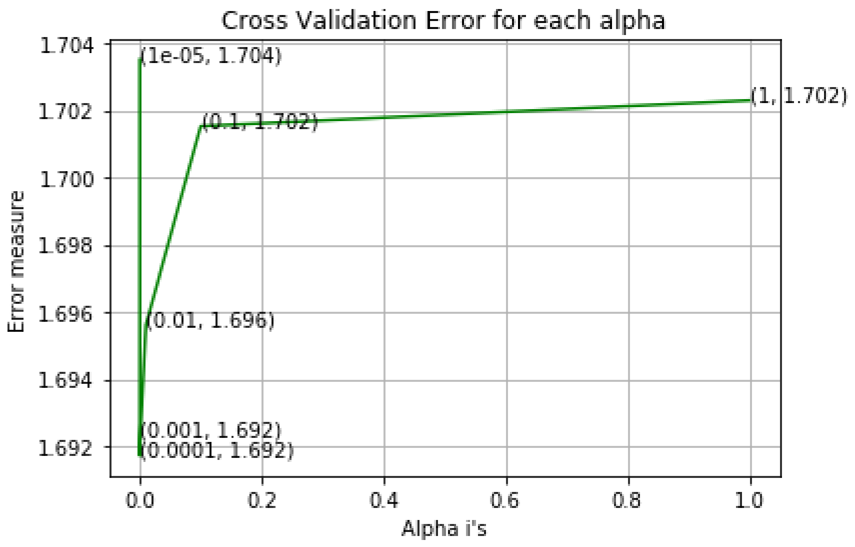
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1968)
```

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

Let's build a model just like the earlier!

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/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: 1.7035132240512456
For values of alpha =  0.0001 The log loss is: 1.6917156069879855
For values of alpha =  0.001 The log loss is: 1.6923502883408756
For values of alpha =  0.01 The log loss is: 1.6956152891943763
For values of alpha =  0.1 The log loss is: 1.7015261261862156
For values of alpha =  1 The log loss is: 1.702294309953088
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7598615462901389
For values of best alpha =  0.0001 The cross validation log loss is: 1.691715
6069879855
For values of best alpha =  0.0001 The test log loss is: 1.7084080142791527
```

