Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25

2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins

that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [40]:
```
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 [3]:
```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[3]:

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

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

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

3.1.2. Reading Text Data

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

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

Out[4]:

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

3.1.3. Preprocessing of text

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


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

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

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

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

```
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_tim
e, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 32.272412 seconds
```

In [7]:
```
#merging both gene_varis and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[7]:

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

In [8]:
```
result[result.isnull().any(axis=1)]
```

Out[8]:

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

	ID	Gene	Variation	Class	TEXT
2755	2755	BRAF	G596C	7	NaN

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

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

Out[10]:

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

3.1.4. Test, Train and Cross Validation Split

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

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

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

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

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

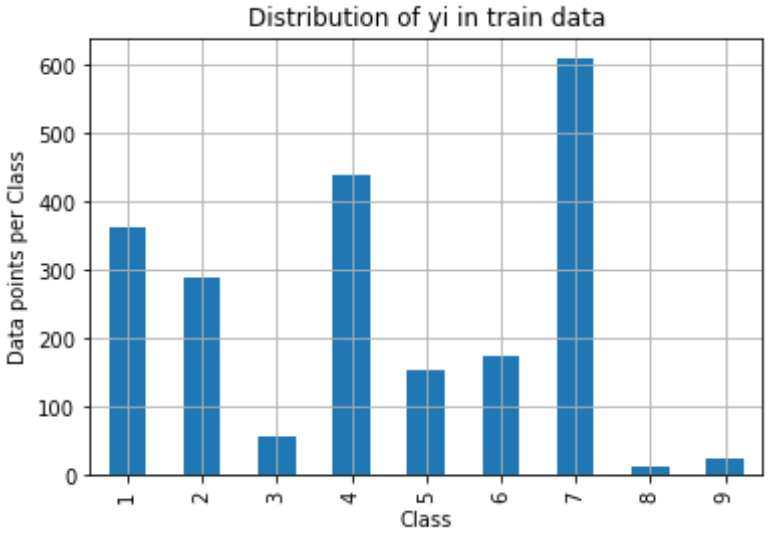
```
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
])
```

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

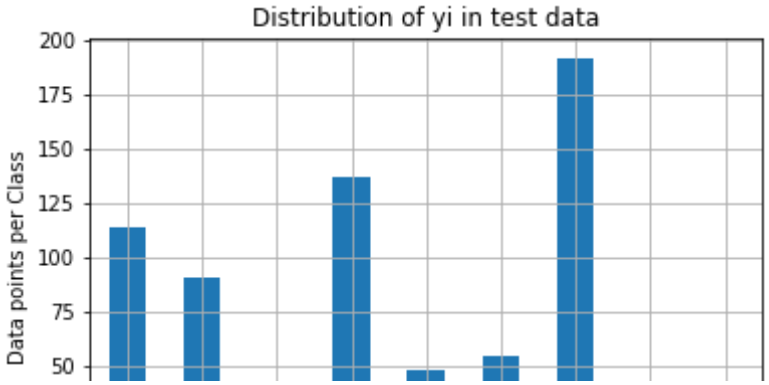
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

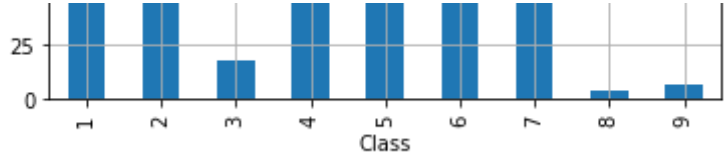
In [14]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index
()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distri
bution.values[i], '(', np.round((train_class_distribution.values[i]/tra
in_df.shape[0]*100), 3), '%)')


print('-'*80)
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/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distrib
ution.values[i], '(', np.round((test_class_distribution.values[i]/test_
df.shape[0]*100), 3), '%)')

print('-'*80)
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/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distribut
ion.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.sha
pe[0]*100), 3), '%)')
```