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

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

In [37]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1937  genes in test and cross
validation data sets?
Ans
1. In test data 73 out of 665 : 10.977443609022556
2. In cross validation data 60 out of  532 : 11.278195488721805
```

3.2.3 Univariate Analysis on Text Feature

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

In [38]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_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 [40]:
```
'''# building a CountVectorizer with all the words that occured minimum 3 time
s in train data
text_vectorizer = CountVectorizer(min_df=3)
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_feature
s))'''
```

Out[40]:
```
'# building a CountVectorizer with all the words that occured minimum 3 times
in train data\ntext_vectorizer = CountVectorizer(min_df=3)\ntrain_text_featur
e_onehotCoding = text_vectorizer.fit_transform(train_df[\'TEXT\'])\n# getting
all the feature names (words)\ntrain_text_features= text_vectorizer.get_featu
re_names()\n\n# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every
row and returns (1*number of features) vector\ntrain_text_fea_counts = train_
text_feature_onehotCoding.sum(axis=0).A1\n\n# zip(list(text_features),text_fe
a_counts) will zip a word with its number of times it occured\ntext_fea_dict
= dict(zip(list(train_text_features),train_text_fea_counts))\n\n\nprint("Tota
l number of unique words in train data :", len(train_text_features))'
```

Tf-idf

In [41]:
```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(min_df=3,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 [42]:
```
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 [43]:
```
#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 [44]:
```
# 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 [45]:
```
# 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 [46]:
```
#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 [47]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({213.50540632752643: 1, 144.66434610192505: 1, 127.13961534939372: 1,
109.72121332045366: 1, 103.80124313279741: 1, 98.33430169087708: 1, 98.046062
60281415: 1, 98.03638993581221: 1, 96.48939182143378: 1, 91.4113928866851: 1,
87.25702238181873: 1, 77.80465598975134: 1, 76.01413286752086: 1, 75.53497985
565677: 1, 71.43607890850274: 1, 67.10259090255944: 1, 65.93243524743558: 1,
65.80055586023452: 1, 65.63128084579978: 1, 64.74335643133374: 1, 62.57014228
883806: 1, 61.68045718367326: 1, 57.96143563157932: 1, 57.54483090033584: 1,
56.46387117116276: 1, 55.31068274476658: 1, 53.8517587011591: 1, 53.538664612
35393: 1, 53.458340532998214: 1, 53.030159323848906: 1, 52.700225833336305:
1, 52.468011846774196: 1, 51.21739314562899: 1, 49.82788169339875: 1, 49.7126
4318118018: 1, 49.20090332087628: 1, 48.78555793321164: 1, 48.02928123121769:
1, 47.28812589344916: 1, 46.12917282869014: 1, 45.78179535738898: 1, 44.54667
325611311: 1, 44.011177846807406: 1, 43.61333783318754: 1, 43.43598945393916
5: 1, 40.876859108837046: 1, 40.66222800109604: 1, 40.58336063904948: 1, 39.9
5420021818157: 1, 38.43905183534458: 1, 38.164371932236534: 1, 37.92186812707
6586: 1, 36.86925777160881: 1, 36.8532861054109: 1, 36.74214772350739: 1, 36.
152838704942624: 1, 35.898482228253485: 1, 35.880914925477484: 1, 35.83992720
283042: 1, 35.22303786159549: 1, 35.11919005267137: 1, 34.911838446516214: 1,
34.791541367537334: 1, 34.275265526933: 1, 34.22170172968406: 1, 34.061387946
72053: 1, 34.02908287753224: 1, 33.95983586367009: 1, 33.53211977318552: 1, 3
3.48543526404831: 1, 32.62952238489561: 1, 32.426368483043504: 1, 32.18437272
4489506: 1, 32.08919489119906: 1, 32.07496898130141: 1, 31.778327319188968:
1, 30.61140114384626: 1, 30.508429654405937: 1, 30.469073516812298: 1, 30.433
011015428043: 1, 30.051217978590778: 1, 29.84615956027801: 1, 29.538810006722
326: 1, 29.49271636714874: 1, 29.201585199677723: 1, 29.181313104806094: 1, 2
8.95912825877566: 1, 28.938188010644872: 1, 28.79713959326275: 1, 28.77477618
9181182: 1, 28.282796837328846: 1, 28.112210109938474: 1, 27.958238021705576:
1, 27.909977727302632: 1, 27.560656201438103: 1, 27.303813657776118: 1, 27.20
5356891575107: 1, 27.145978991160757: 1, 27.135176260600264: 1, 27.0338221781
71807: 1, 26.993849123126655: 1, 26.961867223817727: 1, 26.71813341933111: 1,
26.60389646757246: 1, 26.51942568360349: 1, 26.397384869286313: 1, 26.2627191
0423334: 1, 26.237363340405974: 1, 26.12768452602363: 1, 26.000815895602482:
1, 25.894975998899177: 1, 25.857059645400298: 1, 25.78675246875417: 1, 25.723
80134314359: 1, 25.29371464469266: 1, 25.138394072474746: 1, 25.1360105689460
9: 1, 25.134212551702277: 1, 25.076589302484937: 1, 24.997888287558915: 1, 2
4.94264325440828: 1, 24.87295298771774: 1, 24.791610186395502: 1, 24.56655490
3351456: 1, 24.45183279867034: 1, 24.427225941117538: 1, 24.405185545401874:
1, 24.3591798343005: 1, 24.268929229540852: 1, 24.18253718666044: 1, 23.99623
6198885065: 1, 23.967208894831014: 1, 23.77809315289846: 1, 23.67010113277379
8: 1, 23.55040173124185: 1, 23.397481700777792: 1, 23.25506913381958: 1, 23.2
47722543555813: 1, 23.211877663941785: 1, 22.892318063126133: 1, 22.732990802
076735: 1, 22.498285919617203: 1, 22.469838055552597: 1, 22.340196868927055:
1, 22.292703964593628: 1, 22.11326236683598: 1, 22.060391810907074: 1, 22.048
13925034946: 1, 21.98725440617563: 1, 21.973788694871356: 1, 21.9482936505302
67: 1, 21.927421399197385: 1, 21.74202009503573: 1, 21.62196993246615: 1, 21.
489360798194404: 1, 21.413291088781303: 1, 21.293367377035818: 1, 21.28380108
8572638: 1, 21.095902925752387: 1, 21.03767488423232: 1, 20.961200465816933:
1, 20.86261778814362: 1, 20.79990240974338: 1, 20.7822571390031: 1, 20.756179
078843946: 1, 20.632840274781422: 1, 20.58703476484809: 1, 20.41937526402058:
1, 20.4159541530352: 1, 20.38383593893046: 1, 20.334591298585476: 1, 20.30123
6573402054: 1, 20.23816152637258: 1, 20.20684803885165: 1, 20.20135959259755:
1, 20.120293864308415: 1, 20.060277064706852: 1, 19.980952178421344: 1, 19.92
2649091545274: 1, 19.871817570779502: 1, 19.80343444115558: 1, 19.65781679742
6158: 1, 19.65070098143952: 1, 19.521349490613694: 1, 19.40763036478906: 1, 1
9.366771252445506: 1, 19.26140766666988: 1, 19.202887489241185: 1, 19.1366349
36408434: 1, 19.1174619457745: 1, 19.08595222109366: 1, 19.054850480751853:
1, 19.02640841922875: 1, 19.023518478666944: 1, 18.975485483372143: 1, 18.951
```

```
77058613527: 1, 18.949158191875444: 1, 18.929690824802133: 1, 18.880707818685
79: 1, 18.83527376455919: 1, 18.817596037902913: 1, 18.736170660480678: 1, 1
8.72943468832591: 1, 18.60330662504377: 1, 18.56530736837092: 1, 18.494584208
374203: 1, 18.462114923738838: 1, 18.373831139661046: 1, 18.363645759914046:
1, 18.294634393869444: 1, 18.275960689159643: 1, 18.275714198791743: 1, 18.21
9190069155385: 1, 18.133645846732254: 1, 18.1314948619986: 1, 18.120248090762
46: 1, 18.08296557639709: 1, 18.077059734389326: 1, 18.068044549991015: 1, 1
8.061402325341966: 1, 18.025930471349994: 1, 17.98654058660288: 1, 17.9855800
13827974: 1, 17.97684001981463: 1, 17.93961922727918: 1, 17.93466397082985:
1, 17.88888006388902: 1, 17.85827194116596: 1, 17.839848905727028: 1, 17.8336
2543599384: 1, 17.802224644586104: 1, 17.782157332975252: 1, 17.7706560075872
7: 1, 17.74170027810453: 1, 17.659594976828963: 1, 17.529495126655043: 1, 17.
488727610252038: 1, 17.438632989059734: 1, 17.333276500217163: 1, 17.33117739
0013938: 1, 17.260974584382154: 1, 17.17631288560101: 1, 17.16930445144521:
1, 17.15282190834002: 1, 17.136456275380432: 1, 17.129150380114467: 1, 17.107
407291842062: 1, 17.007208238731813: 1, 16.99981899146388: 1, 16.992948120369
95: 1, 16.953512756419013: 1, 16.927411950587903: 1, 16.91059507829113: 1, 1
6.884012066251255: 1, 16.868546216661585: 1, 16.85668384827971: 1, 16.8483796
41516882: 1, 16.81054652697738: 1, 16.78942692371062: 1, 16.68124631456322:
1, 16.66328127305932: 1, 16.65455071135381: 1, 16.640068933427195: 1, 16.6168
5290511487: 1, 16.519075793963808: 1, 16.416102245220035: 1, 16.4155531623509
46: 1, 16.36150686226658: 1, 16.28598852056978: 1, 16.279119510066586: 1, 16.
18896513361321: 1, 16.187069252448556: 1, 16.15878327213825: 1, 16.1505147838
57002: 1, 16.095984948919313: 1, 16.093453932033373: 1, 16.08154405121483: 1,
16.07567417675615: 1, 16.050960034709853: 1, 16.045118772008525: 1, 15.987238
864317828: 1, 15.954987139374737: 1, 15.940346835855602: 1, 15.91589058884618
2: 1, 15.902700558856708: 1, 15.865046249757363: 1, 15.863669122917734: 1, 1
5.739313198890661: 1, 15.709572544499938: 1, 15.665361003448773: 1, 15.638994
426847118: 1, 15.527508010604128: 1, 15.514452921405248: 1, 15.46945070117919
8: 1, 15.39856278917956: 1, 15.396853628128893: 1, 15.30824438118602: 1, 15.2
85394356913695: 1, 15.252671234033787: 1, 15.196659904533954: 1, 15.191318652
884114: 1, 15.186747882205434: 1, 15.163760823174027: 1, 15.15732171877824:
1, 15.155825608198445: 1, 15.112838340776374: 1, 15.068048666959001: 1, 15.04
1531217747579: 1, 15.036494775265458: 1, 15.035987367066841: 1, 15.0226386209
61178: 1, 15.017168310641123: 1, 14.992909689813414: 1, 14.973403978344225:
1, 14.947053331235173: 1, 14.919921579059956: 1, 14.865129914489641: 1, 14.86
5043832353297: 1, 14.797035112631807: 1, 14.747444125083987: 1, 14.7277276383
98358: 1, 14.544616519592877: 1, 14.543218663993853: 1, 14.540338702772456:
1, 14.50107037701381: 1, 14.496377659036838: 1, 14.487413212191752: 1, 14.482
83120548993: 1, 14.420730250615543: 1, 14.387073910618419: 1, 14.381530596045
01: 1, 14.371022262793199: 1, 14.259834010147847: 1, 14.251846561409623: 1, 1
4.214444601802702: 1, 14.185993057689148: 1, 14.143247692131123: 1, 14.139859
344917593: 1, 14.134747487824212: 1, 14.120434259189464: 1, 14.10027283795279
9: 1, 14.070270663744427: 1, 13.933290498805304: 1, 13.915051978017441: 1, 1
3.909937603725439: 1, 13.909476157910362: 1, 13.818488766763846: 1, 13.806006
849873993: 1, 13.80159424332526: 1, 13.763875188654401: 1, 13.75872405538327
4: 1, 13.744197873517935: 1, 13.743018976356495: 1, 13.732448722147254: 1, 1
3.727049936503567: 1, 13.71896823152375: 1, 13.70647194861068: 1, 13.65129565
5809118: 1, 13.630556122006482: 1, 13.622290370070056: 1, 13.613912591185333:
1, 13.611834135024123: 1, 13.508041347539855: 1, 13.45217197537693: 1, 13.428
506816807081: 1, 13.421276605937381: 1, 13.419115426304941: 1, 13.40270801073
9175: 1, 13.400381830416999: 1, 13.356771599487816: 1, 13.340921849055519: 1,
13.334191577812216: 1, 13.330322921603969: 1, 13.300126256359691: 1, 13.24960
919392708: 1, 13.230999178090391: 1, 13.227162679487439: 1, 13.20011418152093
6: 1, 13.198528183930625: 1, 13.144270305142252: 1, 13.104946241241272: 1, 1
3.084105875374698: 1, 13.076855009839704: 1, 13.06008420406865: 1, 13.0521903
5977709: 1, 13.002665134375766: 1, 12.966213392811836: 1, 12.96370258318279:
```

```
1, 12.951091826709717: 1, 12.95034744300754: 1, 12.943984319221569: 1, 12.943
388431498976: 1, 12.940169307604322: 1, 12.933277825607822: 1, 12.91616765856
7565: 1, 12.913903785098164: 1, 12.880922677606176: 1, 12.844263360192835: 1,
12.83504050006937: 1, 12.82826874632347: 1, 12.825711625217913: 1, 12.8080234
60685948: 1, 12.804283258502718: 1, 12.799729053219016: 1, 12.78341988829715
9: 1, 12.772298781455671: 1, 12.755976754180459: 1, 12.7536865159767: 1, 12.7
11538101949516: 1, 12.691022729125677: 1, 12.68604813511069: 1, 12.6552428440
20473: 1, 12.639197567418321: 1, 12.635195908424565: 1, 12.557013202412506:
1, 12.527189070474885: 1, 12.52134084370951: 1, 12.514652334983776: 1, 12.499
908759914195: 1, 12.493756074813842: 1, 12.491239400414257: 1, 12.43246408476
6991: 1, 12.426478845252285: 1, 12.398362912245249: 1, 12.354224873276662: 1,
12.348756354195393: 1, 12.345389887735115: 1, 12.325466584834267: 1, 12.32219
1501533046: 1, 12.31712458860547: 1, 12.302005915419645: 1, 12.23980951826382
6: 1, 12.217610449569506: 1, 12.207128726755899: 1, 12.201304125162263: 1, 1
2.185618894445803: 1, 12.152515586529121: 1, 12.15136543132078: 1, 12.1361890
13461761: 1, 12.122630968655567: 1, 12.09427995505224: 1, 12.085162697114285:
1, 12.081264009463546: 1, 12.07898852123432: 1, 12.061815652350402: 1, 12.050
59595412901: 1, 12.044388504979974: 1, 12.03501396997731: 1, 12.0077194305052
9: 1, 11.969001956509192: 1, 11.935142400691674: 1, 11.932221088176187: 1, 1
1.928716816939815: 1, 11.920402334607694: 1, 11.904638382676435: 1, 11.860987
389916934: 1, 11.843121098782795: 1, 11.840685196509988: 1, 11.83383643478939
4: 1, 11.805081885630695: 1, 11.789814466454908: 1, 11.783823340451063: 1, 1
1.760352361411817: 1, 11.754660420000752: 1, 11.739935596329826: 1, 11.714371
455382896: 1, 11.700004501095425: 1, 11.696891222422439: 1, 11.67387726709112
7: 1, 11.660678679731129: 1, 11.65326765785526: 1, 11.652065234730284: 1, 11.
649739742889265: 1, 11.632423377808308: 1, 11.627992290833285: 1, 11.61632221
7502853: 1, 11.608295756640961: 1, 11.608108973872325: 1, 11.60441675696204:
1, 11.572346037013855: 1, 11.562265395860697: 1, 11.53095646152823: 1, 11.500
557657461796: 1, 11.493406850051985: 1, 11.474130414664737: 1, 11.44748431771
5613: 1, 11.429860629836863: 1, 11.426489066926425: 1, 11.415209528550406: 1,
11.403245289703166: 1, 11.394089391802847: 1, 11.38407453019985: 1, 11.374735
408573697: 1, 11.33826280008547: 1, 11.324677625194148: 1, 11.31305163349451
5: 1, 11.294122958608058: 1, 11.280092020706588: 1, 11.270956834549594: 1, 1
1.270498136486387: 1, 11.267065777926279: 1, 11.24695217012731: 1, 11.2415899
3874425: 1, 11.186010984609405: 1, 11.184422236458543: 1, 11.167109727881007:
1, 11.165990110223792: 1, 11.165769116392271: 1, 11.147232945904024: 1, 11.14
5412850054868: 1, 11.126065966035886: 1, 11.11787192083808: 1, 11.08880334919
4947: 1, 11.070771721785615: 1, 11.035528917108127: 1, 11.033460273774065: 1,
11.030397630822527: 1, 11.019806448945294: 1, 11.012988573071942: 1, 10.98198
8011173033: 1, 10.975072656204958: 1, 10.972888839158918: 1, 10.9674709350316
28: 1, 10.91507773491326: 1, 10.914518036007793: 1, 10.911014659597706: 1, 1
0.895908909975887: 1, 10.852093161224497: 1, 10.84814972011811: 1, 10.8129584
75596606: 1, 10.809661044057274: 1, 10.801668858948979: 1, 10.79789205888433:
1, 10.793873094384047: 1, 10.785925801635832: 1, 10.782296395135685: 1, 10.76
8707121224756: 1, 10.7192797285172: 1, 10.688341356937109: 1, 10.663053579824
226: 1, 10.654874532407842: 1, 10.64555425173855: 1, 10.638131928594438: 1, 1
0.634063525226804: 1, 10.633483018214282: 1, 10.609883171997504: 1, 10.593055
285738291: 1, 10.580305175086515: 1, 10.542152787539989: 1, 10.53871254358797
1: 1, 10.535874809015857: 1, 10.535330983587997: 1, 10.534036121908857: 1, 1
0.530417519538405: 1, 10.52539907224785: 1, 10.513680020137134: 1, 10.4817871
76393214: 1, 10.473419869956482: 1, 10.454510579293157: 1, 10.45245782056521
3: 1, 10.444924501971675: 1, 10.437778933052993: 1, 10.432694994439197: 1, 1
0.431305381676355: 1, 10.399182854230121: 1, 10.378867110802492: 1, 10.332709
510388682: 1, 10.321179805738984: 1, 10.319142548990412: 1, 10.31867783356020
1: 1, 10.315839291496372: 1, 10.315708128232647: 1, 10.311934163149125: 1, 1
0.306590677292569: 1, 10.263724659170768: 1, 10.22229998474546: 1, 10.2197815
35081772: 1, 10.219336524020619: 1, 10.214408433740653: 1, 10.20617350093083
```

```
3: 1, 10.205713678507323: 1, 10.158187364047931: 1, 10.152991207180161: 1, 1
0.146323684537743: 1, 10.143586225718327: 1, 10.113160116584533: 1, 10.102579
597823011: 1, 10.076090797351283: 1, 10.069147086305609: 1, 10.0450815514792
1: 1, 10.036544643773697: 1, 10.015172807523015: 1, 9.964975457414367: 1, 9.9
64404657904202: 1, 9.948432270868619: 1, 9.932157896183703: 1, 9.911307232678
215: 1, 9.904625524029191: 1, 9.88853093305104: 1, 9.88359488422432: 1, 9.879
929697848485: 1, 9.859284998922377: 1, 9.853734722641224: 1, 9.82776368549162
7: 1, 9.820410399340746: 1, 9.817483299802765: 1, 9.799820124063718: 1, 9.794
06809117362: 1, 9.784354504188268: 1, 9.780422296170848: 1, 9.76883783844500
9: 1, 9.768501163892612: 1, 9.702655755802503: 1, 9.701968649278479: 1, 9.691
824542456986: 1, 9.659463554564468: 1, 9.653380265841689: 1, 9.64559490902732
9: 1, 9.633102102644054: 1, 9.622799959430186: 1, 9.607942682244126: 1, 9.584
26536610471: 1, 9.580497215948458: 1, 9.555140744554146: 1, 9.55330701243153
3: 1, 9.533369525407412: 1, 9.48528087745737: 1, 9.479087909750731: 1, 9.4707
21200799433: 1, 9.45060861617804: 1, 9.44327612140301: 1, 9.433606026834456:
1, 9.418139899121595: 1, 9.409854157969198: 1, 9.387041764284374: 1, 9.383254
361910975: 1, 9.368733532106113: 1, 9.368232959315646: 1, 9.363465744989204:
1, 9.35885822419332: 1, 9.35868590869994: 1, 9.35410810252073: 1, 9.348097235
76474: 1, 9.347105644161596: 1, 9.34443385367533: 1, 9.334040806271721: 1, 9.
331204385844995: 1, 9.323847221223033: 1, 9.321709160595837: 1, 9.31408969180
1188: 1, 9.301721563492901: 1, 9.295995533089574: 1, 9.288917867183958: 1, 9.
287785107361874: 1, 9.275367562166881: 1, 9.25463188516873: 1, 9.250748347843
933: 1, 9.241497633778271: 1, 9.239296323372166: 1, 9.236753470281808: 1, 9.2
2312060085896: 1, 9.221480440236755: 1, 9.220095086437569: 1, 9.1713633326731
3: 1, 9.165567364119003: 1, 9.164451215649397: 1, 9.155632352200485: 1, 9.145
598246637334: 1, 9.135712647481194: 1, 9.122931278994264: 1, 9.10401327150666
3: 1, 9.102276687642862: 1, 9.101876871742924: 1, 9.093404197732763: 1, 9.093
125860755197: 1, 9.083641531342682: 1, 9.055428523038245: 1, 9.05417215362955
7: 1, 9.04845412279826: 1, 9.03647405775735: 1, 9.034832670153705: 1, 9.02503
6056191178: 1, 9.020253010808716: 1, 9.01840935698188: 1, 9.004079793936791:
1, 8.999996454253374: 1, 8.995313105226426: 1, 8.991121285028694: 1, 8.989304
120156408: 1, 8.975875882910476: 1, 8.969688246793588: 1, 8.949341720677095:
1, 8.947042856828094: 1, 8.936825349150043: 1, 8.936489664885608: 1, 8.926751
117742759: 1, 8.908890096005226: 1, 8.905940049334061: 1, 8.902477596024962:
1, 8.894834951696623: 1, 8.889055881525499: 1, 8.886536781615527: 1, 8.885216
299279742: 1, 8.863669942715646: 1, 8.842154441536033: 1, 8.840229544672907:
1, 8.829129228368664: 1, 8.827315380913758: 1, 8.802481823408632: 1, 8.786419
709161617: 1, 8.754053958780345: 1, 8.75109556216875: 1, 8.744102709348473:
1, 8.718800302138682: 1, 8.717455519971066: 1, 8.706793995952099: 1, 8.705687
79924043: 1, 8.701268296801397: 1, 8.696927576457659: 1, 8.695618530355691:
1, 8.688782714483544: 1, 8.688749284015616: 1, 8.680618379837995: 1, 8.679079
43513345: 1, 8.674198094996632: 1, 8.66114604717396: 1, 8.657684040406238: 1,
8.655089171052598: 1, 8.6481262798021: 1, 8.646550510623335: 1, 8.64645370298
9795: 1, 8.645529218546317: 1, 8.642614837252706: 1, 8.638051522687283: 1, 8.
636037656128153: 1, 8.612655788804545: 1, 8.611038262825401: 1, 8.60836626134
5316: 1, 8.607666271487172: 1, 8.601795659394703: 1, 8.584626623121675: 1, 8.
57798377493331: 1, 8.573144783977492: 1, 8.573107020750085: 1, 8.564286208403
821: 1, 8.548792841045662: 1, 8.548143042652635: 1, 8.542100794643938: 1, 8.5
32508972937846: 1, 8.501385702209836: 1, 8.500866953715663: 1, 8.500534748249
107: 1, 8.492246577404906: 1, 8.491897107454534: 1, 8.488770994109563: 1, 8.4
82830942906755: 1, 8.479386898906759: 1, 8.477265485789962: 1, 8.448699133131
774: 1, 8.448002375526224: 1, 8.437585634784552: 1, 8.433938343411517: 1, 8.4
3069071936747: 1, 8.423286464941592: 1, 8.420521821742136: 1, 8.4092317313097
49: 1, 8.406785878216539: 1, 8.406592659373652: 1, 8.394328111182169: 1, 8.39
4103663704199: 1, 8.356593574330697: 1, 8.346162059287874: 1, 8.3385800536942
5: 1, 8.337318927384002: 1, 8.33542949745582: 1, 8.33495596315572: 1, 8.32965
4523116764: 1, 8.3279714584119: 1, 8.327714884760125: 1, 8.312241542768962:
```

```
1, 8.304041915307065: 1, 8.299633273063968: 1, 8.292035810414012: 1, 8.281439
132543662: 1, 8.271848933251166: 1, 8.269304178337437: 1, 8.267230507545523:
1, 8.266981535641115: 1, 8.26491331138955: 1, 8.258171173042717: 1, 8.2554066
36664196: 1, 8.252569458209633: 1, 8.24840254609648: 1, 8.246979484598384: 1,
8.23136527424105: 1, 8.22538524565359: 1, 8.211508594406984: 1, 8.20874108202
9784: 1, 8.201805844542323: 1, 8.193042850484122: 1, 8.172040462789026: 1, 8.
168310618485789: 1, 8.165202699656248: 1, 8.157138362862565: 1, 8.1478225812
07881: 1, 8.141038337575633: 1, 8.140533134197486: 1, 8.133256361457532: 1, 8.
131852150176329: 1, 8.13063912153468: 1, 8.128260331162627: 1, 8.106356531850
555: 1, 8.096944181198745: 1, 8.095324429671406: 1, 8.093342305708504: 1, 8.0
83080838177796: 1, 8.079327706188472: 1, 8.076039685810857: 1, 8.071073156065
738: 1, 8.046776134368292: 1, 8.046369929431599: 1, 8.045782451122108: 1, 8.0
41969951443647: 1, 8.041868704297753: 1, 8.02920610861023: 1, 8.0253390333383
07: 1, 8.021215545703143: 1, 8.020770995306506: 1, 8.020434507520738: 1, 8.00
7358143870446: 1, 7.997974182734808: 1, 7.981406840067845: 1, 7.9781235125616
51: 1, 7.974371191264622 5: 1, 7.970603795719547: 1, 7.963499979333802: 1, 7.9
613597761953985: 1, 7.949889100146671: 1, 7.946501079569211: 1, 7.94056292476
3497: 1, 7.927196846829321: 1, 7.918194695825289: 1, 7.91617061924758: 1, 7.9
10330752157482: 1, 7.900636938389049: 1, 7.891751897087624: 1, 7.888795475172
691: 1, 7.874031095466162: 1, 7.870474403032094: 1, 7.8529251593362925: 1, 7.
837708809160847: 1, 7.827446127745114: 1, 7.823737468360175: 1, 7.81480845038
65345: 1, 7.803517484657986: 1, 7.795507492361665: 1, 7.787632685854584: 1,
7.7550747092126056: 1, 7.745457319476502: 1, 7.736204670635211: 1, 7.73409855
2554119: 1, 7.722547140914808: 1, 7.722261837186299: 1, 7.718700502099731: 1,
7.716791371129894: 1, 7.7111333074475: 1, 7.692533308348056: 1, 7.67269773779
49135: 1, 7.6711659285591125: 1, 7.6696205488441045: 1, 7.666918080740963: 1,
7.66109624204413: 1, 7.658788674810165: 1, 7.652700914829496: 1, 7.6521769482
36934: 1, 7.638277348052469: 1, 7.636872066464154: 1, 7.625128640241353: 1,
7.625003733768375: 1, 7.6221840941447185: 1, 7.617685328584226: 1, 7.61536535
8981024: 1, 7.5969185507964525: 1, 7.589179718540664: 1, 7.584284624970813:
1, 7.580602653882905: 1, 7.578337389199575: 1, 7.577550442470754: 1, 7.576307
5688611865: 1, 7.566022076048096: 1, 7.556425827054552: 1, 7.553444175456246:
1, 7.540930054068974: 1, 7.53603700424909: 1, 7.521760006615677: 1, 7.5213406
66555954: 1, 7.511208804416028: 1, 7.498613171455177: 1, 7.492115310507773:
1, 7.489574876741051: 1, 7.4881202554077335: 1, 7.485978982507506: 1, 7.48387
26492335885: 1, 7.482394868545576: 1, 7.478641708676197: 1, 7.47641506729559
9: 1, 7.470841506549069: 1, 7.46298585473214: 1, 7.455975901658415: 1, 7.4545
52000524596: 1, 7.448754171881558: 1, 7.43379920240995: 1, 7.423879648584374:
1, 7.414402580479673: 1, 7.403787077752996: 1, 7.400075304791188: 1, 7.397776
639557775: 1, 7.392513924898549: 1, 7.389422118677216: 1, 7.377247930695634:
1, 7.376108461034538: 1, 7.373228262655965: 1, 7.3699251526161325: 1, 7.36942
5283129178: 1, 7.366663708019553: 1, 7.3655035961694315: 1, 7.365097983204071
5: 1, 7.359419030540826: 1, 7.356381496479656: 1, 7.354623676213033: 1, 7.348
029817745642: 1, 7.343952897502078: 1, 7.327552156565308: 1, 7.32588919725705
5: 1, 7.324551720096353: 1, 7.32245119976339: 1, 7.318492004167422: 1, 7.3131
72225048971: 1, 7.306596693200478: 1, 7.305092632345298: 1, 7.30355900047219
8: 1, 7.281869519616849: 1, 7.280874392896142: 1, 7.276124224371676: 1, 7.271
945304120238: 1, 7.259572611267601: 1, 7.252589010272433: 1, 7.24486609359834
9: 1, 7.2266317134531155: 1, 7.2252771036701215: 1, 7.221790137816581: 1, 7.2
19293572449749: 1, 7.218543862159028: 1, 7.210139653706805: 1, 7.209705728465
19: 1, 7.193541009335871: 1, 7.185547524547693: 1, 7.1852520668533195: 1, 7.1
61172613411562: 1, 7.145028271871149: 1, 7.1417679372113385: 1, 7.14118881252
85525: 1, 7.130181606452219: 1, 7.128905506718712: 1, 7.120653807978591: 1,
7.119230472166616: 1, 7.115846005020543: 1, 7.115640595811791: 1, 7.110671799
308889: 1, 7.109464761819278: 1, 7.100521383260864: 1, 7.097759390386562: 1,
7.0969305319693525: 1, 7.092887578143352: 1, 7.092541762361351: 1, 7.07138689
9559899: 1, 7.06262280553467: 1, 7.0564468190339715: 1, 7.048543525446029: 1,
7.045789423431142: 1, 7.036620615669567: 1, 7.033133486045957: 1, 7.025175799
887949: 1, 7.021667209758397: 1, 7.015338263107645: 1, 7.007336042346394: 1,
7.0071930130784015: 1, 7.000700855323475: 1, 6.998558645363004: 1, 6.99816033
985288: 1, 6.993896867249873: 1, 6.98744998187031: 1, 6.9706124467921535: 1,
6.967703685354518: 1, 6.964020641048291: 1, 6.958134723760886: 1, 6.955676159
260168: 1, 6.953461062731179: 1, 6.942517812290064: 1, 6.94208702113464: 1,
6.9385266791167135: 1, 6.9335074232347225: 1, 6.932032626246076: 1, 6.9190579
04767175: 1, 6.91713890440427: 1, 6.9145382596780784: 1, 6.900677481454298:
1, 6.896670488547684: 1, 6.893826665455915: 1, 6.893779108022015: 1, 6.888113
182418272: 1, 6.884023810381861: 1, 6.877405685717337: 1, 6.868252686995527:
1, 6.8677393850679795: 1, 6.864769678268428: 1, 6.857809163025481: 1, 6.84737
07839946085: 1, 6.846848666303932: 1, 6.8403851044619: 1, 6.836487638962093:
1, 6.833399481889513: 1, 6.825432143896479: 1, 6.822552925695284: 1, 6.821822
207968425: 1, 6.82041030448553: 1, 6.819126581056519: 1, 6.818113230050367:
1, 6.810868623977844: 1, 6.810860332324122: 1, 6.806569930562095: 1, 6.798397
448160665: 1, 6.794078961684939: 1, 6.793684622861269: 1, 6.791862776468567:
1, 6.7887467066369185: 1, 6.785894716274481: 1, 6.780726324069157: 1, 6.77989
5727349448: 1, 6.773427236100417: 1, 6.769571995994475: 1, 6.76359584633258:
1, 6.757631187980941: 1, 6.757111899360696: 1, 6.7409935262912954: 1, 6.73899
8890092965: 1, 6.734691123381208: 1, 6.7288715729366135: 1, 6.72677886592095
4: 1, 6.724626367074387: 1, 6.720886489578485: 1, 6.716027444097944: 1, 6.709
072188358768: 1, 6.708871732891857: 1, 6.704873427072738: 1, 6.69849004675444
7: 1, 6.695421202849242: 1, 6.693044765444934: 1, 6.67414730564863: 1, 6.6678
60618704658: 1, 6.666742873466017: 1, 6.665895513566176: 1, 6.65153768056315
4: 1, 6.6498726891302065: 1, 6.648353996180989: 1, 6.647996142036899: 1, 6.63
8941871928873: 1, 6.628737941917795: 1, 6.621737985969746: 1, 6.621038028920
853: 1, 6.613646976773809: 1, 6.610626727489323: 1, 6.6099271532852955: 1, 6.6
0908735500405: 1, 6.607251628531886: 1, 6.60179106674264: 1, 6.60080475086214
9: 1, 6.594689800981405: 1, 6.5887502924955195: 1, 6.585204224622141: 1, 6.58
4372241999299: 1, 6.583635123599614: 1, 6.579435093026637: 1, 6.5774183105211
03: 1, 6.576045562692866: 1, 6.574066781415687: 1, 6.564333607872583: 1, 6.56
3384681841839: 1, 6.552695197499499: 1, 6.546411546299659: 1, 6.5370953306317
3: 1, 6.535271949594034: 1, 6.530986523651377: 1, 6.523307904911596: 1, 6.515
614819439843: 1, 6.515533837547603: 1, 6.511893958110907: 1, 6.49649699920934
7: 1, 6.487266064848492: 1, 6.479737289172679: 1, 6.4790883477632555: 1, 6.47
8550557140692: 1, 6.477906116396711: 1, 6.466507005347804: 1, 6.4582760652313
36: 1, 6.45813962304098: 1, 6.452614597020427: 1, 6.4507074768793835: 1, 6.44
5984003709886: 1, 6.441788366845089: 1, 6.439974854512626: 1, 6.4290808135617
74: 1, 6.426700366244763: 1, 6.414524484362581: 1, 6.412962579531014: 1, 6.40
24575884854915: 1, 6.394970658915292: 1, 6.385904714797236: 1, 6.383165897604
417: 1, 6.372729148104161: 1, 6.370272792568022: 1, 6.370160002019849: 1, 6.3
661133719715925: 1, 6.364562118812718: 1, 6.362500638833644: 1, 6.36102516868
7198: 1, 6.360173432627908: 1, 6.3590464717555735: 1, 6.355745821357282: 1,
6.351916279999195: 1, 6.349301101968864: 1, 6.348611463422629: 1, 6.330237303
696887: 1, 6.3283110961873295: 1, 6.325176747424083: 1, 6.3228074779298: 1,
6.318294059008048: 1, 6.316931646296243: 1, 6.316466200887188: 1, 6.309022193
946158: 1, 6.2960886168275945: 1, 6.293331317229959: 1, 6.289516582638485: 1,
6.282277655452247: 1, 6.281862055278661: 1, 6.277315813568226: 1, 6.272756673
677417: 1, 6.271765491194507: 1, 6.270143276688261: 1, 6.262635882289919: 1,
6.25766227725444: 1, 6.255965962992718: 1, 6.254977441160161: 1, 6.2479407164
45909: 1, 6.246164354504947: 1, 6.246144679545504: 1, 6.240114686468282: 1,
6.238337203529532: 1, 6.2353243136372605: 1, 6.2351202361350495: 1, 6.2351019
56553363: 1, 6.232714853784338: 1, 6.221341425594107: 1, 6.211537060164003:
1, 6.209797595392051: 1, 6.206824306853961: 1, 6.206622557584366: 1, 6.200570
2984598585: 1, 6.1815455515428495: 1, 6.180249734931595: 1, 6.17767897463304
9: 1, 6.173936968970905: 1, 6.172808468169935: 1, 6.161617864042027: 1, 6.159
370482311454: 1, 6.145310096823756: 1, 6.140877117966744: 1, 6.13222141124551
```

```
9: 1, 6.1185991317535: 1, 6.116090798442026: 1, 6.113489930779181: 1, 6.11127
0892006612: 1, 6.091723742342908: 1, 6.091481049830946: 1, 6.091038525977518:
1, 6.081357871636875: 1, 6.073069938035405: 1, 6.072940758131793: 1, 6.070752
165516656: 1, 6.067974468395012: 1, 6.061753587965877: 1, 6.055672252060924:
1, 6.053949369561778: 1, 6.050691492967972: 1, 6.047536714856033: 1, 6.046977
672433087: 1, 6.04647752305207: 1, 6.030309391445247: 1, 6.0256320831886665:
1, 6.022021976057387: 1, 6.020752936608074: 1, 6.020574916776129: 1, 6.016687
46266157: 1, 6.015900716016651: 1, 6.010008826442721: 1, 6.006846154017154:
1, 6.002045221023901: 1, 5.998578477134552: 1, 5.996270061287954: 1, 5.989844
780769987: 1, 5.989149134782097: 1, 5.984513690411787: 1, 5.975689420382713:
1, 5.973665858111171: 1, 5.967336249159517: 1, 5.966077518448075: 1, 5.961511
052679201: 1, 5.960755047733758: 1, 5.956925348967431: 1, 5.943719120665964:
1, 5.936639238951499: 1, 5.936412640483884: 1, 5.934491223267961: 1, 5.932016
035544567: 1, 5.929859979485467: 1, 5.92132323190883: 1, 5.921176237477271:
1, 5.917190744780187: 1, 5.91111733958058: 1, 5.889403853240221: 1, 5.8881831
29506238: 1, 5.885727809536202: 1, 5.883232362419421: 1, 5.880097848227378:
1, 5.878683610683846: 1, 5.877024537219541: 1, 5.875871042303453: 1, 5.872205
0925519165: 1, 5.865649912642426: 1, 5.865267954008405: 1, 5.864226856109735:
1, 5.857604674168458: 1, 5.853670224252783: 1, 5.852577350649581: 1, 5.852483
805479356: 1, 5.846897346495717: 1, 5.842262248882525: 1, 5.840445357566831:
1, 5.8390463216191675: 1, 5.838996797081014: 1, 5.836454543049697: 1, 5.83269
2696885406: 1, 5.830843697193816: 1, 5.827925426943021: 1, 5.825940149902711:
1, 5.814587595983717: 1, 5.801876313893932: 1, 5.797606087289381: 1, 5.797435
764873212: 1, 5.792091681544821: 1, 5.790166986027261: 1, 5.786721272803226:
1, 5.763720975357854: 1, 5.763200812117891: 1, 5.755377081821163: 1, 5.753532
684842774: 1, 5.747385262739145: 1, 5.742803912458045: 1, 5.7347239843238125:
1, 5.732988131389828: 1, 5.7100774679739486: 1, 5.709649825360367: 1, 5.70864
49595969855: 1, 5.704644489074561: 1, 5.699053025693743: 1, 5.68976957856744
5: 1, 5.689525441215926: 1, 5.686949587359374: 1, 5.683879175509451: 1, 5.677
154716005005: 1, 5.671198581154176: 1, 5.671137655521031: 1, 5.67105452619449
4: 1, 5.665258781867667: 1, 5.659377861294531: 1, 5.658442361584858: 1, 5.653
5630296522825: 1, 5.645272036270859: 1, 5.644463039996048: 1, 5.6340371493537
93: 1, 5.631406240025287: 1, 5.627682748322202: 1, 5.6182018377334595: 1, 5.6
13549723914717: 1, 5.610586865708517: 1, 5.607902383471384: 1, 5.606929351216
953: 1, 5.606480760425612: 1, 5.606100866197957: 1, 5.594773953785431: 1, 5.5
92949549125477: 1, 5.592265955236032: 1, 5.5893751297585315: 1, 5.58482554941
44725: 1, 5.579491138139806: 1, 5.579435450777394: 1, 5.577197179271363: 1,
5.576651917579601: 1, 5.575309367994053: 1, 5.574865817313206: 1, 5.566467951
867118: 1, 5.560181230419852: 1, 5.5572430360599405: 1, 5.5551319961735155:
1, 5.549114635555032: 1, 5.546488421235607: 1, 5.540122229751908: 1, 5.539103
685465627: 1, 5.5382129951027155: 1, 5.533256866847277: 1, 5.527525842972306:
1, 5.525823399896011: 1, 5.525092909648835: 1, 5.523953290867478: 1, 5.517093
808660939: 1, 5.504487129449134: 1, 5.50156022314873: 1, 5.497704713542346:
1, 5.497398782045207: 1, 5.495262074766986: 1, 5.49458777462101: 1, 5.4921470
91900747: 1, 5.49112954653771: 1, 5.4864746703478495: 1, 5.473674833315864:
1, 5.473200924123203: 1, 5.47301848462992: 1, 5.464257370539457: 1, 5.4526632
57749741: 1, 5.448462070542298: 1, 5.44536877211946: 1, 5.434786727634329: 1,
5.426861559074385: 1, 5.42538562030258: 1, 5.423389378100735: 1, 5.4177579721
50144: 1, 5.41412905152464: 1, 5.410310747725259: 1, 5.408151991739899: 1, 5.
405254973206208: 1, 5.403324660753339: 1, 5.400603499333684: 1, 5.39991076430
4369: 1, 5.397067360538302: 1, 5.390876615818525: 1, 5.386917857707983: 1, 5.
385762510362293: 1, 5.378838929710963: 1, 5.373853191290335: 1, 5.37005790503
0907: 1, 5.36925979495106: 1, 5.367819369554628: 1, 5.367796435806606: 1, 5.3
6593994388916: 1, 5.355256599191076: 1, 5.3552481353099015: 1, 5.352018962275
275: 1, 5.341873585506352: 1, 5.34047185537057: 1, 5.34000740303809: 1, 5.336
927117594601: 1, 5.3318009498488745: 1, 5.328178135254688: 1, 5.3241203000190
6: 1, 5.318507746565118: 1, 5.316797631459034: 1, 5.304472893134923: 1, 5.303
```

```
571514090819: 1, 5.295576009548019: 1, 5.293469586489636: 1, 5.29342811790520
8: 1, 5.288013882205691: 1, 5.287078947608058: 1, 5.286189037430435: 1, 5.280
0973936650895: 1, 5.279173808972164: 1, 5.276853025705664: 1, 5.2751246658169
5: 1, 5.26709737892225: 1, 5.26633422756434: 1, 5.259691915391318: 1, 5.25542
8293040419: 1, 5.25312809463734: 1, 5.25038962105053: 1, 5.2402769344023135:
1, 5.239252635942185: 1, 5.2391141858561445: 1, 5.230045750819418: 1, 5.22924
8235602179: 1, 5.228394095128971: 1, 5.2282306050396565: 1, 5.22259741819496
6: 1, 5.219747209674529: 1, 5.219710351804213: 1, 5.21926851334679: 1, 5.2188
50806543854: 1, 5.2177989407585565: 1, 5.217647743941026: 1, 5.21575947354273
7: 1, 5.2154862320866435: 1, 5.209760948376883: 1, 5.209368589670923: 1, 5.20
8661545489801: 1, 5.196711656502643: 1, 5.196481532275906: 1, 5.1940233801683
7: 1, 5.192008029511487: 1, 5.191469743956434: 1, 5.187426617014412: 1, 5.182
542300954152: 1, 5.179663471669938: 1, 5.178029166787163: 1, 5.1772746135037
2: 1, 5.173416529836942: 1, 5.167961758095888: 1, 5.166068097324087: 1, 5.160
039421556804: 1, 5.151434397689478: 1, 5.150640718015276: 1, 5.14715697084702
3: 1, 5.141656785533016: 1, 5.139545414776285: 1, 5.1391087396799415: 1, 5.13
9102726284526: 1, 5.1387277135643705: 1, 5.137561323083222: 1, 5.137251416534
906: 1, 5.130398020693726: 1, 5.129406222166804: 1, 5.124176265166728: 1, 5.1
233900365018945: 1, 5.1204097275264076: 1, 5.117165738510431: 1, 5.1167244831
029235: 1, 5.110830575697476: 1, 5.109742383995429: 1, 5.108520118363152: 1,
5.10129937961773: 1, 5.099683029529435: 1, 5.095160180115223: 1, 5.0915006284
84348: 1, 5.088285581823394: 1, 5.085881691608736: 1, 5.0857391892516395: 1,
5.081984728757342: 1, 5.079443269600216: 1, 5.077214523667051: 1, 5.073334486
76339: 1, 5.07259388935648: 1, 5.07199702599723: 1, 5.071238665056737: 1, 5.0
69704596760204: 1, 5.066064928179798: 1, 5.047072275637284: 1, 5.044564482834
343: 1, 5.041365852636633: 1, 5.036810378722365: 1, 5.034196714288374: 1, 5.0
330755767067785: 1, 5.024933047142936: 1, 5.019304369618124: 1, 5.01714772156
2332: 1, 5.0163340062759945: 1, 5.016197932226737: 1, 5.015762399130292: 1,
5.013613529126699: 1, 5.012937639376456: 1, 5.004119396335387: 1, 5.003990887
075734: 1, 5.000816715706612: 1, 5.0002302741459985: 1, 4.99917590228945: 1,
4.996987795689085: 1, 4.995937959135309: 1, 4.990401455554361: 1, 4.989830519
801238: 1, 4.9861720575383615: 1, 4.983760375607721: 1, 4.983392211871997: 1,
4.979996630709471: 1, 4.97542933981273: 1, 4.967317864134086: 1, 4.9667709843
36635: 1, 4.964531571431469: 1, 4.963267000134191: 1, 4.958812078131683: 1,
4.954011458666871: 1, 4.948826820610826: 1, 4.940618530733799: 1, 4.935283665
449871: 1, 4.9295445261121165: 1, 4.926661684256046: 1, 4.923658969388224: 1,
4.917073808541159: 1, 4.9141533884132125: 1, 4.913069304624087: 1, 4.91304643
0609061: 1, 4.909855708960286: 1, 4.909626971613429: 1, 4.888744194379636: 1,
4.884705969505607: 1, 4.881961437255893: 1, 4.875273219503743: 1, 4.872204359
096002: 1, 4.867244611269203: 1, 4.867230655052645: 1, 4.8637461698744975: 1,
4.862001451344294: 1, 4.861436089575404: 1, 4.860663013573066: 1, 4.857473338
67432: 1, 4.854376676153361: 1, 4.853846679531762: 1, 4.85011011994169: 1, 4.
843521895395964: 1, 4.840402646259852: 1, 4.840015122671506: 1, 4.83577968141
666: 1, 4.8321568025725075: 1, 4.830161275986583: 1, 4.826521074470655: 1, 4.
825783871071119: 1, 4.824271504165992: 1, 4.823643580017655: 1, 4.82092275192
8807: 1, 4.819143528952095: 1, 4.818589520217347: 1, 4.814966144042782: 1, 4.
814796519817051: 1, 4.808373950067952: 1, 4.800619872162599: 1, 4.78479969457
958: 1, 4.783574308105261: 1, 4.780554980125186: 1, 4.779811401132637: 1, 4.7
77240505444823: 1, 4.774870714054617: 1, 4.772463670546829: 1, 4.766792540655
918: 1, 4.765396673893666: 1, 4.764429045475557: 1, 4.758548763867346: 1, 4.7
5551272968975: 1, 4.7510580845433354: 1, 4.750845340958494: 1, 4.732777928719
402: 1, 4.731376037590164: 1, 4.72590816068288: 1, 4.719936455104009: 1, 4.71
8151568833255: 1, 4.717396166402073: 1, 4.716005018660816: 1, 4.7151279836237
04: 1, 4.709041974071358: 1, 4.708248416348708: 1, 4.707682136466585: 1, 4.70
7478424491602: 1, 4.705892268203699: 1, 4.704187951148797: 1, 4.7012924968874
71: 1, 4.6977303590494: 1, 4.693401753147816: 1, 4.691763966160766: 1, 4.6911
62474064539: 1, 4.689753717252184: 1, 4.686682319795097: 1, 4.68616910360381
```

```
6: 1, 4.685505776855505: 1, 4.68201052511367: 1, 4.681152341107336: 1, 4.6735
07161359228: 1, 4.672206142263701: 1, 4.668435453966127: 1, 4.66758691689535
4: 1, 4.667045763404589: 1, 4.662109869085797: 1, 4.659985397994447: 1, 4.654
327012997777: 1, 4.652174851488445: 1, 4.651772105642936: 1, 4.63756946651179
4: 1, 4.6363427006814435: 1, 4.6344660142605205: 1, 4.631283701104526: 1, 4.6
26186491243589: 1, 4.623012188159043: 1, 4.618023403698603: 1, 4.611989579572
197: 1, 4.6117793661227875: 1, 4.605435551595464: 1, 4.603860615684213: 1, 4.
601785001735931: 1, 4.601657398129927: 1, 4.600237249569179: 1, 4.59877743634
3352: 1, 4.595770110218607: 1, 4.59287938134959: 1, 4.587152452573922: 1, 4.5
825054642414775: 1, 4.581840637516545: 1, 4.5691742563423645: 1, 4.5685195427
49151: 1, 4.56253367083691: 1, 4.562103491970768: 1, 4.559927883180678: 1, 4.
558124462193932: 1, 4.554186092502289: 1, 4.549663447106328: 1, 4.54945606147
7006: 1, 4.549185715137342: 1, 4.544609190776423: 1, 4.540328789473806: 1, 4.
5390305043849875: 1, 4.532849077261454: 1, 4.531480291641631: 1, 4.5309305750
284325: 1, 4.528759921096193: 1, 4.522976835677435: 1, 4.518004545774948: 1,
4.5131425385314135: 1, 4.513049670287853: 1, 4.512931056306354: 1, 4.51137698
1035881: 1, 4.509143690769421: 1, 4.503676925812732: 1, 4.502645070579363: 1,
4.492534786685032: 1, 4.490808346239298: 1, 4.486489094193614: 1, 4.481622065
91572: 1, 4.480508172521678: 1, 4.480151410013625: 1, 4.478993417622771: 1,
4.471230728654521: 1, 4.46480075885361: 1, 4.462938806898724: 1, 4.4623956200
536945: 1, 4.461300296555349: 1, 4.460369124171409: 1, 4.449420395744561: 1,
4.443741339557745: 1, 4.440086943901259: 1, 4.438444288810698: 1, 4.433531939
251009: 1, 4.432917022734689: 1, 4.432899293555821: 1, 4.432623557692948: 1,
4.431765945416446: 1, 4.42952389546809: 1, 4.428549105281753: 1, 4.4264377197
69937: 1, 4.421977076383307: 1, 4.421067409974928: 1, 4.419859822323346: 1,
4.413127641580736: 1, 4.4055089351434455: 1, 4.401871110232936: 1, 4.39744175
9094916: 1, 4.397302414033012: 1, 4.393757584297554: 1, 4.380694062559525: 1,
4.377381378713802: 1, 4.376757136911188: 1, 4.376742548616693: 1, 4.375764295
730915: 1, 4.372393055978392: 1, 4.371178154190673: 1, 4.370025513601144: 1,
4.368178456397431: 1, 4.367456817194874: 1, 4.367116320732281: 1, 4.364379588
05694: 1, 4.360849766314797: 1, 4.359567713521798: 1, 4.359306188971186: 1,
4.350855887861104: 1, 4.346510709482785: 1, 4.344594123341414: 1, 4.344501347
2655445: 1, 4.344287405590336: 1, 4.342828412140884: 1, 4.335401474756036: 1,
4.3323712137829204: 1, 4.3299237524631025: 1, 4.328541957328493: 1, 4.3265684
56032919: 1, 4.326351692478465: 1, 4.326003170034008: 1, 4.325290774360554:
1, 4.323218921420116: 1, 4.32058669656242: 1, 4.319366389137365: 1, 4.3186418
00196039: 1, 4.317328681421649: 1, 4.316966148700658: 1, 4.316541506406982:
1, 4.315748329160896: 1, 4.309507108129744: 1, 4.308044836992526: 1, 4.307084
459675557: 1, 4.306345545136171: 1, 4.306096909696907: 1, 4.305638763332069:
1, 4.303384269651744: 1, 4.2929516792361255: 1, 4.290622906409278: 1, 4.28940
341336744: 1, 4.286546893568522: 1, 4.282070701700747: 1, 4.281020469268494:
1, 4.280189507124145: 1, 4.277773504701998: 1, 4.277153851008868: 1, 4.275890
216498071: 1, 4.272452535680559: 1, 4.271136458291137: 1, 4.270116430011218:
1, 4.262866194998701: 1, 4.261857238804941: 1, 4.257083904087646: 1, 4.256470
617323121: 1, 4.256024549187585: 1, 4.254935660547367: 1, 4.254869165798242:
1, 4.250263222536104: 1, 4.2486129916440465: 1, 4.231331775153977: 1, 4.23083
9269603646: 1, 4.229816388374982: 1, 4.229599420161982: 1, 4.225044608581610
5: 1, 4.223707342332759: 1, 4.222076773760913: 1, 4.213972528222072: 1, 4.213
480334657126: 1, 4.213393958918472: 1, 4.212522777723636: 1, 4.20760753182868
3: 1, 4.207335445930045: 1, 4.203524686254418: 1, 4.2027881978089745: 1, 4.20
1295419803789: 1, 4.199677760381985: 1, 4.198852854815441: 1, 4.1857289807522
1: 1, 4.183559193749119: 1, 4.18213148558672: 1, 4.181175870593073: 1, 4.1802
434376473085: 1, 4.171900732975008: 1, 4.17018633059898: 1, 4.16826683524212
5: 1, 4.166137332523187: 1, 4.165112126158085: 1, 4.157542677434523: 1, 4.156
919807802443: 1, 4.147936391712854: 1, 4.143764890892602: 1, 4.13555588852573
4: 1, 4.129879828581019: 1, 4.127028379396388: 1, 4.1257410685743405: 1, 4.12
4043230914463: 1, 4.123300961384624: 1, 4.122335788126411: 1, 4.1165505438427
```

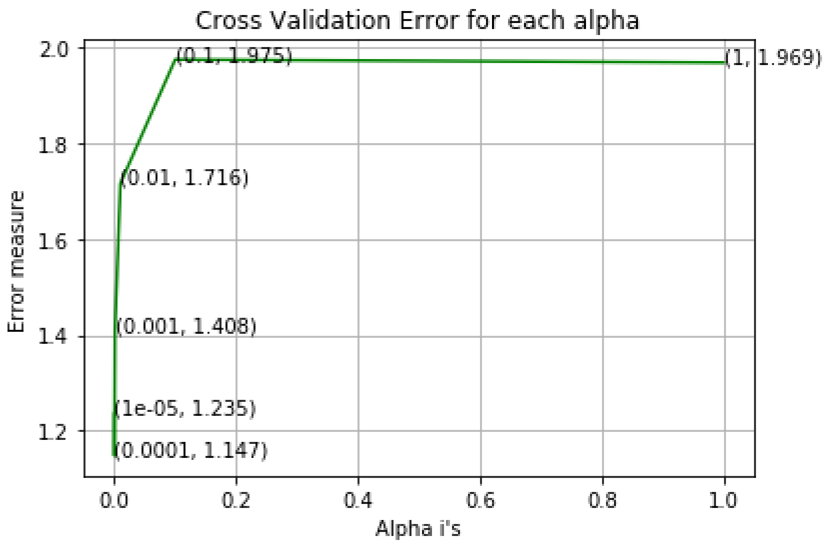
```
32: 1, 4.11643997837879: 1, 4.114175524090575: 1, 4.111695457503326: 1, 4.109
735625223532: 1, 4.099382761158114: 1, 4.083484426026838: 1, 4.07988680440521
3: 1, 4.0748946496898695: 1, 4.069555398849994: 1, 4.063188577413284: 1, 4.06
2904045662565: 1, 4.059583645493776: 1, 4.058941684327489: 1, 4.0581410354592
17: 1, 4.057101822091445: 1, 4.054806664455797: 1, 4.05376818061998: 1, 4.052
717644563658: 1, 4.052042083588036: 1, 4.051748457781748: 1, 4.04981719561848
8: 1, 4.049461500281358: 1, 4.048194596172725: 1, 4.041414780847652: 1, 4.040
937680516905: 1, 4.040351588726581: 1, 4.031410629156825: 1, 4.02466333057338
4: 1, 4.01572493434426: 1, 4.013774235673933: 1, 4.006487567957035: 1, 4.0063
05164824853: 1, 4.004679836522561: 1, 4.003163846168536: 1, 3.99593485117573
5: 1, 3.993068265681581: 1, 3.992756927442648: 1, 3.984116334128005: 1, 3.982
7773417028136: 1, 3.9734162095455887: 1, 3.967945291622662: 1, 3.964762603375
347: 1, 3.9587853298330637: 1, 3.9528702645613842: 1, 3.9483937176912476: 1,
3.944566415641645: 1, 3.9433282008074184: 1, 3.9410381432077637: 1, 3.9388585
67542327: 1, 3.9376334453871107: 1, 3.936935866183822: 1, 3.9358763788146263:
1, 3.935754725104361: 1, 3.9242996892171367: 1, 3.9229779910443567: 1, 3.9226
852021634335: 1, 3.922055303675956: 1, 3.9216044886656074: 1, 3.9168491855318
57: 1, 3.9121187636644157: 1, 3.909176898431175: 1, 3.904285495062316: 1, 3.8
999481218209295: 1, 3.8963481271829354: 1, 3.8962588105484013: 1, 3.894990563
9079234: 1, 3.8899169331497347: 1, 3.8886426028016228: 1, 3.8865948292649333:
1, 3.8861807877800305: 1, 3.8846797383886535: 1, 3.882204841963448: 1, 3.8748
2881307732: 1, 3.8741566272076944: 1, 3.8718312158086694: 1, 3.86689325196122
3: 1, 3.8651498630481713: 1, 3.862209614681115: 1, 3.856569490893882: 1, 3.85
0250655020688: 1, 3.84832821685527: 1, 3.8442718632757935: 1, 3.8404114040471
145: 1, 3.8370534246385013: 1, 3.8342632780897095: 1, 3.82557831254686: 1, 3.
823394763622514: 1, 3.8215020417953305: 1, 3.8174101977525483: 1, 3.802996366
0138058: 1, 3.799851364186463: 1, 3.7923491189259817: 1, 3.79183875455049: 1,
3.780332616282511З: 1, 3.7797904930328086: 1, 3.779353263651681: 1, 3.7732733
74744315: 1, 3.7678886190882683: 1, 3.7598422360045505: 1, 3.759331748931161:
1, 3.754172180470034: 1, 3.7534110270769503: 1, 3.7458817283244032: 1, 3.7425
000878886: 1, 3.7386300170966065: 1, 3.7357184401078696: 1, 3.732233508753227
5: 1, 3.731020651470904: 1, 3.714058416836788: 1, 3.71183347447474: 1, 3.7108
93861715661: 1, 3.7093912031074967: 1, 3.7022977969279487: 1, 3.7006137636531
053: 1, 3.699647983749271: 1, 3.698566418736542: 1, 3.697968594970404: 1, 3.6
954522839365356: 1, 3.693569392982347: 1, 3.6796584281043736: 1, 3.6787787736
77083: 1, 3.6771074221822775: 1, 3.665612553223362: 1, 3.664007275462196: 1,
3.6559150590195366: 1, 3.649748892328028: 1, 3.6493412155219906: 1, 3.6448242
60017095: 1, 3.6372209139346094: 1, 3.6357374102185367: 1, 3.635631836744049:
1, 3.6238743158811677: 1, 3.6232962055143916: 1, 3.6213998893797483: 1, 3.618
342448994218: 1, 3.617390638682892: 1, 3.6149830263416245: 1, 3.6120025785494
567: 1, 3.608938665913949: 1, 3.6084825472652606: 1, 3.60606196866689: 1, 3.6
0584336931105: 1, 3.605075436154209: 1, 3.599215862674989: 1, 3.5922560330109
65: 1, 3.5910499296421063: 1, 3.5903566090030776: 1, 3.5890501610735708: 1,
3.5834567099685417: 1, 3.579700167260542: 1, 3.5778653687578346: 1, 3.5748099
335473777: 1, 3.5610035405092244: 1, 3.555132428107007: 1, 3.552410834540552
3: 1, 3.542692271706438: 1, 3.5406144974969864: 1, 3.539121062501042: 1, 3.53
53665938691825: 1, 3.5313139248770424: 1, 3.530011406647952: 1, 3.51960723151
22717: 1, 3.518878227110436: 1, 3.514394226927604: 1, 3.4949029860212: 1, 3.4
94298483747906: 1, 3.476619318230893: 1, 3.4736533826819045: 1, 3.46562335623
6276: 1, 3.4630488681844134: 1, 3.459734375009226: 1, 3.4584914478238415: 1,
3.451599657341827: 1, 3.4441117280130307: 1, 3.4288148513287955: 1, 3.4285858
088348453: 1, 3.4266104913766777: 1, 3.422446152652424: 1, 3.420416815591325:
1, 3.4203638814273183: 1, 3.4194922410568886: 1, 3.4170203672876993: 1, 3.410
9222976178417: 1, 3.4083825780730703: 1, 3.4046451245020934: 1, 3.39239236279
81654: 1, 3.3779936043288274: 1, 3.375142765702431: 1, 3.3712266410126546: 1,
3.3650215089775624: 1, 3.3623998624727043: 1, 3.3592968711255917: 1, 3.324408
2609487973: 1, 3.321682462603624: 1, 3.3116835006788055: 1, 3.311007670585437
```

```
4: 1, 3.285102275617504: 1, 3.2826119981271624: 1, 3.279865359000517: 1, 3.26
50618333861394: 1, 3.264829316169007: 1, 3.2451861860609696: 1, 3.24160734888
93438: 1, 3.2329109237661546: 1, 3.232102281320863: 1, 3.2314681535085623: 1,
3.2302060670910673: 1, 3.2283855716231846: 1, 3.22182559381736: 1, 3.19261933
7525961: 1, 3.191696059025165: 1, 3.17530741471275: 1, 3.1611050699238548: 1,
3.1372605383494165: 1, 3.12962849105549: 1, 3.1151649941921136: 1, 3.11070474
2762536: 1, 3.029227647230069: 1, 3.011290252475502: 1, 2.9844354094402337:
1, 2.9481930267343914: 1, 2.8969503237621868: 1, 2.8733186343448827: 1, 2.804
8636289212063: 1})
```

In [48]:
```
# 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: 1.2348758040945294
For values of alpha =  0.0001 The log loss is: 1.1472992695168795
For values of alpha =  0.001 The log loss is: 1.408227120945089
For values of alpha =  0.01 The log loss is: 1.7158457094716275
For values of alpha =  0.1 The log loss is: 1.9749212413732633
For values of alpha =  1 The log loss is: 1.968553429107737
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7225986656836202
For values of best alpha =  0.0001 The cross validation log loss is: 1.147299
2695168795
For values of best alpha =  0.0001 The test log loss is: 1.0928193506142339
```

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

Ans. Yes, it seems like!