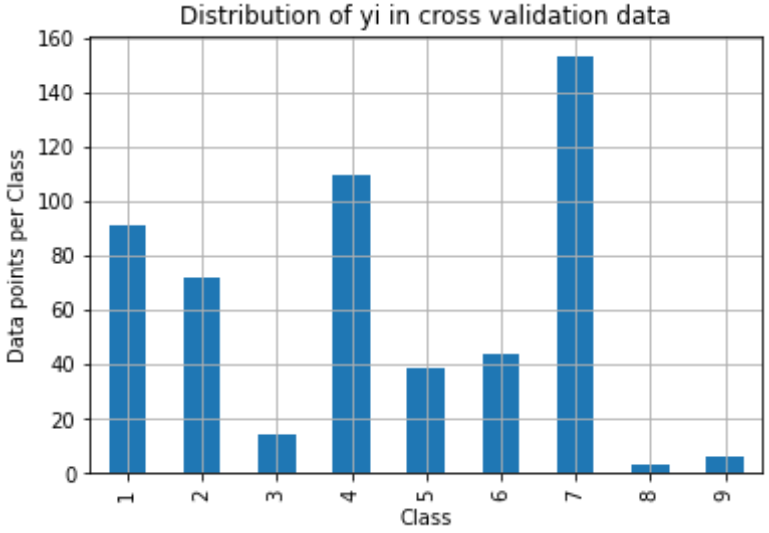



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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.

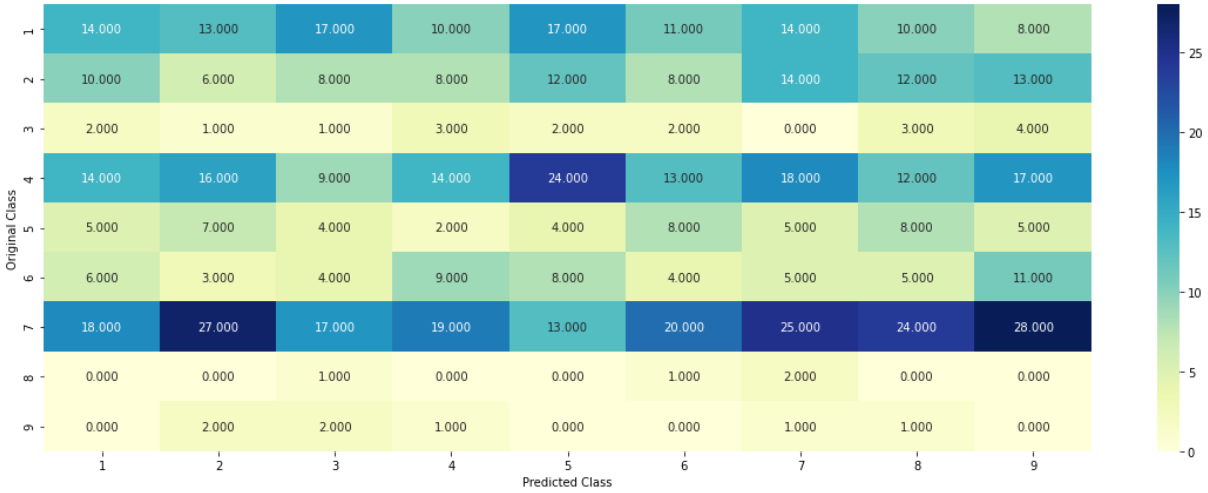
In [15]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 correspo
nds 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 element
s in that row
```

```
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
nds 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=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