In [57]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features = 2000, min_df = 3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [58]:
```
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")
```

```
93.4 % of word of test data appeared in train data
93.15 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [59]:
```
#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 [60]:
```
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 [62]:
```
# 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(max_features = 2000, min_df = 3)

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

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

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

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

Stacking the three types of features

In [63]:
```
# 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 [64]:
```
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, 4197)
(number of data points * number of features) in test data =  (665, 4197)
(number of data points * number of features) in cross validation data = (532,
4197)
```

In [65]:
```
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

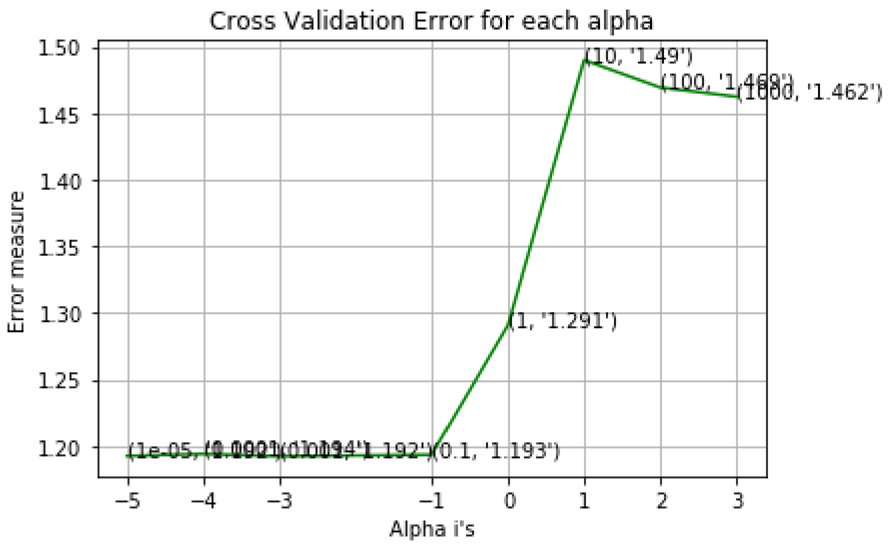
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [66]:
```
# 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 : 1.1924285370066234
for alpha = 0.0001
Log Loss : 1.1939526689215836
for alpha = 0.001
Log Loss : 1.1922098062212645
for alpha = 0.1
Log Loss : 1.1933462578193588
for alpha = 1
Log Loss : 1.290964242051232
for alpha = 10
Log Loss : 1.489934025529102
for alpha = 100
Log Loss : 1.4692870103565572
for alpha = 1000
Log Loss : 1.4624021714197004
```

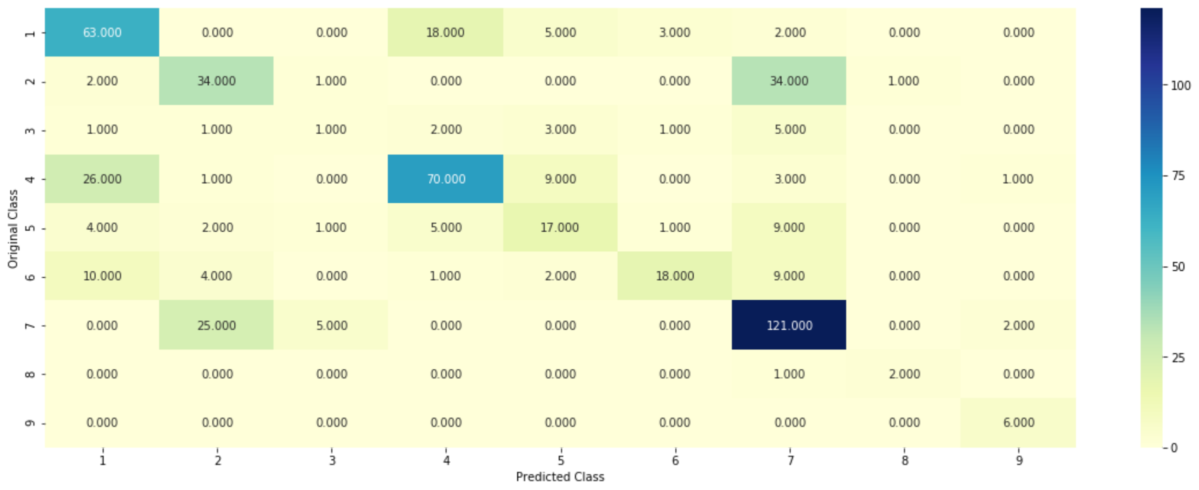


```
For values of best alpha =  0.001 The train log loss is: 0.5758869835068372
For values of best alpha =  0.001 The cross validation log loss is: 1.1922098
062212645
For values of best alpha =  0.001 The test log loss is: 1.2355585566598517
```

4.1.1.2. Testing the model with best hyper paramters

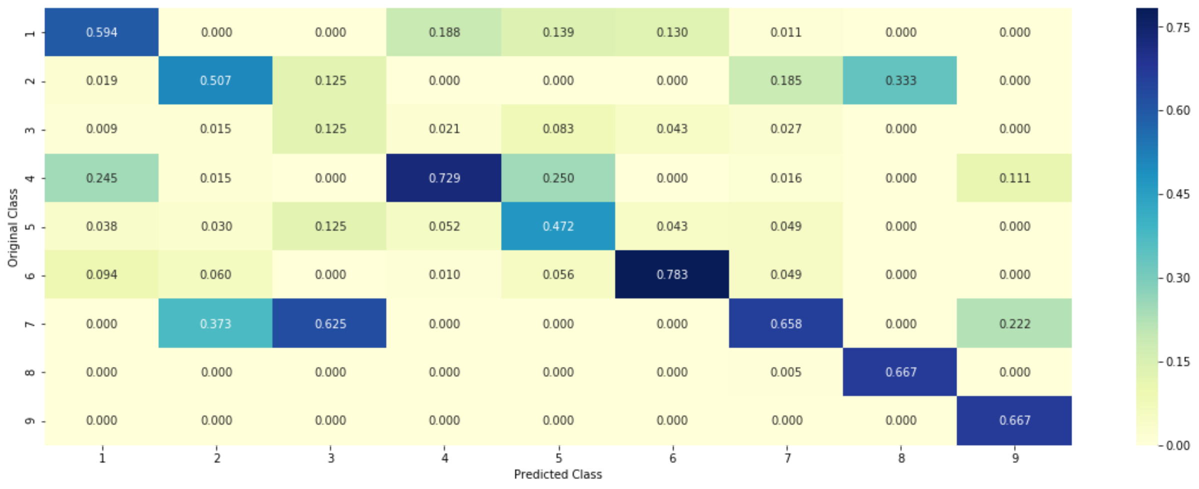
In [67]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

# 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 : 1.1922098062212645
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```



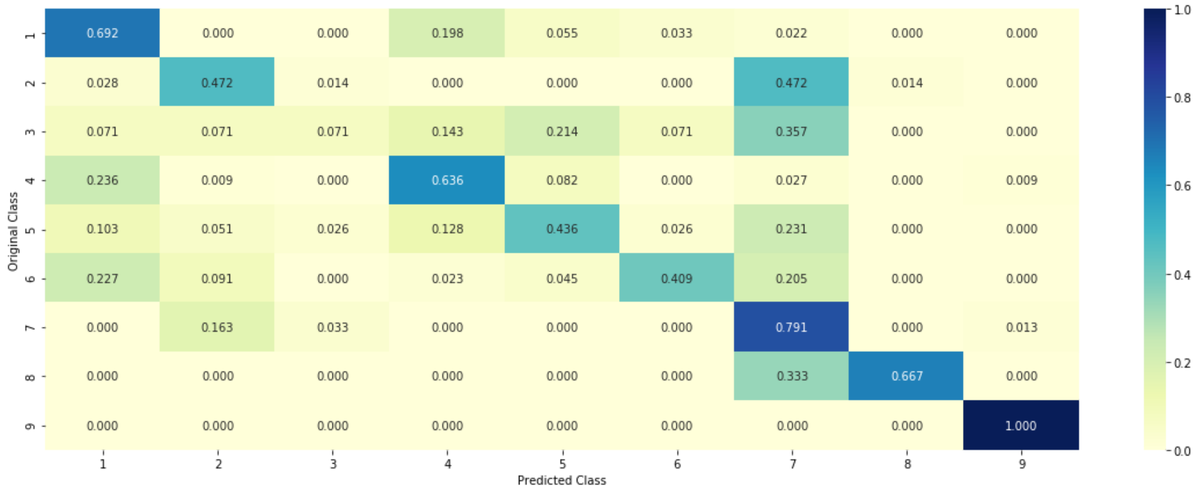
Original \ Predicted	1	2	3	4	5	6	7	8	9
1	63.000	0.000	0.000	18.000	5.000	3.000	2.000	0.000	0.000
2	2.000	34.000	1.000	0.000	0.000	0.000	34.000	1.000	0.000
3	1.000	1.000	1.000	2.000	3.000	1.000	5.000	0.000	0.000
4	26.000	1.000	0.000	70.000	9.000	0.000	3.000	0.000	1.000
5	4.000	2.000	1.000	5.000	17.000	1.000	9.000	0.000	0.000
6	10.000	4.000	0.000	1.000	2.000	18.000	9.000	0.000	0.000
7	0.000	25.000	5.000	0.000	0.000	0.000	121.000	0.000	2.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.594	0.000	0.000	0.188	0.139	0.130	0.011	0.000	0.000
2	0.019	0.507	0.125	0.000	0.000	0.000	0.185	0.333	0.000
3	0.009	0.015	0.125	0.021	0.083	0.043	0.027	0.000	0.000
4	0.245	0.015	0.000	0.729	0.250	0.000	0.016	0.000	0.111
5	0.038	0.030	0.125	0.052	0.472	0.043	0.049	0.000	0.000
6	0.094	0.060	0.000	0.010	0.056	0.783	0.049	0.000	0.000
7	0.000	0.373	0.625	0.000	0.000	0.000	0.658	0.000	0.222
8	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.667	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.667

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



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.692	0.000	0.000	0.198	0.055	0.033	0.022	0.000	0.000
2	0.028	0.472	0.014	0.000	0.000	0.000	0.472	0.014	0.000
3	0.071	0.071	0.071	0.143	0.214	0.071	0.357	0.000	0.000
4	0.236	0.009	0.000	0.636	0.082	0.000	0.027	0.000	0.009
5	0.103	0.051	0.026	0.128	0.436	0.026	0.231	0.000	0.000
6	0.227	0.091	0.000	0.023	0.045	0.409	0.205	0.000	0.000
7	0.000	0.163	0.033	0.000	0.000	0.000	0.791	0.000	0.013
8	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

4.1.1.3. Feature Importance, Correctly classified point

In [68]:
```
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 : 2
Predicted Class Probabilities: [[0.0748 0.3886 0.015  0.089  0.0425 0.0448 0.
337  0.0052 0.0032]]
Actual Class : 2
--------------------------------------------------
12 Text feature [treatment] present in test data point [True]
14 Text feature [patients] present in test data point [True]
15 Text feature [response] present in test data point [True]
17 Text feature [clinical] present in test data point [True]
19 Text feature [study] present in test data point [True]
25 Text feature [13] present in test data point [True]
27 Text feature [molecular] present in test data point [True]
28 Text feature [however] present in test data point [True]
30 Text feature [11] present in test data point [True]
31 Text feature [18] present in test data point [True]
32 Text feature [15] present in test data point [True]
33 Text feature [including] present in test data point [True]
34 Text feature [12] present in test data point [True]
35 Text feature [different] present in test data point [True]
37 Text feature [another] present in test data point [True]
38 Text feature [also] present in test data point [True]
39 Text feature [may] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [mutations] present in test data point [True]
44 Text feature [kinase] present in test data point [True]
45 Text feature [mutation] present in test data point [True]
46 Text feature [19] present in test data point [True]
47 Text feature [harbor] present in test data point [True]
49 Text feature [using] present in test data point [True]
51 Text feature [rate] present in test data point [True]
52 Text feature [recently] present in test data point [True]
53 Text feature [analysis] present in test data point [True]
55 Text feature [small] present in test data point [True]
56 Text feature [17] present in test data point [True]
57 Text feature [cases] present in test data point [True]
58 Text feature [reported] present in test data point [True]
59 Text feature [detection] present in test data point [True]
60 Text feature [according] present in test data point [True]
61 Text feature [achieved] present in test data point [True]
62 Text feature [treated] present in test data point [True]
63 Text feature [one] present in test data point [True]
64 Text feature [due] present in test data point [True]
66 Text feature [observed] present in test data point [True]
67 Text feature [common] present in test data point [True]
68 Text feature [positive] present in test data point [True]
70 Text feature [respectively] present in test data point [True]
71 Text feature [disease] present in test data point [True]
72 Text feature [higher] present in test data point [True]
73 Text feature [although] present in test data point [True]
75 Text feature [harboring] present in test data point [True]
76 Text feature [inhibitor] present in test data point [True]
77 Text feature [gene] present in test data point [True]
80 Text feature [complete] present in test data point [True]
82 Text feature [table] present in test data point [True]
83 Text feature [studies] present in test data point [True]
84 Text feature [line] present in test data point [True]
85 Text feature [overall] present in test data point [True]
```

```
87 Text feature [previously] present in test data point [True]
88 Text feature [subsequently] present in test data point [True]
89 Text feature [27] present in test data point [True]
90 Text feature [described] present in test data point [True]
92 Text feature [similar] present in test data point [True]
93 Text feature [present] present in test data point [True]
94 Text feature [patient] present in test data point [True]
97 Text feature [found] present in test data point [True]
98 Text feature [detected] present in test data point [True]
99 Text feature [shown] present in test data point [True]
Out of the top  100  features  62 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [69]:
```
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 : 1
Predicted Class Probabilities: [[0.5018 0.0518 0.0127 0.2606 0.036  0.038  0.
092  0.0044 0.0027]]
Actual Class : 4
--------------------------------------------------
11 Text feature [protein] present in test data point [True]
12 Text feature [dna] present in test data point [True]
13 Text feature [one] present in test data point [True]
14 Text feature [type] present in test data point [True]
15 Text feature [results] present in test data point [True]
16 Text feature [function] present in test data point [True]
19 Text feature [wild] present in test data point [True]
20 Text feature [loss] present in test data point [True]
21 Text feature [also] present in test data point [True]
22 Text feature [specific] present in test data point [True]
23 Text feature [table] present in test data point [True]
24 Text feature [role] present in test data point [True]
25 Text feature [two] present in test data point [True]
26 Text feature [containing] present in test data point [True]
27 Text feature [therefore] present in test data point [True]
28 Text feature [region] present in test data point [True]
29 Text feature [using] present in test data point [True]
30 Text feature [either] present in test data point [True]
31 Text feature [human] present in test data point [True]
32 Text feature [functions] present in test data point [True]
33 Text feature [shown] present in test data point [True]
34 Text feature [determined] present in test data point [True]
35 Text feature [binding] present in test data point [True]
36 Text feature [may] present in test data point [True]
37 Text feature [control] present in test data point [True]
38 Text feature [present] present in test data point [True]
39 Text feature [effect] present in test data point [True]
40 Text feature [however] present in test data point [True]
41 Text feature [involved] present in test data point [True]
42 Text feature [important] present in test data point [True]
43 Text feature [result] present in test data point [True]
44 Text feature [similar] present in test data point [True]
45 Text feature [affect] present in test data point [True]
46 Text feature [expression] present in test data point [True]
47 Text feature [analysis] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [three] present in test data point [True]
50 Text feature [several] present in test data point [True]
51 Text feature [suggest] present in test data point [True]
52 Text feature [within] present in test data point [True]
53 Text feature [discussion] present in test data point [True]
54 Text feature [critical] present in test data point [True]
55 Text feature [gene] present in test data point [True]
56 Text feature [indicate] present in test data point [True]
57 Text feature [possible] present in test data point [True]
58 Text feature [25] present in test data point [True]
59 Text feature [least] present in test data point [True]
60 Text feature [amino] present in test data point [True]
61 Text feature [ability] present in test data point [True]
62 Text feature [described] present in test data point [True]
63 Text feature [used] present in test data point [True]
64 Text feature [including] present in test data point [True]
```

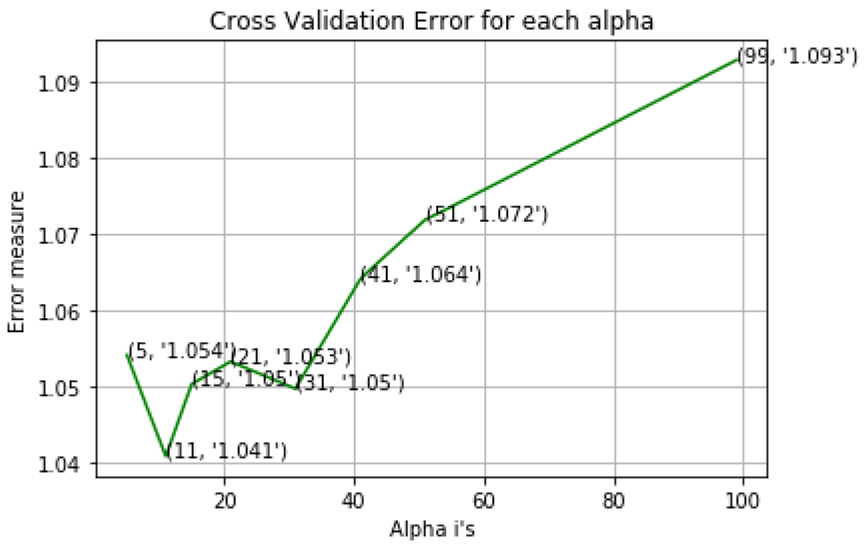
```
65 Text feature [whether] present in test data point [True]
66 Text feature [performed] present in test data point [True]
67 Text feature [previously] present in test data point [True]
68 Text feature [10] present in test data point [True]
69 Text feature [large] present in test data point [True]
70 Text feature [four] present in test data point [True]
71 Text feature [transcriptional] present in test data point [True]
72 Text feature [indicating] present in test data point [True]
73 Text feature [proteins] present in test data point [True]
74 Text feature [surface] present in test data point [True]
75 Text feature [observed] present in test data point [True]
76 Text feature [form] present in test data point [True]
77 Text feature [remaining] present in test data point [True]
78 Text feature [data] present in test data point [True]
79 Text feature [following] present in test data point [True]
82 Text feature [yet] present in test data point [True]
83 Text feature [many] present in test data point [True]
84 Text feature [obtained] present in test data point [True]
85 Text feature [example] present in test data point [True]
86 Text feature [deletion] present in test data point [True]
87 Text feature [essential] present in test data point [True]
88 Text feature [different] present in test data point [True]
89 Text feature [whereas] present in test data point [True]
90 Text feature [analyzed] present in test data point [True]
91 Text feature [addition] present in test data point [True]
92 Text feature [compared] present in test data point [True]
93 Text feature [transcription] present in test data point [True]
94 Text feature [associated] present in test data point [True]
95 Text feature [reduced] present in test data point [True]
96 Text feature [additional] present in test data point [True]
97 Text feature [cancer] present in test data point [True]
98 Text feature [sequence] present in test data point [True]
99 Text feature [although] present in test data point [True]
Out of the top  100  features  85 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [70]:
```
# 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 : 1.054236350281611
for alpha = 11
Log Loss : 1.0409264766934028
for alpha = 15
Log Loss : 1.0504139375128223
for alpha = 21
Log Loss : 1.0533736239496834
for alpha = 31
Log Loss : 1.0498125347616825
for alpha = 41
Log Loss : 1.0641849607938725
for alpha = 51
Log Loss : 1.0719345763203172
for alpha = 99
Log Loss : 1.092949745223565
```