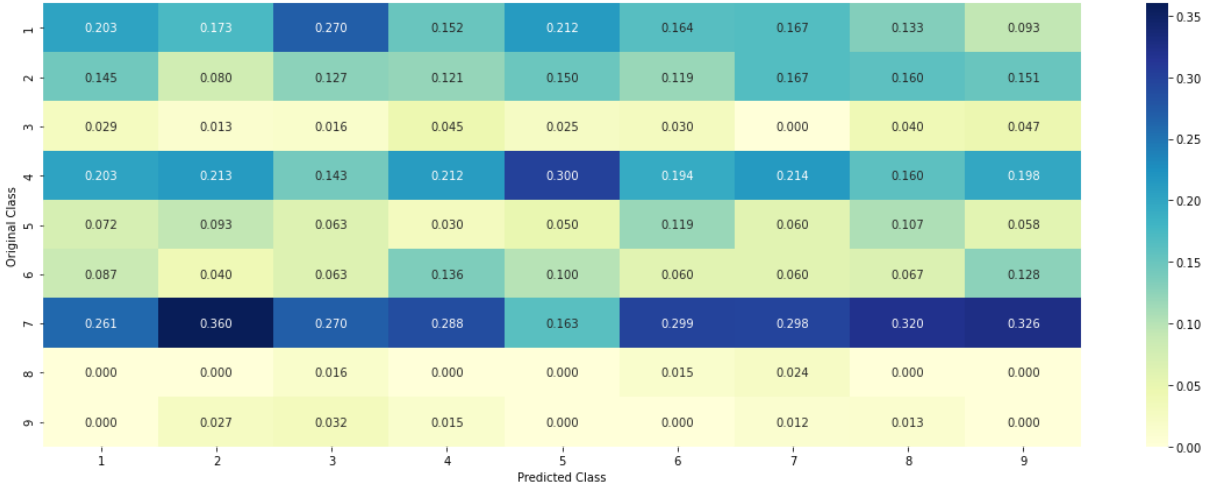
In [16]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
```

```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

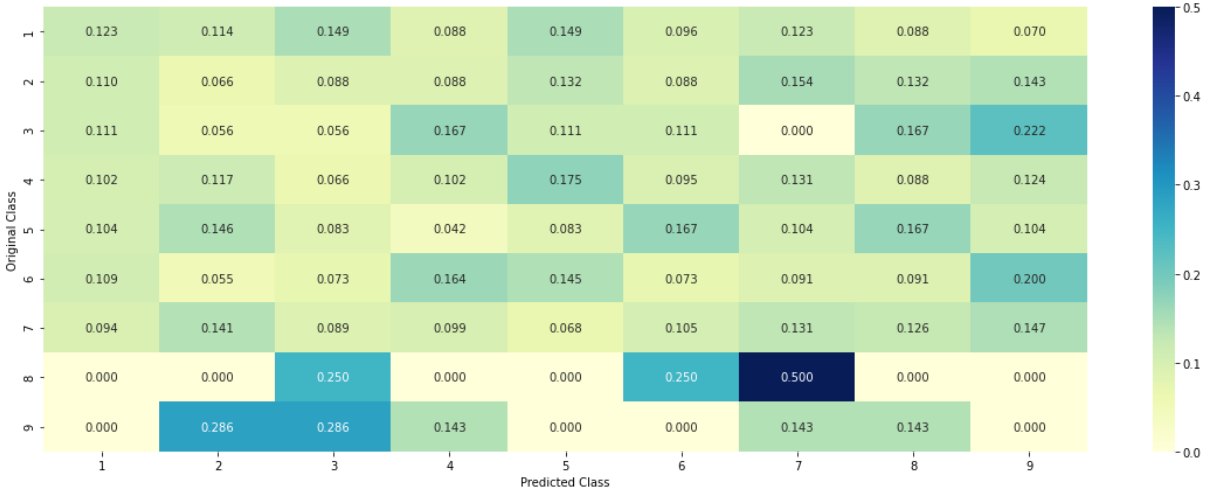
```
Log loss on Cross Validation Data using Random Model 2.456626724741349
Log loss on Test Data using Random Model 2.4569519713728787
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [17]:
```
# 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 feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation 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_f
ea'
```

```
# 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 arr
ay for each gene/variation
    gv_dict = dict()
```

```
    # denominator will contain the number of time that particular featu
re occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation b
elongs 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['Ge
ne']=='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[f
eature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of ti
me 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 a
s 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, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
```

```
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.068181818181818177, 0.068181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.0606060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
    #      ...
    #     }
    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_vari feature, it will contain the feature for each f
eature 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_fe
a
    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?

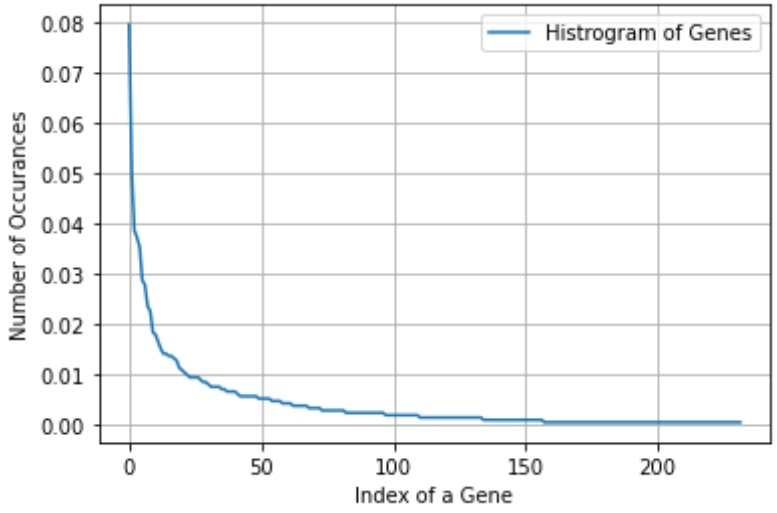
In [18]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1     169
TP53      107
EGFR       82
PTEN       79
BRCA2      75
KIT        61
BRAF       59
ALK        50
ERBB2      48
PIK3CA     39
Name: Gene, dtype: int64
```

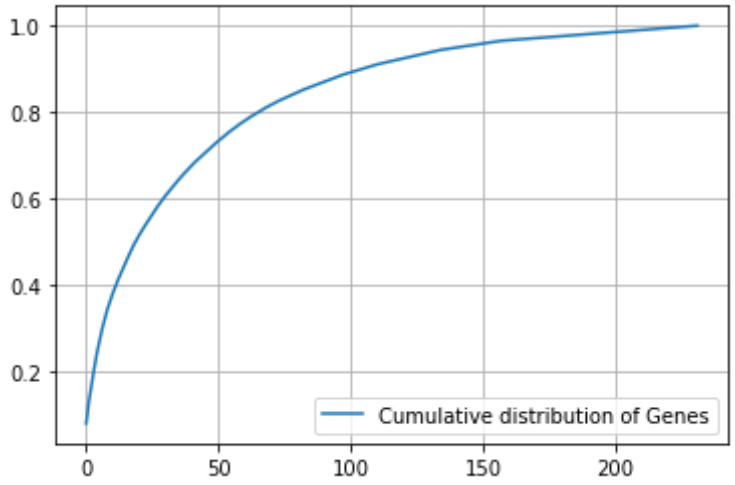
In [19]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

```
Ans: There are 232 different categories of genes in the train data, and
they are distibuted as follows
```

In [20]:
```
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 [21]:
```
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 [22]:
```
#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, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
```

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

In [23]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [24]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_TfidfVec = gene_vectorizer.fit_transform(train_df['G
ene'])
test_gene_feature_TfidfVec = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_TfidfVec = gene_vectorizer.transform(cv_df['Gene'])
```

In [25]:
```
train_df['Gene'].head()
```

Out[25]:
```
301     TMPRSS2
526        TP53
2345       JAK2
2231       PTEN
1868       MTOR
Name: Gene, dtype: object
```

In [26]:
```
gene_vectorizer.get_feature_names()
```

Out[26]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
```

```
                'araf',
                'arid1a',
                'arid1b',
                'arid2',
                'asxl1',
                'asxl2',
                'atm',
                'atr',
                'atrx',
                'aurka',
                'axin1',
                'b2m',
                'bap1',
                'bard1',
                'bcl10',
                'bcl2',
                'bcl2l11',
                'bcor',
                'braf',
                'brca1',
                'brca2',
                'brd4',
                'brip1',
                'btk',
                'card11',
                'carm1',
                'casp8',
                'cbl',
                'ccnd1',
                'ccnd2',
                'ccnd3',
                'ccne1',
                'cdh1',
                'cdk12',
                'cdk4',
                'cdk6',
                'cdk8',
                'cdkn1a',
                'cdkn1b',
```

```
                'cdkn2a',
                'cdkn2b',
                'cdkn2c',
                'cebpa',
                'chek2',
                'cic',
                'crebbp',
                'ctcf',
                'ctnnb1',
                'ddr2',
                'dicer1',
                'dnmt3a',
                'dnmt3b',
                'dusp4',
                'egfr',
                'elf3',
                'ep300',
                'epas1',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'errfi1',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fanca',
                'fancc',
                'fat1',
                'fbxw7',
                'fgf19',
                'fgf3',
                'fgfr1',
                'fgfr2',
```