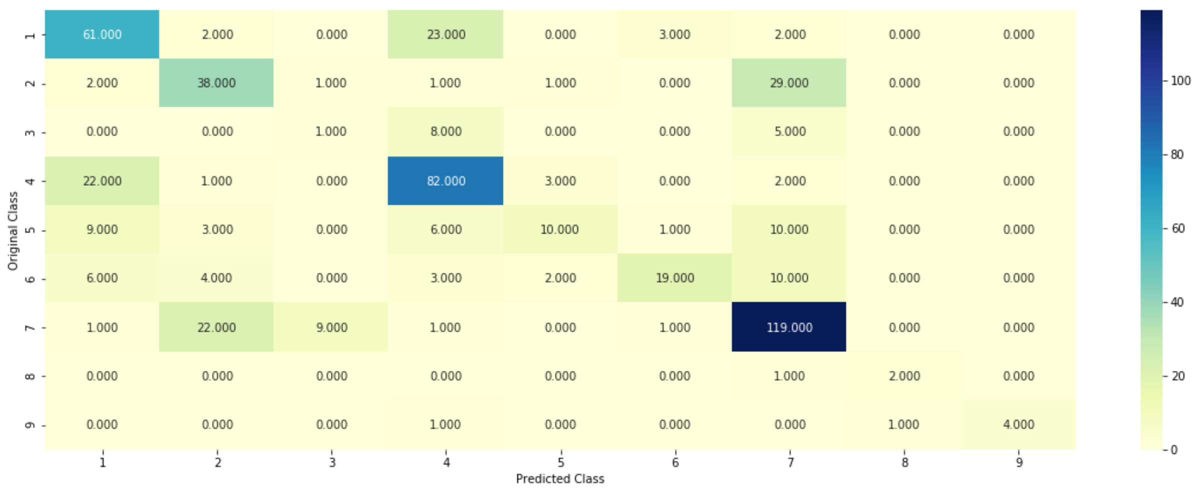


```
For values of best alpha =  11 The train log loss is: 0.6522900347475329
For values of best alpha =  11 The cross validation log loss is: 1.0409264766
934028
For values of best alpha =  11 The test log loss is: 1.0331000674723805
```

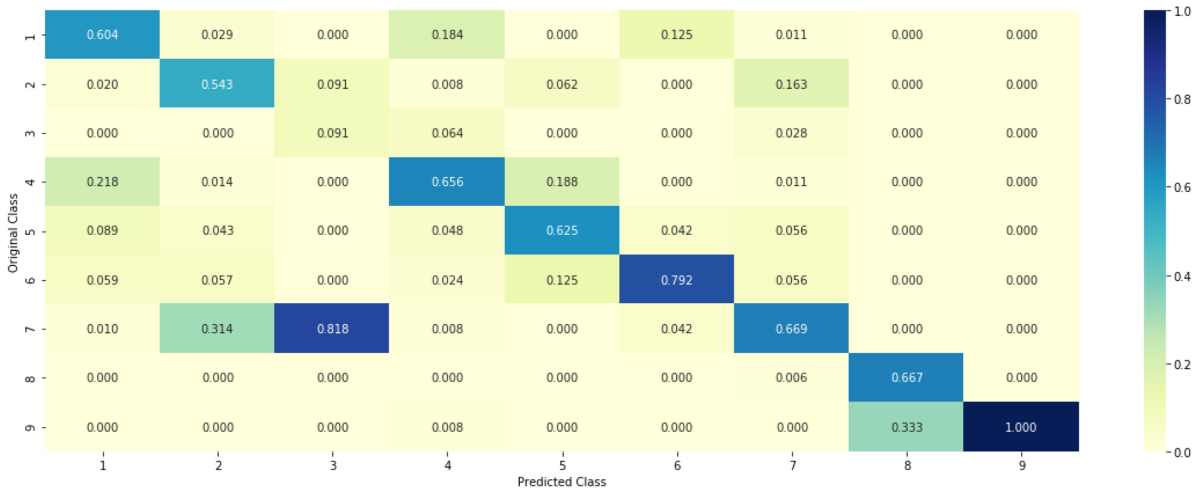
4.2.2. Testing the model with best hyper paramters

In [71]:
```
# 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)
```

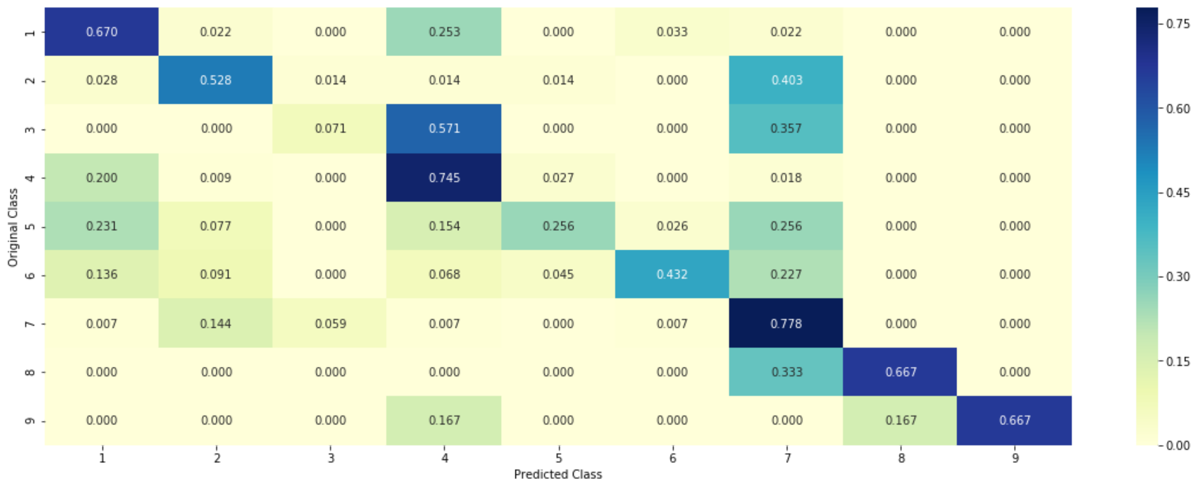
```
Log loss : 1.0409264766934028
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

In [72]:
```
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  11  nearest neighbours of the test points belongs to classes [2 2 2 2 2
2 7 5 7 2 7]
Fequency of nearest points : Counter({2: 7, 7: 3, 5: 1})
```

4.2.4. Sample Query Point-2

In [73]:
```
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 : 1
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belon
gs to classes [1 1 1 1 1 1 4 1 1 1 1]
Fequency of nearest points : Counter({1: 10, 4: 1})
```

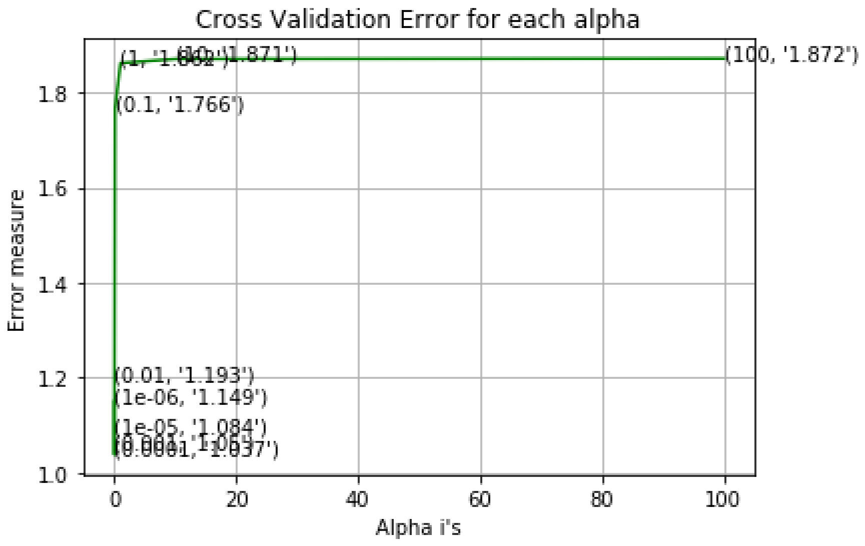
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [74]:
```
# 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 : 1.1494658222347294
for alpha = 1e-05
Log Loss : 1.084040543646714
for alpha = 0.0001
Log Loss : 1.0369927050543275
for alpha = 0.001
Log Loss : 1.0495644556404966
for alpha = 0.01
Log Loss : 1.1926060200335453
for alpha = 0.1
Log Loss : 1.7657651570203903
for alpha = 1
Log Loss : 1.8621036710744836
for alpha = 10
Log Loss : 1.8708135243254613
for alpha = 100
Log Loss : 1.8717431262885664
```

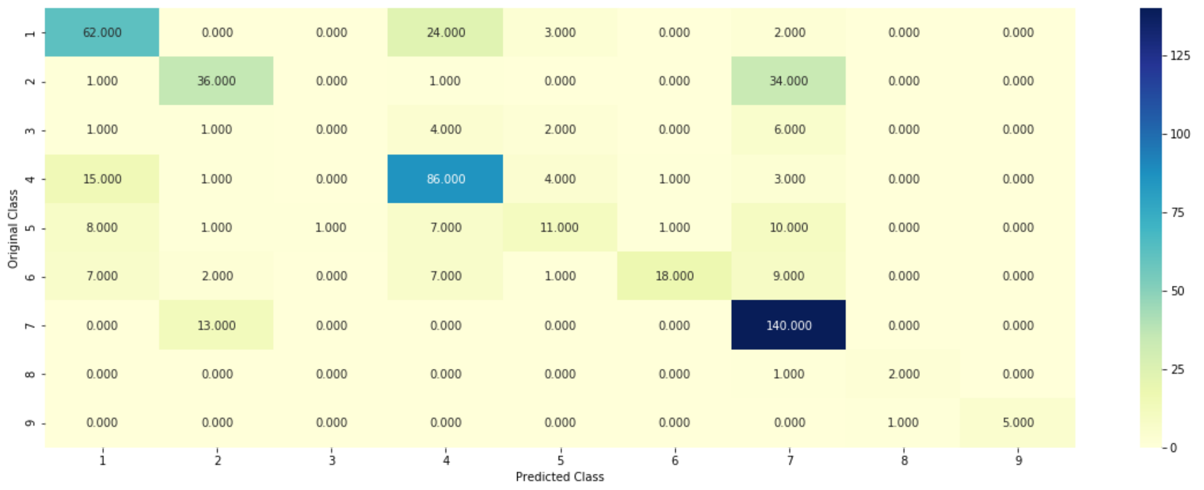


```
For values of best alpha =  0.0001 The train log loss is: 0.42273639669016916
For values of best alpha =  0.0001 The cross validation log loss is: 1.036992
7050543275
For values of best alpha =  0.0001 The test log loss is: 0.9679696082645026
```

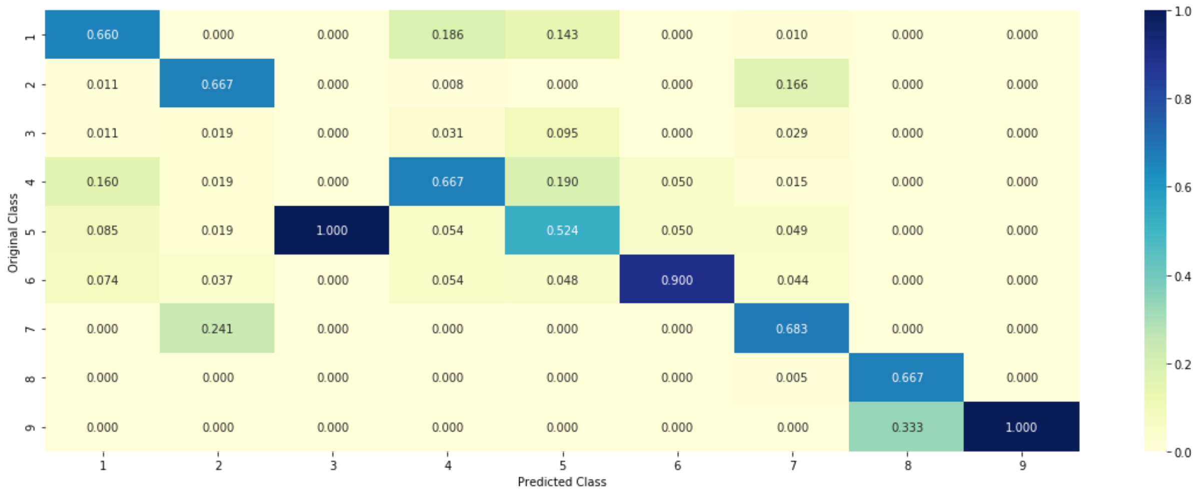
4.3.1.2. Testing the model with best hyper paramters

In [75]:
```
# 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)
```

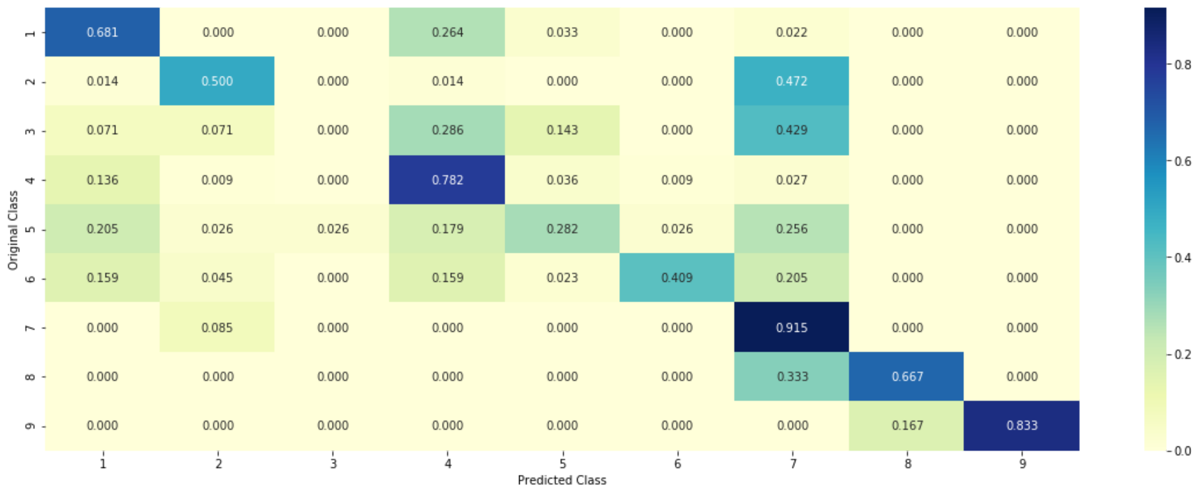
```
Log loss : 1.0369927050543275
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [76]:
```
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 [77]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0043 0.7697 0.0121 0.0154 0.029  0.0039 0.
1619 0.0025 0.0013]]
Actual Class : 2
--------------------------------------------------
160 Text feature [detection] present in test data point [True]
162 Text feature [play] present in test data point [True]
168 Text feature [actin] present in test data point [True]
176 Text feature [additionally] present in test data point [True]
184 Text feature [discovery] present in test data point [True]
192 Text feature [tcga] present in test data point [True]
193 Text feature [response] present in test data point [True]
195 Text feature [viability] present in test data point [True]
196 Text feature [90] present in test data point [True]
205 Text feature [currently] present in test data point [True]
206 Text feature [01] present in test data point [True]
209 Text feature [achieved] present in test data point [True]
224 Text feature [soft] present in test data point [True]
238 Text feature [subsequently] present in test data point [True]
239 Text feature [300] present in test data point [True]
241 Text feature [especially] present in test data point [True]
250 Text feature [rate] present in test data point [True]
253 Text feature [mek] present in test data point [True]
269 Text feature [step] present in test data point [True]
272 Text feature [novel] present in test data point [True]
282 Text feature [need] present in test data point [True]
296 Text feature [rates] present in test data point [True]
302 Text feature [80] present in test data point [True]
303 Text feature [weeks] present in test data point [True]
307 Text feature [active] present in test data point [True]
308 Text feature [craf] present in test data point [True]
310 Text feature [patients] present in test data point [True]
328 Text feature [compound] present in test data point [True]
339 Text feature [fusions] present in test data point [True]
341 Text feature [explain] present in test data point [True]
342 Text feature [plays] present in test data point [True]
346 Text feature [statistically] present in test data point [True]
348 Text feature [eight] present in test data point [True]
362 Text feature [apoptosis] present in test data point [True]
363 Text feature [finally] present in test data point [True]
364 Text feature [activates] present in test data point [True]
365 Text feature [group] present in test data point [True]
367 Text feature [another] present in test data point [True]
372 Text feature [05] present in test data point [True]
375 Text feature [nras] present in test data point [True]
376 Text feature [15] present in test data point [True]
380 Text feature [knockdown] present in test data point [True]
390 Text feature [complete] present in test data point [True]
399 Text feature [light] present in test data point [True]
402 Text feature [conformation] present in test data point [True]
403 Text feature [33] present in test data point [True]
405 Text feature [treatment] present in test data point [True]
413 Text feature [models] present in test data point [True]
415 Text feature [important] present in test data point [True]
417 Text feature [selective] present in test data point [True]
418 Text feature [oncogenic] present in test data point [True]
420 Text feature [inactive] present in test data point [True]
```

```
427 Text feature [member] present in test data point [True]
428 Text feature [200] present in test data point [True]
433 Text feature [87] present in test data point [True]
434 Text feature [regression] present in test data point [True]
436 Text feature [harboring] present in test data point [True]
439 Text feature [36] present in test data point [True]
441 Text feature [molecular] present in test data point [True]
443 Text feature [part] present in test data point [True]
444 Text feature [contribution] present in test data point [True]
448 Text feature [www] present in test data point [True]
450 Text feature [isoform] present in test data point [True]
455 Text feature [small] present in test data point [True]
456 Text feature [suggests] present in test data point [True]
458 Text feature [cases] present in test data point [True]
459 Text feature [types] present in test data point [True]
460 Text feature [concentrations] present in test data point [True]
462 Text feature [harbor] present in test data point [True]
464 Text feature [clinical] present in test data point [True]
466 Text feature [deletions] present in test data point [True]
473 Text feature [001] present in test data point [True]
474 Text feature [different] present in test data point [True]
478 Text feature [potentially] present in test data point [True]
480 Text feature [hek293] present in test data point [True]
481 Text feature [kras] present in test data point [True]
484 Text feature [clinically] present in test data point [True]
490 Text feature [discovered] present in test data point [True]
491 Text feature [since] present in test data point [True]
497 Text feature [150] present in test data point [True]
499 Text feature [mainly] present in test data point [True]
Out of the top  500  features  81 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [78]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7247 0.0097 0.0041 0.2327 0.0089 0.0093 0.
0061 0.0015 0.0029]]
Actual Class : 4
--------------------------------------------------
24 Text feature [aggregation] present in test data point [True]
103 Text feature [surface] present in test data point [True]
178 Text feature [hydrophobic] present in test data point [True]
180 Text feature [folding] present in test data point [True]
189 Text feature [plays] present in test data point [True]
193 Text feature [panels] present in test data point [True]
205 Text feature [panel] present in test data point [True]
207 Text feature [labeled] present in test data point [True]
221 Text feature [4d] present in test data point [True]
222 Text feature [matched] present in test data point [True]
224 Text feature [contain] present in test data point [True]
230 Text feature [alter] present in test data point [True]
234 Text feature [cultures] present in test data point [True]
242 Text feature [thereby] present in test data point [True]
252 Text feature [nuclear] present in test data point [True]
260 Text feature [mapping] present in test data point [True]
262 Text feature [define] present in test data point [True]
265 Text feature [order] present in test data point [True]
268 Text feature [reactions] present in test data point [True]
269 Text feature [mapped] present in test data point [True]
271 Text feature [functions] present in test data point [True]
278 Text feature [interacting] present in test data point [True]
284 Text feature [notably] present in test data point [True]
287 Text feature [bovine] present in test data point [True]
290 Text feature [mutagenesis] present in test data point [True]
299 Text feature [whole] present in test data point [True]
302 Text feature [toward] present in test data point [True]
304 Text feature [page] present in test data point [True]
305 Text feature [zinc] present in test data point [True]
313 Text feature [deficient] present in test data point [True]
314 Text feature [particularly] present in test data point [True]
325 Text feature [1d] present in test data point [True]
326 Text feature [often] present in test data point [True]
331 Text feature [6b] present in test data point [True]
332 Text feature [affecting] present in test data point [True]
335 Text feature [display] present in test data point [True]
342 Text feature [upon] present in test data point [True]
343 Text feature [representation] present in test data point [True]
345 Text feature [side] present in test data point [True]
346 Text feature [deletion] present in test data point [True]
351 Text feature [patterns] present in test data point [True]
360 Text feature [via] present in test data point [True]
362 Text feature [mediate] present in test data point [True]
365 Text feature [close] present in test data point [True]
368 Text feature [region] present in test data point [True]
369 Text feature [calculated] present in test data point [True]
376 Text feature [translation] present in test data point [True]
384 Text feature [reduction] present in test data point [True]
385 Text feature [arrest] present in test data point [True]
387 Text feature [signal] present in test data point [True]
389 Text feature [transcriptional] present in test data point [True]
390 Text feature [infected] present in test data point [True]
```