```
                'fgfr3',
                'fgfr4',
                'flt3',
                'foxa1',
                'fubp1',
                'gata3',
                'gna11',
                'gnas',
                'h3f3a',
                'hist1h1c',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikbke',
                'inpp4b',
                'jak1',
                'jak2',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
                'klf4',
                'kmt2a',
                'kmt2b',
                'kmt2c',
                'kmt2d',
                'knstrn',
                'kras',
                'lats1',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
```

```
                'mdm4',
                'med12',
                'mef2b',
                'men1',
                'met',
                'mga',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'myod1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'nras',
                'nsd1',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pax8',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pik3r3',
                'pim1',
```

```
                'pms2',
                'pole',
                'ppm1d',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
                'rab35',
                'rac1',
                'rad21',
                'rad50',
                'rad51b',
                'rad51c',
                'rad51d',
                'raf1',
                'rara',
                'rasa1',
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rictor',
                'rit1',
                'ros1',
                'rras2',
                'runx1',
                'rxra',
                'sdhc',
                'setd2',
                'sf3b1',
                'shq1',
                'smad2',
                'smad3',
                'smad4',
```

```
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [27]:
```
print("train_gene_feature_TfidfVec is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_TfidfVec.shape)
```

```
train_gene_feature_TfidfVec is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 232)
```

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

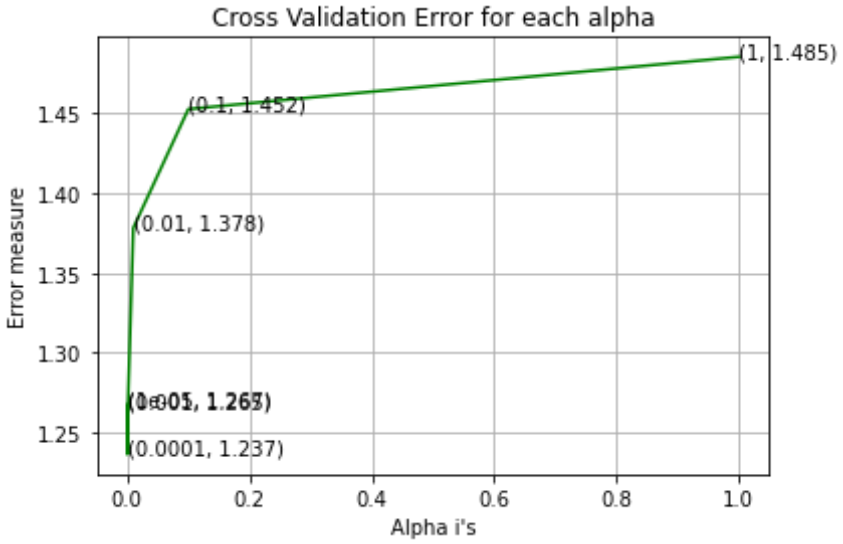
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [29]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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_TfidfVec, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_TfidfVec, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_TfidfVec)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
```

```
, predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_TfidfVec, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_TfidfVec, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.2672382819381838
For values of alpha =  0.0001 The log loss is: 1.2365461156891313
For values of alpha =  0.001 The log loss is: 1.2654900343756936
For values of alpha =  0.01 The log loss is: 1.3780754349871998
For values of alpha =  0.1 The log loss is: 1.4524147246169021
For values of alpha =  1 The log loss is: 1.4850956262407986
```




```
For values of best alpha =  0.0001 The train log loss is: 0.97596928958
76501
For values of best alpha =  0.0001 The cross validation log loss is: 1.
2365461156891313
For values of best alpha =  0.0001 The test log loss is: 1.173391001200
3297
```

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 [30]:
```
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'])))].shap
e[0]

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

```
Q6. How many data points in Test and CV datasets are covered by the  23
2  genes in train dataset?
Ans
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 512 out of  532 : 96.2406015037594
```

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 [31]:
```
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))
```