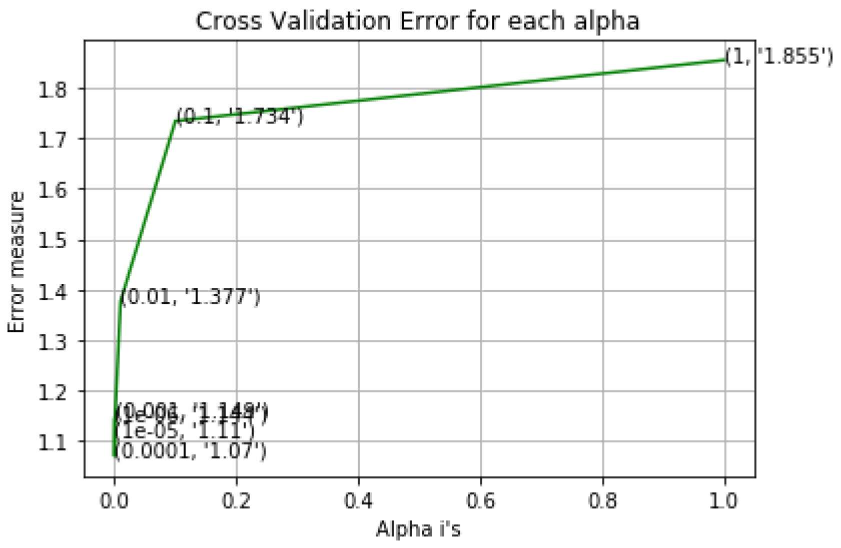
```
392 Text feature [directed] present in test data point [True]
394 Text feature [complexes] present in test data point [True]
396 Text feature [species] present in test data point [True]
397 Text feature [early] present in test data point [True]
401 Text feature [hotspot] present in test data point [True]
406 Text feature [parallel] present in test data point [True]
412 Text feature [rt] present in test data point [True]
413 Text feature [sirna] present in test data point [True]
414 Text feature [contact] present in test data point [True]
415 Text feature [39] present in test data point [True]
416 Text feature [nearly] present in test data point [True]
422 Text feature [specificity] present in test data point [True]
424 Text feature [difficult] present in test data point [True]
427 Text feature [future] present in test data point [True]
434 Text feature [identify] present in test data point [True]
435 Text feature [molecules] present in test data point [True]
436 Text feature [within] present in test data point [True]
438 Text feature [encoding] present in test data point [True]
442 Text feature [94] present in test data point [True]
449 Text feature [human] present in test data point [True]
452 Text feature [transiently] present in test data point [True]
454 Text feature [inactivation] present in test data point [True]
455 Text feature [technologies] present in test data point [True]
456 Text feature [actin] present in test data point [True]
458 Text feature [reveal] present in test data point [True]
460 Text feature [pdb] present in test data point [True]
464 Text feature [defined] present in test data point [True]
466 Text feature [carrying] present in test data point [True]
467 Text feature [knowledge] present in test data point [True]
468 Text feature [blot] present in test data point [True]
470 Text feature [cultured] present in test data point [True]
471 Text feature [contacts] present in test data point [True]
472 Text feature [noted] present in test data point [True]
474 Text feature [2c] present in test data point [True]
477 Text feature [structure] present in test data point [True]
478 Text feature [lane] present in test data point [True]
481 Text feature [roche] present in test data point [True]
482 Text feature [sds] present in test data point [True]
485 Text feature [mutational] present in test data point [True]
490 Text feature [structural] present in test data point [True]
491 Text feature [manufacturer] present in test data point [True]
495 Text feature [chain] present in test data point [True]
496 Text feature [located] present in test data point [True]
497 Text feature [versus] present in test data point [True]
499 Text feature [types] present in test data point [True]
Out of the top  500  features  97 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [79]:
```
# 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 : 1.1436661993662753
for alpha = 1e-05
Log Loss : 1.110246081839168
for alpha = 0.0001
Log Loss : 1.0699723996858146
for alpha = 0.001
Log Loss : 1.1491651471431763
for alpha = 0.01
Log Loss : 1.376851400038558
for alpha = 0.1
Log Loss : 1.7337695358436656
for alpha = 1
Log Loss : 1.8545138883498131
```

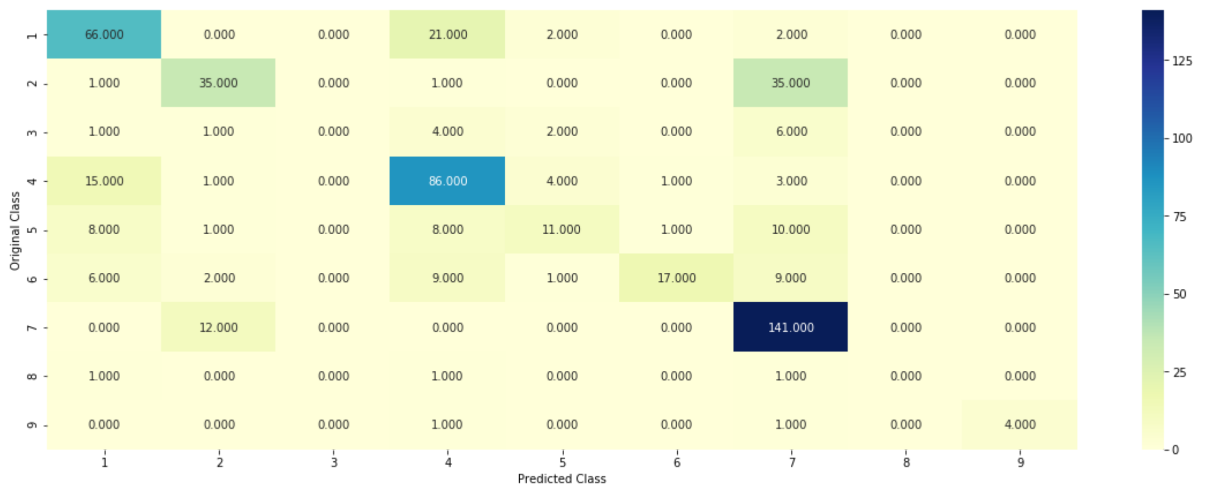


```
For values of best alpha =  0.0001 The train log loss is: 0.4167646465349769
For values of best alpha =  0.0001 The cross validation log loss is: 1.069972
3996858146
For values of best alpha =  0.0001 The test log loss is: 0.9912961061915728
```

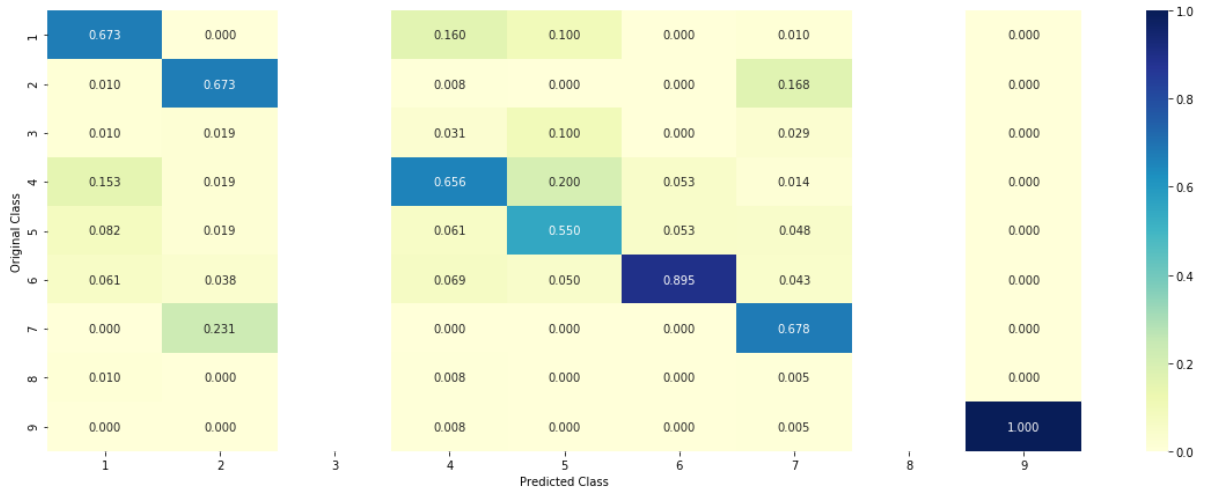
4.3.2.2. Testing model with best hyper parameters

In [80]:
```
# 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)
```

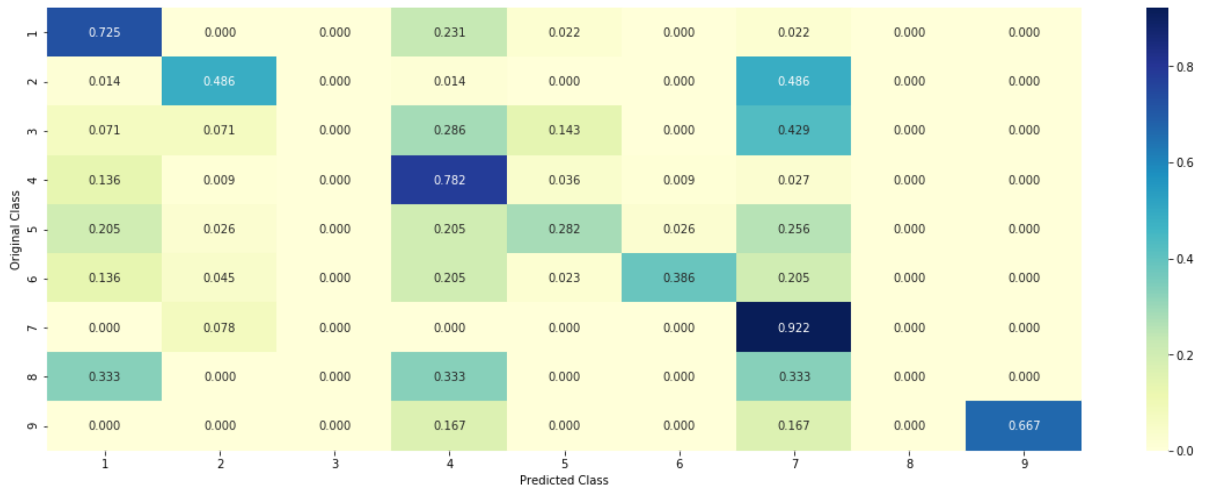
```
Log loss : 1.0699723996858146
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [81]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[4.900e-03 7.906e-01 5.300e-03 1.890e-02 2.86
0e-02 3.800e-03 1.459e-01
  1.800e-03 3.000e-04]]
Actual Class : 2
--------------------------------------------------
151 Text feature [play] present in test data point [True]
159 Text feature [actin] present in test data point [True]
164 Text feature [detection] present in test data point [True]
174 Text feature [additionally] present in test data point [True]
184 Text feature [discovery] present in test data point [True]
187 Text feature [viability] present in test data point [True]
195 Text feature [currently] present in test data point [True]
196 Text feature [response] present in test data point [True]
197 Text feature [tcga] present in test data point [True]
199 Text feature [90] present in test data point [True]
204 Text feature [01] present in test data point [True]
209 Text feature [achieved] present in test data point [True]
224 Text feature [300] present in test data point [True]
234 Text feature [subsequently] present in test data point [True]
235 Text feature [soft] present in test data point [True]
239 Text feature [especially] present in test data point [True]
253 Text feature [mek] present in test data point [True]
262 Text feature [rate] present in test data point [True]
265 Text feature [novel] present in test data point [True]
277 Text feature [step] present in test data point [True]
285 Text feature [need] present in test data point [True]
291 Text feature [rates] present in test data point [True]
295 Text feature [80] present in test data point [True]
302 Text feature [active] present in test data point [True]
306 Text feature [craf] present in test data point [True]
317 Text feature [weeks] present in test data point [True]
321 Text feature [patients] present in test data point [True]
324 Text feature [fusions] present in test data point [True]
325 Text feature [explain] present in test data point [True]
333 Text feature [plays] present in test data point [True]
335 Text feature [compound] present in test data point [True]
337 Text feature [05] present in test data point [True]
342 Text feature [eight] present in test data point [True]
348 Text feature [another] present in test data point [True]
359 Text feature [apoptosis] present in test data point [True]
360 Text feature [activates] present in test data point [True]
364 Text feature [finally] present in test data point [True]
365 Text feature [15] present in test data point [True]
366 Text feature [33] present in test data point [True]
368 Text feature [group] present in test data point [True]
374 Text feature [statistically] present in test data point [True]
380 Text feature [nras] present in test data point [True]
384 Text feature [light] present in test data point [True]
387 Text feature [knockdown] present in test data point [True]
395 Text feature [conformation] present in test data point [True]
399 Text feature [complete] present in test data point [True]
402 Text feature [important] present in test data point [True]
405 Text feature [selective] present in test data point [True]
406 Text feature [treatment] present in test data point [True]
407 Text feature [models] present in test data point [True]
412 Text feature [inactive] present in test data point [True]
```

```
414 Text feature [oncogenic] present in test data point [True]
416 Text feature [contribution] present in test data point [True]
427 Text feature [200] present in test data point [True]
430 Text feature [regression] present in test data point [True]
431 Text feature [deletions] present in test data point [True]
432 Text feature [www] present in test data point [True]
433 Text feature [36] present in test data point [True]
434 Text feature [hek293] present in test data point [True]
435 Text feature [member] present in test data point [True]
437 Text feature [isoform] present in test data point [True]
438 Text feature [part] present in test data point [True]
442 Text feature [harbor] present in test data point [True]
443 Text feature [carcinoma] present in test data point [True]
446 Text feature [different] present in test data point [True]
448 Text feature [001] present in test data point [True]
450 Text feature [clinically] present in test data point [True]
459 Text feature [concentrations] present in test data point [True]
461 Text feature [87] present in test data point [True]
469 Text feature [since] present in test data point [True]
472 Text feature [150] present in test data point [True]
476 Text feature [potentially] present in test data point [True]
478 Text feature [clinical] present in test data point [True]
480 Text feature [cases] present in test data point [True]
481 Text feature [harboring] present in test data point [True]
482 Text feature [molecular] present in test data point [True]
486 Text feature [types] present in test data point [True]
490 Text feature [loop] present in test data point [True]
491 Text feature [mainly] present in test data point [True]
493 Text feature [braf] present in test data point [True]
494 Text feature [kras] present in test data point [True]
499 Text feature [suggests] present in test data point [True]
Out of the top  500  features  82 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [82]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7333 0.0098 0.0036 0.2258 0.0074 0.0089 0.
0089 0.0013 0.0012]]
Actual Class : 4
--------------------------------------------------
36 Text feature [aggregation] present in test data point [True]
107 Text feature [surface] present in test data point [True]
185 Text feature [folding] present in test data point [True]
186 Text feature [plays] present in test data point [True]
188 Text feature [hydrophobic] present in test data point [True]
201 Text feature [panels] present in test data point [True]
202 Text feature [panel] present in test data point [True]
214 Text feature [labeled] present in test data point [True]
227 Text feature [4d] present in test data point [True]
228 Text feature [matched] present in test data point [True]
234 Text feature [contain] present in test data point [True]
239 Text feature [alter] present in test data point [True]
243 Text feature [thereby] present in test data point [True]
247 Text feature [interacting] present in test data point [True]
252 Text feature [nuclear] present in test data point [True]
257 Text feature [cultures] present in test data point [True]
261 Text feature [mapping] present in test data point [True]
265 Text feature [define] present in test data point [True]
273 Text feature [order] present in test data point [True]
275 Text feature [bovine] present in test data point [True]
276 Text feature [reactions] present in test data point [True]
285 Text feature [toward] present in test data point [True]
289 Text feature [zinc] present in test data point [True]
291 Text feature [notably] present in test data point [True]
296 Text feature [mutagenesis] present in test data point [True]
298 Text feature [mapped] present in test data point [True]
302 Text feature [affecting] present in test data point [True]
303 Text feature [functions] present in test data point [True]
305 Text feature [particularly] present in test data point [True]
309 Text feature [whole] present in test data point [True]
317 Text feature [page] present in test data point [True]
321 Text feature [representation] present in test data point [True]
323 Text feature [often] present in test data point [True]
326 Text feature [6b] present in test data point [True]
327 Text feature [display] present in test data point [True]
330 Text feature [deficient] present in test data point [True]
333 Text feature [upon] present in test data point [True]
342 Text feature [1d] present in test data point [True]
347 Text feature [side] present in test data point [True]
358 Text feature [reduction] present in test data point [True]
360 Text feature [39] present in test data point [True]
361 Text feature [directed] present in test data point [True]
365 Text feature [deletion] present in test data point [True]
366 Text feature [via] present in test data point [True]
370 Text feature [calculated] present in test data point [True]
371 Text feature [arrest] present in test data point [True]
372 Text feature [patterns] present in test data point [True]
374 Text feature [infected] present in test data point [True]
376 Text feature [mediate] present in test data point [True]
378 Text feature [region] present in test data point [True]
380 Text feature [sirna] present in test data point [True]
382 Text feature [translation] present in test data point [True]
```

```
383 Text feature [future] present in test data point [True]
388 Text feature [close] present in test data point [True]
391 Text feature [signal] present in test data point [True]
393 Text feature [species] present in test data point [True]
396 Text feature [transcriptional] present in test data point [True]
403 Text feature [hotspot] present in test data point [True]
404 Text feature [nearly] present in test data point [True]
405 Text feature [parallel] present in test data point [True]
407 Text feature [difficult] present in test data point [True]
409 Text feature [early] present in test data point [True]
412 Text feature [complexes] present in test data point [True]
413 Text feature [contact] present in test data point [True]
414 Text feature [identify] present in test data point [True]
424 Text feature [specificity] present in test data point [True]
425 Text feature [rt] present in test data point [True]
428 Text feature [encoding] present in test data point [True]
431 Text feature [molecules] present in test data point [True]
433 Text feature [pdb] present in test data point [True]
434 Text feature [contacts] present in test data point [True]
438 Text feature [noted] present in test data point [True]
440 Text feature [actin] present in test data point [True]
442 Text feature [21] present in test data point [True]
444 Text feature [human] present in test data point [True]
445 Text feature [cultured] present in test data point [True]
446 Text feature [reveal] present in test data point [True]
447 Text feature [blot] present in test data point [True]
451 Text feature [knowledge] present in test data point [True]
452 Text feature [technologies] present in test data point [True]
453 Text feature [inactivation] present in test data point [True]
455 Text feature [within] present in test data point [True]
458 Text feature [carrying] present in test data point [True]
459 Text feature [sds] present in test data point [True]
461 Text feature [transiently] present in test data point [True]
465 Text feature [94] present in test data point [True]
472 Text feature [partially] present in test data point [True]
475 Text feature [lane] present in test data point [True]
476 Text feature [05] present in test data point [True]
479 Text feature [figures] present in test data point [True]
481 Text feature [synthesis] present in test data point [True]
482 Text feature [versus] present in test data point [True]
483 Text feature [located] present in test data point [True]
484 Text feature [defined] present in test data point [True]
485 Text feature [structural] present in test data point [True]
489 Text feature [structure] present in test data point [True]
490 Text feature [chain] present in test data point [True]
492 Text feature [occurred] present in test data point [True]
493 Text feature [types] present in test data point [True]
494 Text feature [length] present in test data point [True]
497 Text feature [2c] present in test data point [True]
498 Text feature [24] present in test data point [True]
499 Text feature [codon] present in test data point [True]
Out of the top  500  features  103 are present in query point
```

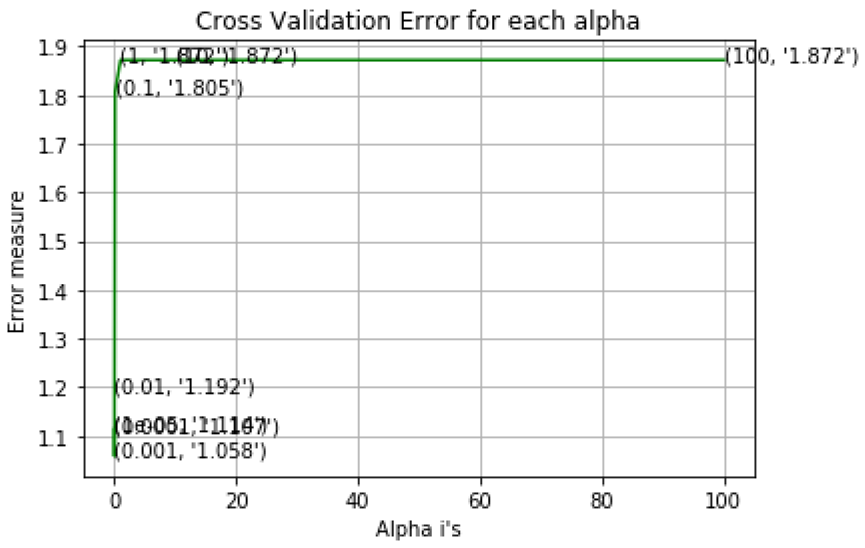
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [83]:
```
# 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 : 1.1138692490898465
for C = 0.0001
Log Loss : 1.1068879241628917
for C = 0.001
Log Loss : 1.0582677656551451
for C = 0.01
Log Loss : 1.1920664662171656
for C = 0.1
Log Loss : 1.8053870320702905
for C = 1
Log Loss : 1.8719365787609634
for C = 10
Log Loss : 1.8719366363135856
for C = 100
Log Loss : 1.8719367251272954
```

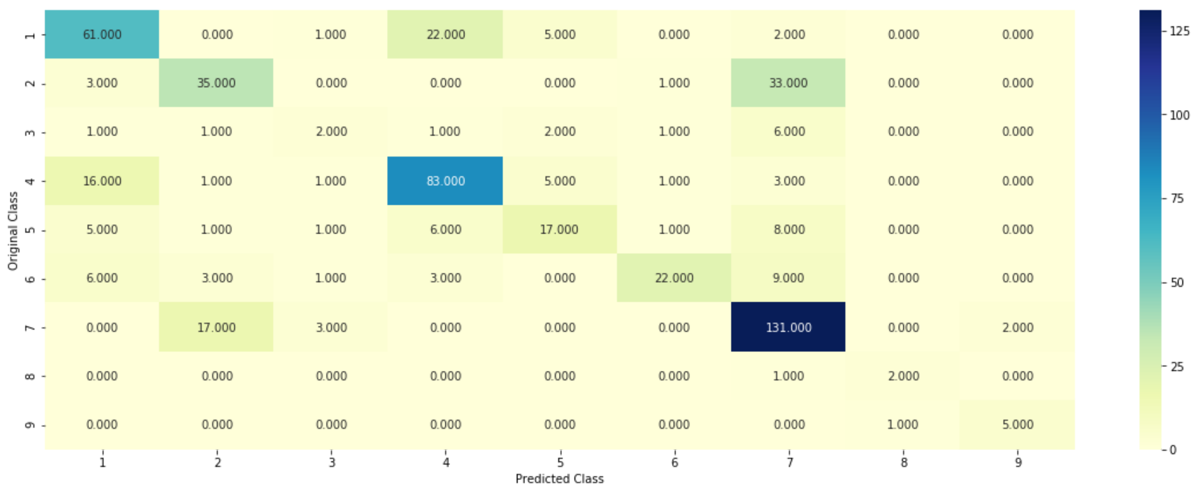


```
For values of best alpha =  0.001 The train log loss is: 0.5575811011119267
For values of best alpha =  0.001 The cross validation log loss is: 1.0582677
656551451
For values of best alpha =  0.001 The test log loss is: 1.0307389263855018
```

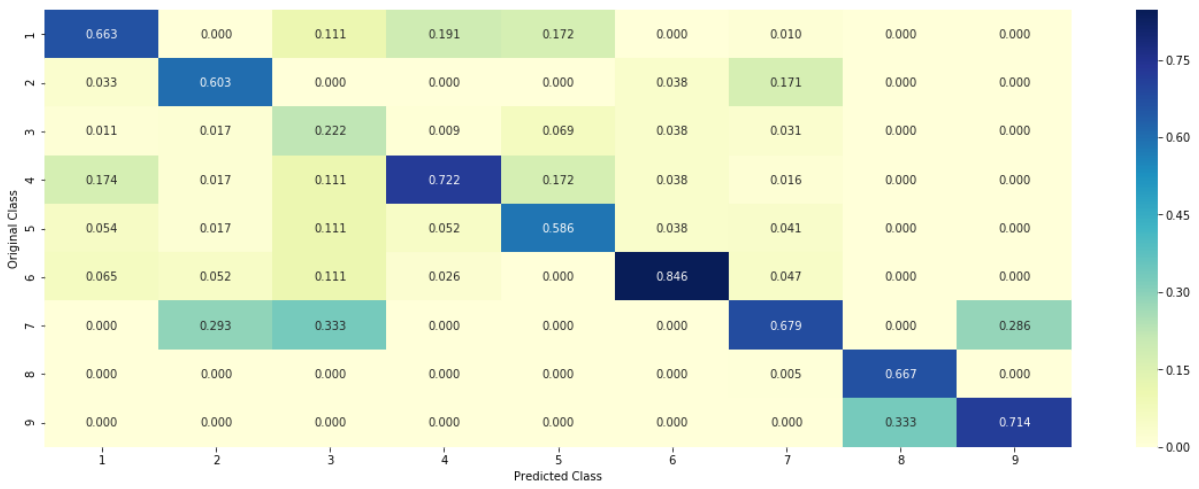
4.4.2. Testing model with best hyper parameters

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

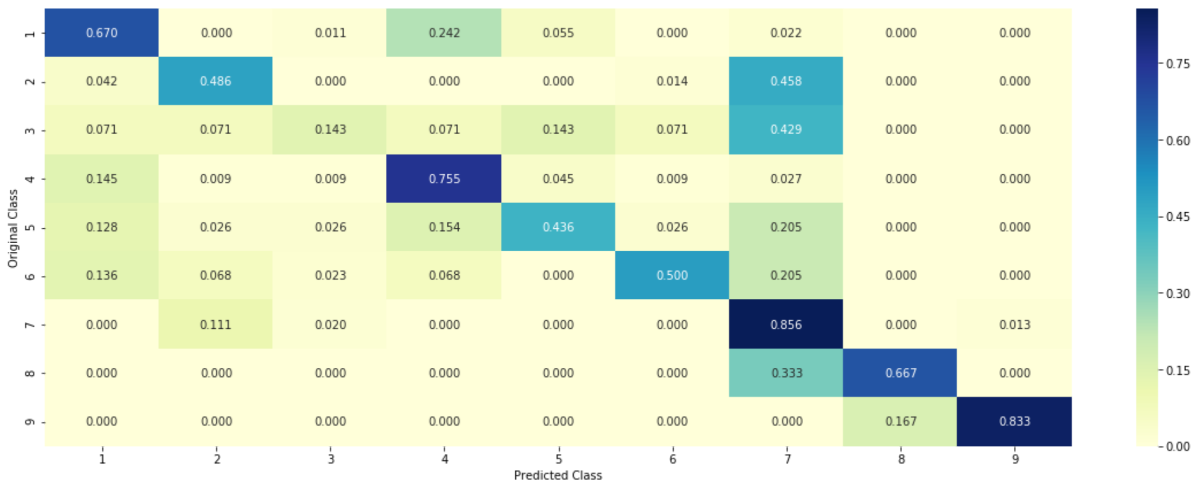
```
Log loss : 1.0582677656551451
Number of mis-classified points : 0.32706766917293234
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [85]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0234 0.7241 0.0201 0.047  0.0364 0.0102 0.
1345 0.003  0.0014]]
Actual Class : 2
--------------------------------------------------
16 Text feature [achieved] present in test data point [True]
198 Text feature [90] present in test data point [True]
199 Text feature [actin] present in test data point [True]
201 Text feature [response] present in test data point [True]
207 Text feature [weeks] present in test data point [True]
209 Text feature [currently] present in test data point [True]
210 Text feature [detection] present in test data point [True]
257 Text feature [additionally] present in test data point [True]
262 Text feature [rate] present in test data point [True]
267 Text feature [significance] present in test data point [True]
268 Text feature [play] present in test data point [True]
273 Text feature [soft] present in test data point [True]
275 Text feature [group] present in test data point [True]
278 Text feature [mek] present in test data point [True]
280 Text feature [craf] present in test data point [True]
283 Text feature [need] present in test data point [True]
285 Text feature [subsequently] present in test data point [True]
286 Text feature [oncogenic] present in test data point [True]
287 Text feature [deletions] present in test data point [True]
289 Text feature [discovery] present in test data point [True]
291 Text feature [01] present in test data point [True]
322 Text feature [step] present in test data point [True]
323 Text feature [patients] present in test data point [True]
328 Text feature [models] present in test data point [True]
331 Text feature [partial] present in test data point [True]
332 Text feature [every] present in test data point [True]
335 Text feature [harboring] present in test data point [True]
338 Text feature [treatment] present in test data point [True]
341 Text feature [rates] present in test data point [True]
342 Text feature [member] present in test data point [True]
344 Text feature [200] present in test data point [True]
345 Text feature [tcga] present in test data point [True]
350 Text feature [mainly] present in test data point [True]
351 Text feature [complete] present in test data point [True]
352 Text feature [novel] present in test data point [True]
353 Text feature [aacrjournals] present in test data point [True]
356 Text feature [suggests] present in test data point [True]
357 Text feature [87] present in test data point [True]
359 Text feature [knockdown] present in test data point [True]
362 Text feature [active] present in test data point [True]
370 Text feature [fusions] present in test data point [True]
372 Text feature [another] present in test data point [True]
374 Text feature [explain] present in test data point [True]
378 Text feature [potentially] present in test data point [True]
383 Text feature [effectively] present in test data point [True]
386 Text feature [reactions] present in test data point [True]
387 Text feature [xenograft] present in test data point [True]
389 Text feature [statistically] present in test data point [True]
393 Text feature [line] present in test data point [True]
396 Text feature [clinical] present in test data point [True]
397 Text feature [80] present in test data point [True]
398 Text feature [plays] present in test data point [True]
```

```
401 Text feature [similarly] present in test data point [True]
402 Text feature [primary] present in test data point [True]
403 Text feature [especially] present in test data point [True]
407 Text feature [15] present in test data point [True]
409 Text feature [disruption] present in test data point [True]
411 Text feature [squamous] present in test data point [True]
414 Text feature [47] present in test data point [True]
417 Text feature [regression] present in test data point [True]
418 Text feature [contribution] present in test data point [True]
421 Text feature [finally] present in test data point [True]
422 Text feature [small] present in test data point [True]
425 Text feature [specimens] present in test data point [True]
426 Text feature [important] present in test data point [True]
427 Text feature [based] present in test data point [True]
451 Text feature [36] present in test data point [True]
454 Text feature [hek293] present in test data point [True]
455 Text feature [05] present in test data point [True]
459 Text feature [braf] present in test data point [True]
460 Text feature [highly] present in test data point [True]
461 Text feature [cases] present in test data point [True]
464 Text feature [18] present in test data point [True]
474 Text feature [www] present in test data point [True]
476 Text feature [62] present in test data point [True]
479 Text feature [viability] present in test data point [True]
480 Text feature [conformation] present in test data point [True]
488 Text feature [resulted] present in test data point [True]
491 Text feature [nras] present in test data point [True]
492 Text feature [clear] present in test data point [True]
496 Text feature [acid] present in test data point [True]
499 Text feature [different] present in test data point [True]
Out of the top  500  features  82 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [86]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.691  0.0238 0.0088 0.182  0.0186 0.0107 0.
0577 0.0019 0.0055]]
Actual Class : 4
--------------------------------------------------
5 Text feature [aggregation] present in test data point [True]
13 Text feature [surface] present in test data point [True]
17 Text feature [folding] present in test data point [True]
18 Text feature [hydrophobic] present in test data point [True]
207 Text feature [matched] present in test data point [True]
208 Text feature [alter] present in test data point [True]
210 Text feature [panel] present in test data point [True]
211 Text feature [contain] present in test data point [True]
213 Text feature [thereby] present in test data point [True]
215 Text feature [bovine] present in test data point [True]
216 Text feature [4d] present in test data point [True]
276 Text feature [mutagenesis] present in test data point [True]
278 Text feature [1d] present in test data point [True]
279 Text feature [plays] present in test data point [True]
280 Text feature [labeled] present in test data point [True]
281 Text feature [mapped] present in test data point [True]
283 Text feature [interacting] present in test data point [True]
288 Text feature [mapping] present in test data point [True]
291 Text feature [mediate] present in test data point [True]
294 Text feature [toward] present in test data point [True]
295 Text feature [notably] present in test data point [True]
296 Text feature [order] present in test data point [True]
297 Text feature [side] present in test data point [True]
300 Text feature [affecting] present in test data point [True]
301 Text feature [whole] present in test data point [True]
303 Text feature [calculated] present in test data point [True]
304 Text feature [close] present in test data point [True]
337 Text feature [signal] present in test data point [True]
338 Text feature [panels] present in test data point [True]
339 Text feature [transiently] present in test data point [True]
340 Text feature [functions] present in test data point [True]
341 Text feature [particularly] present in test data point [True]
342 Text feature [complexes] present in test data point [True]
343 Text feature [via] present in test data point [True]
346 Text feature [contact] present in test data point [True]
350 Text feature [deficient] present in test data point [True]
351 Text feature [directed] present in test data point [True]
352 Text feature [cultured] present in test data point [True]
355 Text feature [molecules] present in test data point [True]
359 Text feature [sirna] present in test data point [True]
360 Text feature [reactions] present in test data point [True]
361 Text feature [bcl] present in test data point [True]
362 Text feature [human] present in test data point [True]
364 Text feature [define] present in test data point [True]
366 Text feature [page] present in test data point [True]
368 Text feature [within] present in test data point [True]
369 Text feature [nuclear] present in test data point [True]
371 Text feature [loss] present in test data point [True]
377 Text feature [region] present in test data point [True]
378 Text feature [length] present in test data point [True]
383 Text feature [sds] present in test data point [True]
386 Text feature [reduction] present in test data point [True]
```

```
388 Text feature [display] present in test data point [True]
389 Text feature [occurred] present in test data point [True]
390 Text feature [39] present in test data point [True]
393 Text feature [yet] present in test data point [True]
394 Text feature [lane] present in test data point [True]
395 Text feature [translation] present in test data point [True]
396 Text feature [patterns] present in test data point [True]
397 Text feature [manufacturer] present in test data point [True]
398 Text feature [lanes] present in test data point [True]
399 Text feature [cultures] present in test data point [True]
401 Text feature [molecule] present in test data point [True]
402 Text feature [identify] present in test data point [True]
403 Text feature [difficult] present in test data point [True]
404 Text feature [derived] present in test data point [True]
406 Text feature [roche] present in test data point [True]
408 Text feature [inactivation] present in test data point [True]
409 Text feature [types] present in test data point [True]
411 Text feature [structural] present in test data point [True]
414 Text feature [upon] present in test data point [True]
416 Text feature [6b] present in test data point [True]
418 Text feature [nearly] present in test data point [True]
419 Text feature [24] present in test data point [True]
423 Text feature [noted] present in test data point [True]
424 Text feature [often] present in test data point [True]
426 Text feature [deletion] present in test data point [True]
429 Text feature [established] present in test data point [True]
430 Text feature [instructions] present in test data point [True]
431 Text feature [knowledge] present in test data point [True]
432 Text feature [exposed] present in test data point [True]
458 Text feature [transcriptional] present in test data point [True]
459 Text feature [roles] present in test data point [True]
460 Text feature [2006] present in test data point [True]
464 Text feature [mutational] present in test data point [True]
466 Text feature [contacts] present in test data point [True]
467 Text feature [early] present in test data point [True]
468 Text feature [next] present in test data point [True]
469 Text feature [94] present in test data point [True]
470 Text feature [technologies] present in test data point [True]
474 Text feature [figures] present in test data point [True]
476 Text feature [2c] present in test data point [True]
477 Text feature [remain] present in test data point [True]
480 Text feature [reveal] present in test data point [True]
482 Text feature [species] present in test data point [True]
483 Text feature [statistically] present in test data point [True]
485 Text feature [greater] present in test data point [True]
488 Text feature [find] present in test data point [True]
491 Text feature [line] present in test data point [True]
492 Text feature [encoding] present in test data point [True]
494 Text feature [reduce] present in test data point [True]
496 Text feature [markedly] present in test data point [True]
497 Text feature [parallel] present in test data point [True]
499 Text feature [arrest] present in test data point [True]
Out of the top  500  features  104 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [87]:
```
# --------------------------------
# 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))
```

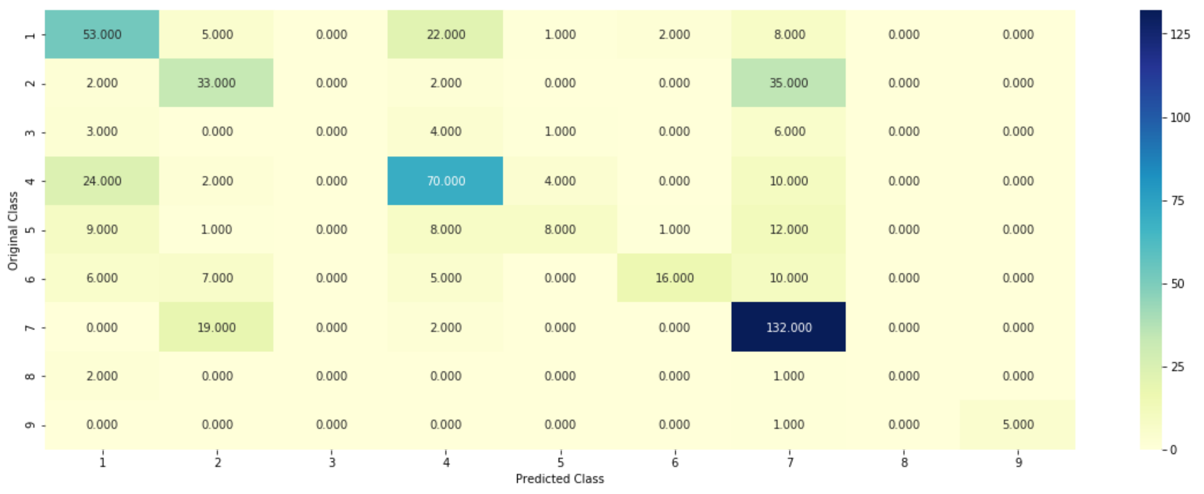
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''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 : 1.2142900745037262
for n_estimators = 100 and max depth =  10
Log Loss : 1.2363809740544436
for n_estimators = 200 and max depth =  5
Log Loss : 1.2012490095350232
for n_estimators = 200 and max depth =  10
Log Loss : 1.2215959087334425
for n_estimators = 500 and max depth =  5
Log Loss : 1.1854018309566863
for n_estimators = 500 and max depth =  10
Log Loss : 1.2122075113381592
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1805568607184735
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2053130179117517
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1766102486451362
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2032707031518974
For values of best estimator =  2000 The train log loss is: 0.871971401468898
2
For values of best estimator =  2000 The cross validation log loss is: 1.1766
102486451362
For values of best estimator =  2000 The test log loss is: 1.1704838170117897
```

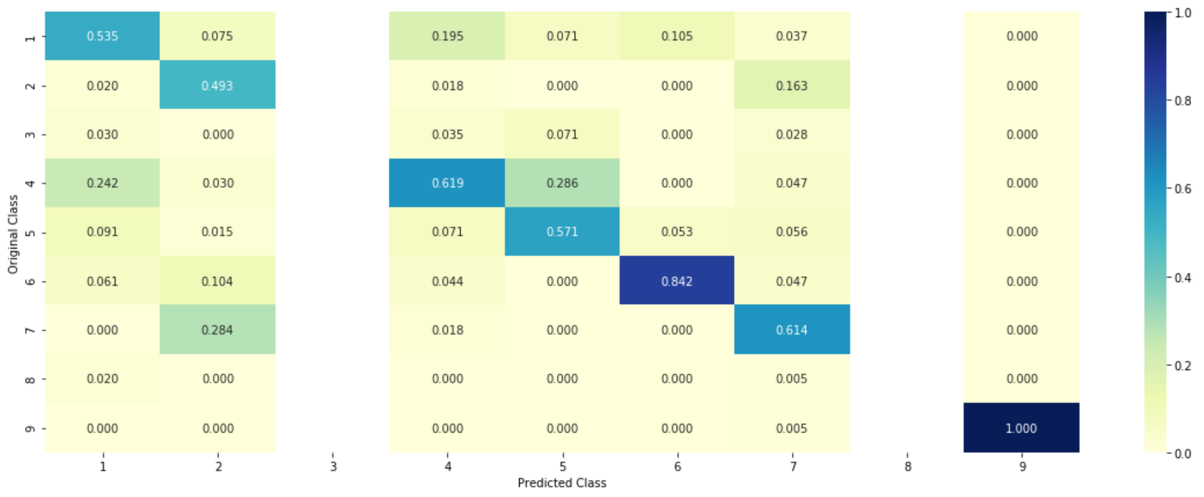
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [88]:
```
# --------------------------------
# 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)
```

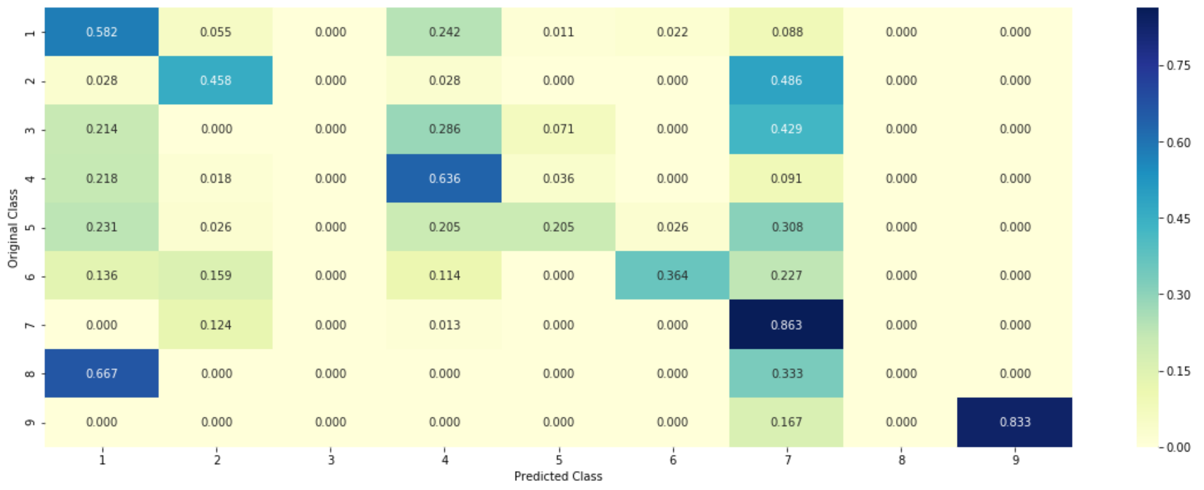
```
Log loss : 1.1766102486451366
Number of mis-classified points : 0.40413533383458647
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [89]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion=
'gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0268 0.4522 0.0204 0.0261 0.0366 0.0282 0.
4022 0.0047 0.0026]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
10 Text feature [function] present in test data point [True]
11 Text feature [oncogenic] present in test data point [True]
13 Text feature [signaling] present in test data point [True]
14 Text feature [constitutive] present in test data point [True]
15 Text feature [missense] present in test data point [True]
16 Text feature [treated] present in test data point [True]
17 Text feature [erk] present in test data point [True]
21 Text feature [loss] present in test data point [True]
25 Text feature [constitutively] present in test data point [True]
26 Text feature [kinases] present in test data point [True]
30 Text feature [57] present in test data point [True]
32 Text feature [activate] present in test data point [True]
34 Text feature [cells] present in test data point [True]
35 Text feature [erk1] present in test data point [True]
38 Text feature [protein] present in test data point [True]
39 Text feature [inhibition] present in test data point [True]
40 Text feature [cell] present in test data point [True]
43 Text feature [inhibited] present in test data point [True]
44 Text feature [variants] present in test data point [True]
47 Text feature [growth] present in test data point [True]
49 Text feature [expression] present in test data point [True]
53 Text feature [3t3] present in test data point [True]
64 Text feature [downstream] present in test data point [True]
66 Text feature [proliferation] present in test data point [True]
68 Text feature [drug] present in test data point [True]
72 Text feature [predicted] present in test data point [True]
75 Text feature [ic50] present in test data point [True]
76 Text feature [76] present in test data point [True]
77 Text feature [ovarian] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
84 Text feature [proteins] present in test data point [True]
85 Text feature [variant] present in test data point [True]
87 Text feature [response] present in test data point [True]
89 Text feature [patients] present in test data point [True]
94 Text feature [expressing] present in test data point [True]
97 Text feature [dose] present in test data point [True]
98 Text feature [catalytic] present in test data point [True]
99 Text feature [functions] present in test data point [True]
Out of the top  100  features  46 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [90]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4635 0.0344 0.0187 0.2812 0.0493 0.0427 0.
0963 0.0071 0.0069]]
Actuall Class : 4
--------------------------------------------------
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
10 Text feature [function] present in test data point [True]
11 Text feature [oncogenic] present in test data point [True]
12 Text feature [suppressor] present in test data point [True]
13 Text feature [signaling] present in test data point [True]
15 Text feature [missense] present in test data point [True]
16 Text feature [treated] present in test data point [True]
18 Text feature [receptor] present in test data point [True]
21 Text feature [loss] present in test data point [True]
25 Text feature [constitutively] present in test data point [True]
29 Text feature [stability] present in test data point [True]
32 Text feature [activate] present in test data point [True]
33 Text feature [classified] present in test data point [True]
34 Text feature [cells] present in test data point [True]
36 Text feature [therapy] present in test data point [True]
37 Text feature [yeast] present in test data point [True]
38 Text feature [protein] present in test data point [True]
39 Text feature [inhibition] present in test data point [True]
40 Text feature [cell] present in test data point [True]
41 Text feature [functional] present in test data point [True]
43 Text feature [inhibited] present in test data point [True]
47 Text feature [growth] present in test data point [True]
49 Text feature [expression] present in test data point [True]
50 Text feature [defective] present in test data point [True]
64 Text feature [downstream] present in test data point [True]
65 Text feature [truncating] present in test data point [True]
70 Text feature [membranes] present in test data point [True]
72 Text feature [predicted] present in test data point [True]
78 Text feature [oncogene] present in test data point [True]
80 Text feature [retained] present in test data point [True]
84 Text feature [proteins] present in test data point [True]
87 Text feature [response] present in test data point [True]
94 Text feature [expressing] present in test data point [True]
96 Text feature [mammalian] present in test data point [True]
97 Text feature [dose] present in test data point [True]
98 Text feature [catalytic] present in test data point [True]
99 Text feature [functions] present in test data point [True]
Out of the top  100  features  43 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [91]:
```
# --------------------------------
# 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))
```