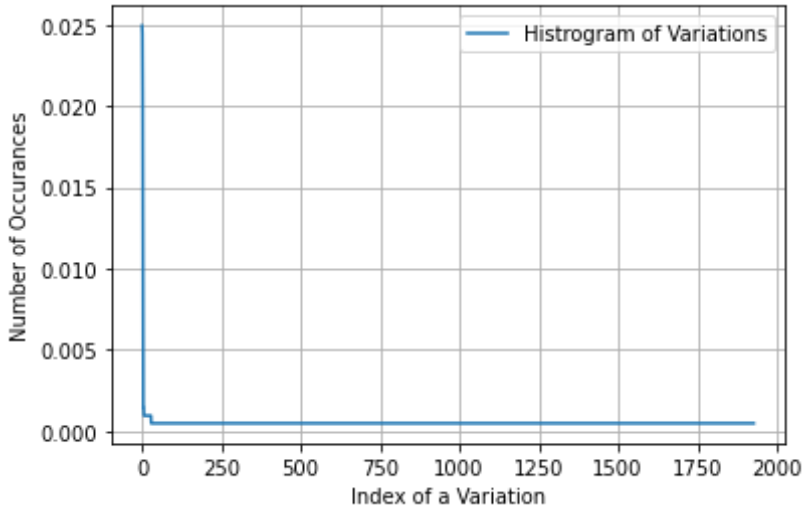
```
Number of Unique Variations : 1928
Deletion                  53
Truncating_Mutations      52
Amplification             44
Fusions                   24
Q61R                       3
```

```
G12V                    3
Q61L                    2
E542K                   2
R170W                   2
ETV6-NTRK3_Fusion       2
Name: Variation, dtype: int64
```

In [32]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

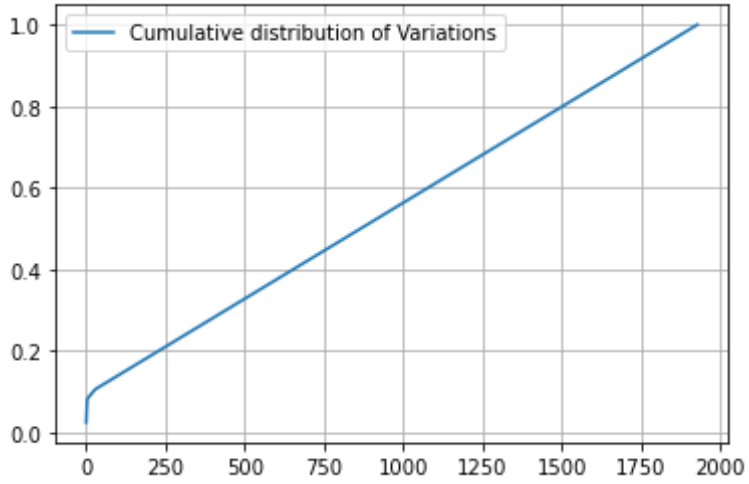
```
Ans: There are 1928 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [33]:
```
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 [34]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02495292 0.04943503 0.07015066 ... 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 [35]:
```
# alpha is used for laplace smoothing
alpha = 1
```

```
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [36]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [37]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_TfidfVec = variation_vectorizer.fit_transform(t
rain_df['Variation'])
test_variation_feature_TfidfVec = variation_vectorizer.transform(test_d
f['Variation'])
cv_variation_feature_TfidfVec = variation_vectorizer.transform(cv_df['V
ariation'])
```

In [40]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the tfidfvec. The shape of Variation feature:", train_variation_featur
e_TfidfVec.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the tf
idfvec. The shape of Variation feature: (2124, 1962)
```

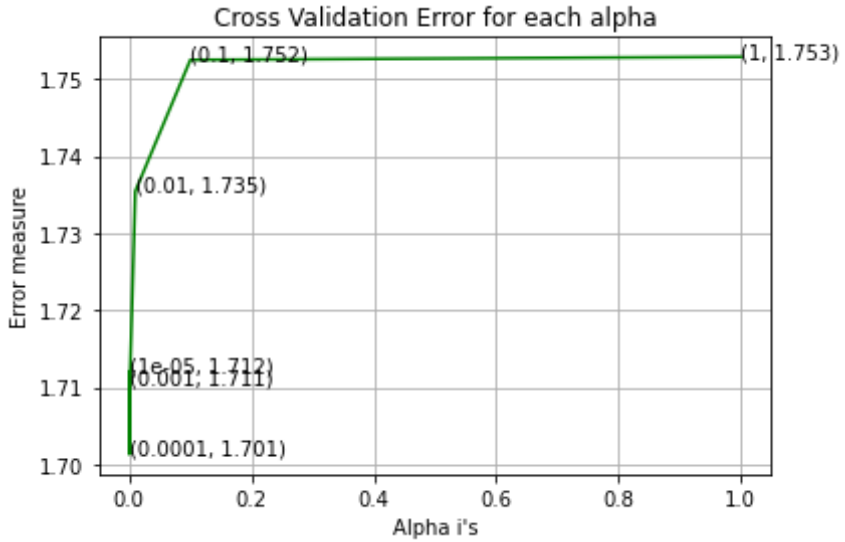
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [42]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_TfidfVec, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_TfidfVec)

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_TfidfVec, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_TfidfVec, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.7119530541920462
For values of alpha =  0.0001 The log loss is: 1.701312518802984
For values of alpha =  0.001 The log loss is: 1.7105077530367725
For values of alpha =  0.01 The log loss is: 1.735432451037612
For values of alpha =  0.1 The log loss is: 1.752475433177028
For values of alpha =  1 The log loss is: 1.7528700084545847
```




```
For values of best alpha =  0.0001 The train log loss is: 0.69001956047
10825
For values of best alpha =  0.0001 The cross validation log loss is: 1.
701312518802984
For values of best alpha =  0.0001 The test log loss is: 1.688050870633
1365
```

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 [43]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
```

```
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1928  genes in test and
cross validation data sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 57 out of  532 : 10.714285714285714
```

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 [44]:
```
# 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 [45]:
```
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 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [48]:
```
# building a TfidfVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_TfidfVec = text_vectorizer.fit_transform(train_df['T
EXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