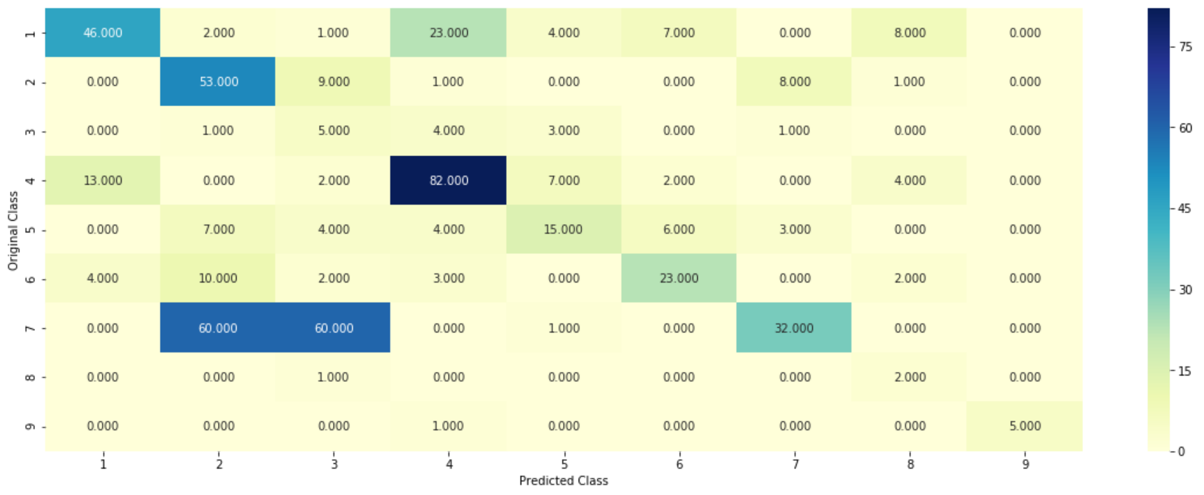
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (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 : 1.9946417680963449
for n_estimators = 10 and max depth =  3
Log Loss : 1.6081956151587693
for n_estimators = 10 and max depth =  5
Log Loss : 1.5194610750162683
for n_estimators = 10 and max depth =  10
Log Loss : 1.9350147897114234
for n_estimators = 50 and max depth =  2
Log Loss : 1.631147461800656
for n_estimators = 50 and max depth =  3
Log Loss : 1.4066572521958398
for n_estimators = 50 and max depth =  5
Log Loss : 1.3194378089916108
for n_estimators = 50 and max depth =  10
Log Loss : 1.7404854475271205
for n_estimators = 100 and max depth =  2
Log Loss : 1.5009875868141873
for n_estimators = 100 and max depth =  3
Log Loss : 1.4547623441425446
for n_estimators = 100 and max depth =  5
Log Loss : 1.3551136031573523
for n_estimators = 100 and max depth =  10
Log Loss : 1.6194245616889538
for n_estimators = 200 and max depth =  2
Log Loss : 1.5118566496859476
for n_estimators = 200 and max depth =  3
Log Loss : 1.4418670939662137
for n_estimators = 200 and max depth =  5
Log Loss : 1.37841841366657
for n_estimators = 200 and max depth =  10
Log Loss : 1.6094261107292454
for n_estimators = 500 and max depth =  2
Log Loss : 1.574462194650029
for n_estimators = 500 and max depth =  3
Log Loss : 1.500676121040427
for n_estimators = 500 and max depth =  5
Log Loss : 1.398060021180483
for n_estimators = 500 and max depth =  10
Log Loss : 1.6707736712244703
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5641364140512688
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5075383688375141
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3973204616175443
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6601179186842105
For values of best alpha =  50 The train log loss is: 0.0605555755196615
For values of best alpha =  50 The cross validation log loss is: 1.3194378089
916112
For values of best alpha =  50 The test log loss is: 1.3088068080609472
```

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

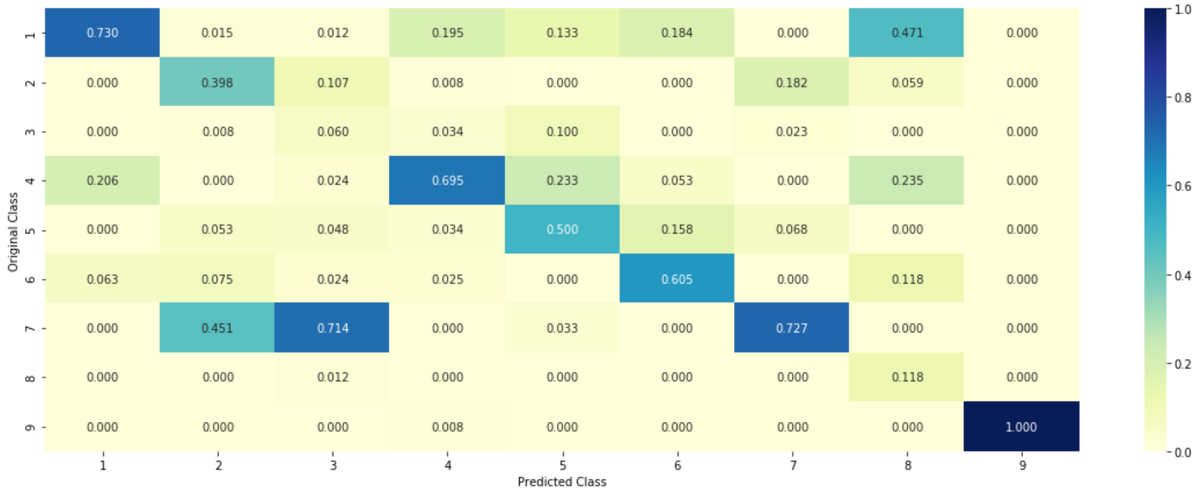
In [92]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# 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 : 1.319437808991611
Number of mis-classified points : 0.5056390977443609
-------------------- Confusion matrix --------------------
```



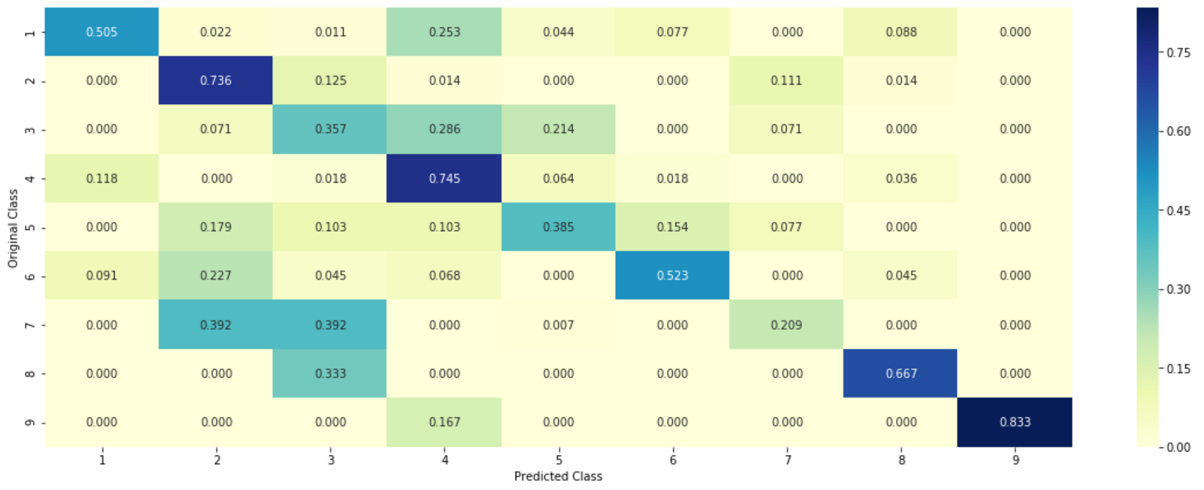
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	46.000	2.000	1.000	23.000	4.000	7.000	0.000	8.000	0.000
2	0.000	53.000	9.000	1.000	0.000	0.000	8.000	1.000	0.000
3	0.000	1.000	5.000	4.000	3.000	0.000	1.000	0.000	0.000
4	13.000	0.000	2.000	82.000	7.000	2.000	0.000	4.000	0.000
5	0.000	7.000	4.000	4.000	15.000	6.000	3.000	0.000	0.000
6	4.000	10.000	2.000	3.000	0.000	23.000	0.000	2.000	0.000
7	0.000	60.000	60.000	0.000	1.000	0.000	32.000	0.000	0.000
8	0.000	0.000	1.000	0.000	0.000	0.000	0.000	2.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	5.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.730	0.015	0.012	0.195	0.133	0.184	0.000	0.471	0.000
2	0.000	0.398	0.107	0.008	0.000	0.000	0.182	0.059	0.000
3	0.000	0.008	0.060	0.034	0.100	0.000	0.023	0.000	0.000
4	0.206	0.000	0.024	0.695	0.233	0.053	0.000	0.235	0.000
5	0.000	0.053	0.048	0.034	0.500	0.158	0.068	0.000	0.000
6	0.063	0.075	0.024	0.025	0.000	0.605	0.000	0.118	0.000
7	0.000	0.451	0.714	0.000	0.033	0.000	0.727	0.000	0.000
8	0.000	0.000	0.012	0.000	0.000	0.000	0.000	0.118	0.000
9	0.000	0.000	0.000	0.008	0.000	0.000	0.000	0.000	1.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.505	0.022	0.011	0.253	0.044	0.077	0.000	0.088	0.000
2	0.000	0.736	0.125	0.014	0.000	0.000	0.111	0.014	0.000
3	0.000	0.071	0.357	0.286	0.214	0.000	0.071	0.000	0.000
4	0.118	0.000	0.018	0.745	0.064	0.018	0.000	0.036	0.000
5	0.000	0.179	0.103	0.103	0.385	0.154	0.077	0.000	0.000
6	0.091	0.227	0.045	0.068	0.000	0.523	0.000	0.045	0.000
7	0.000	0.392	0.392	0.000	0.007	0.000	0.209	0.000	0.000
8	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.667	0.000
9	0.000	0.000	0.000	0.167	0.000	0.000	0.000	0.000	0.833

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [93]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion=
'gini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.009  0.6337 0.1982 0.0117 0.0213 0.0178 0.
0865 0.0157 0.0061]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation 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 [94]:
```
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")
```

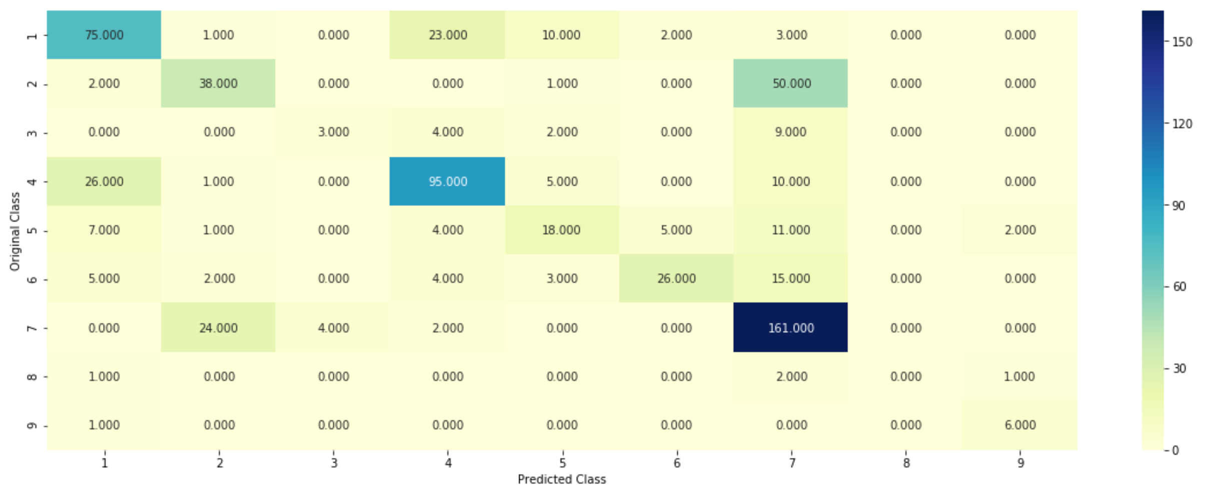
```
Predicted Class : 1
Predicted Class Probabilities: [[0.4    0.024  0.0845 0.3397 0.0247 0.0478 0.
0069 0.0428 0.0295]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation 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

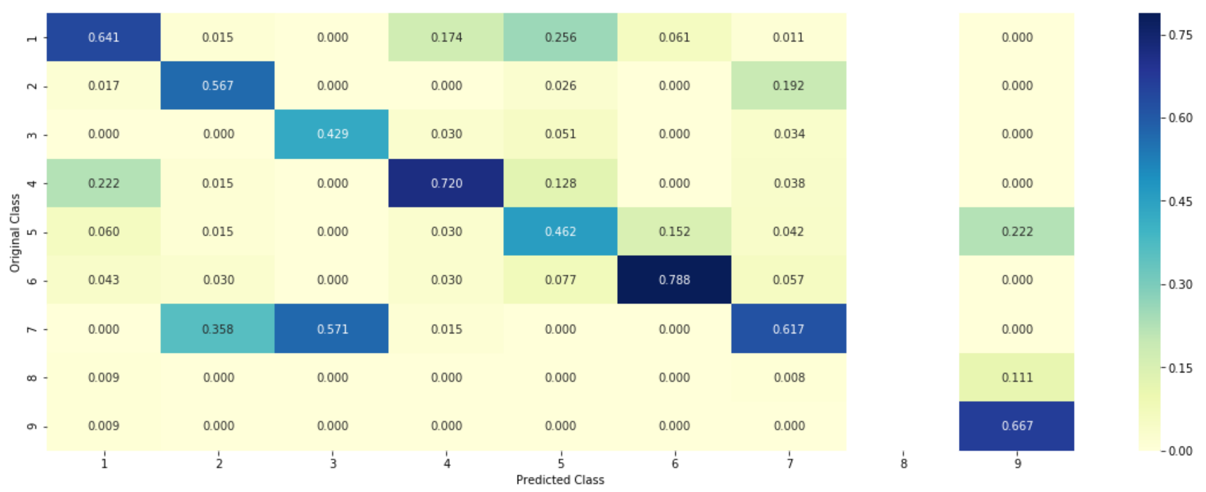
4.7.3 Maximum Voting classifier

In [96]:
```
#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))
```

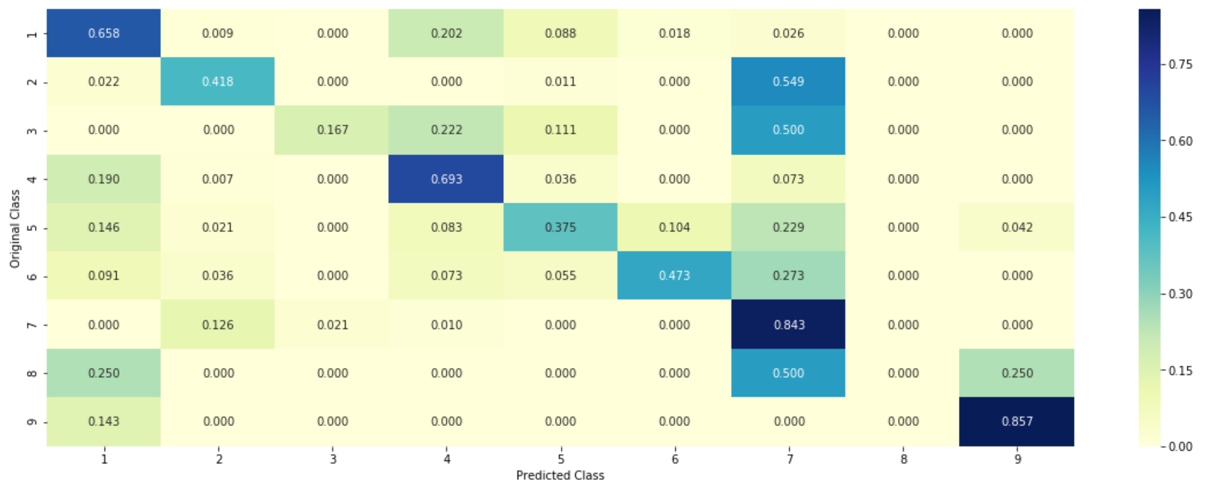
```
Log loss (train) on the VotingClassifier : 0.8415140147202367
Log loss (CV) on the VotingClassifier : 1.186780561927358
Log loss (test) on the VotingClassifier : 1.1794785002905461
Number of missclassified point : 0.3654135338345864 7
-------------------- Confusion matrix --------------------
```



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



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



Performance

In [1]:
```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names =["Models","Train","CV","Test","Misclassified(%)"]

x.add_row(["Naive Bayes (One hot coding)",0.57,1.14,1.23,0.37])
x.add_row(["KNN (Response)",0.65,1.04,1.03,0.36])
x.add_row(["LR(Class balanced) one hot coding",0.42,1.03,0.96,0.32])
x.add_row(["LR(Class unbalanced) one hot coding",0.41,1.06,0.99,0.32])
x.add_row(["Lr SVM one hot encoding",0.55,1.05,1.03,0.32])
x.add_row(["Random Forest one hot coding",0.87,1.17,1.17,0.40])
x.add_row(["Random Forest Response coding",0.60,1.31,1.30,0.49])
x.add_row(["Maximum Voting Classifier",0.84,1.18,1.17,0.36])

print(x)
```

```
+-------------------------------------+-------+------+------+----------------
--+
|                Models               | Train |  CV  | Test | Misclassified
(%) |
+-------------------------------------+-------+------+------+----------------
--+
|     Naive Bayes (One hot coding)    |  0.57 | 1.14 | 1.23 |       0.37
|
|            KNN (Response)           |  0.65 | 1.04 | 1.03 |       0.36
|
|   LR(Class balanced) one hot coding |  0.42 | 1.03 | 0.96 |       0.32
|
| LR(Class unbalanced) one hot coding |  0.41 | 1.06 | 0.99 |       0.32
|
|        Lr SVM one hot encoding      |  0.55 | 1.05 | 1.03 |       0.32
|
|     Random Forest one hot coding    |  0.87 | 1.17 | 1.17 |       0.4
|
|    Random Forest Response coding    |  0.6  | 1.31 | 1.3  |       0.49
|
|      Maximum Voting Classifier      |  0.84 | 1.18 | 1.17 |       0.36
|
+-------------------------------------+-------+------+------+----------------
--+
```