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

In [49]:
```
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 [50]:
```
#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 [51]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [52]:
```
# don't forget to normalize every feature
train_text_feature_TfidfVec = normalize(train_text_feature_TfidfVec, ax
```

```
is=0)

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

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

In [53]:
```
#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 [54]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({250.92992197366917: 1, 181.85276148770382: 1, 138.031226170799
27: 1, 130.93140314723354: 1, 129.8722895278508: 1, 119.22490017878606:
1, 119.19194597164297: 1, 115.43437701423287: 1, 112.91394510164734: 1,
107.06740865722652: 1, 106.82166264643301: 1, 89.61853768463159: 1, 88.
53223334589772: 1, 88.1403372987104: 1, 82.79611854710849: 1, 81.024078
64368368: 1, 80.03322733340296: 1, 78.84071422825666: 1, 78.83015630332
791: 1, 77.30908199847549: 1, 75.62443115096544: 1, 75.32001600237822:
1, 72.75094574665107: 1, 71.01233090842973: 1, 68.93219943094388: 1, 6
7.50805632505266: 1, 67.14209241879489: 1, 65.24152626045918: 1, 65.194
92934493243: 1, 64.73073017420224: 1, 64.60209568691926: 1, 63.55001673
3969365: 1, 63.16185748249172: 1, 60.15113790365577: 1, 59.901482441829
38: 1, 58.43431491290546: 1, 56.48144309794958: 1, 56.262413170333474:
1, 55.0249392860198: 1, 52.33175084434598: 1, 51.34122282573513: 1, 49.
57728668081624: 1, 49.122062985872326: 1, 49.04684781239484: 1, 46.0960
2193592331: 1, 45.74038754614981: 1, 45.740199324238525: 1, 45.62997228
3954174: 1, 45.376060689211606: 1, 44.15647791056747: 1, 43.95052056444
689: 1, 43.65967833402683: 1, 43.40656931896882: 1, 43.39344504541775:
1, 43.3440155818004: 1, 42.980124101101865: 1, 42.90389599082742: 1, 4
```

2.08736478684494: 1, 41.7122521664582: 1, 41.531896269572464: 1, 41.042
573860527924: 1, 40.939778515628326: 1, 40.866869644655644: 1, 40.74248
038264938: 1, 39.854147092619954: 1, 39.398022342049316: 1, 39.31255277
981959: 1, 39.086452847264304: 1, 38.73483414853536: 1, 38.693608375688
18: 1, 38.68980410102161: 1, 37.75431670809225: 1, 37.53727520078585:
1, 36.994830770743135: 1, 36.91191249059054: 1, 36.90023248512929: 1, 3
6.515886745650604: 1, 35.86746977177168: 1, 35.7983173518079: 1, 35.763
83650524605: 1, 35.57243420707005: 1, 35.42099422472959: 1, 35.34979622
744814: 1, 35.33861389091477: 1, 35.16415504302825: 1, 34.7699906149516
8: 1, 34.49538602452278: 1, 33.25289488731924: 1, 33.07490468894552: 1,
32.99463737838315: 1, 32.764613931131436: 1, 32.75211661521408: 1, 32.6
3889758080459: 1, 32.54203736811748: 1, 32.46159293930247: 1, 32.421009
96081329: 1, 32.39555578018017: 1, 32.37145851125727: 1, 32.15568889931
4676: 1, 32.04108226226082: 1, 32.00114764264869: 1, 31.87104624996228
4: 1, 31.80927416525072: 1, 31.80736583514901: 1, 31.760181540841653:
1, 31.56070981192635: 1, 31.386210585755162: 1, 31.378497678566028: 1,
31.18869599972945: 1, 31.156244879612412: 1, 31.154548930021587: 1, 31.
122620038028572: 1, 31.070353606205: 1, 30.96960356992422: 1, 30.828877
213376852: 1, 30.597073065580926: 1, 30.418065664447507: 1, 30.39803187
557856: 1, 30.30838799662717: 1, 30.12469885249164: 1, 30.0661776975180
8: 1, 29.973352657967485: 1, 29.735029674139387: 1, 29.732934655702593:
1, 29.4865203316914: 1, 29.414119245610618: 1, 29.32400437666752: 1, 2
9.17247186392838: 1, 29.035672168736657: 1, 28.99942951028551: 1, 28.45
227158587888: 1, 28.380256964093054: 1, 28.254209063103424: 1, 27.92044
08865883: 1, 27.748976774780406: 1, 27.56113025598292: 1, 27.5478296729
24877: 1, 27.46071277120816: 1, 27.447011391298094: 1, 27.4291115006349
67: 1, 27.356375141838342: 1, 27.198215217448475: 1, 26.94602226866846
8: 1, 26.76232179590238: 1, 26.589363924296993: 1, 26.527829417697106:
1, 26.37168751397422: 1, 26.31142776067955: 1, 26.258896715863223: 1, 2
6.243386987243184: 1, 26.145381670924504: 1, 25.66630643366437: 1, 25.6
23006353871194: 1, 25.546608583763767: 1, 25.505492757605122: 1, 25.499
734251340485: 1, 25.442730492416: 1, 25.17183966824681: 1, 25.130185681
7651: 1, 25.071490885768576: 1, 25.053103949369966: 1, 24.9189059793399
57: 1, 24.91713786861122: 1, 24.913959446261625: 1, 24.722257088047513:
1, 24.65550761551569: 1, 24.58887089697564: 1, 24.51519864646439: 1, 2
4.474439076212633: 1, 24.317521181827367: 1, 24.244746644044632: 1, 24.
152948738015148: 1, 24.146959567074404: 1, 24.107770155507115: 1, 24.07
446706604825: 1, 23.945622255424226: 1, 23.693823890359766: 1, 23.59842
219813054: 1, 23.38572138072636: 1, 23.368603650572467: 1, 23.309245045

16324: 1, 23.229330215578603: 1, 23.147547935143063: 1, 23.093347449822
71: 1, 23.09008922355809: 1, 22.985277980613105: 1, 22.914088674073994:
1, 22.814191932399527: 1, 22.813228867901902: 1, 22.80903163191193: 1,
22.786810811505138: 1, 22.686001936341267: 1, 22.66648798730827: 1, 22.
607309247105146: 1, 22.59658317206194: 1, 22.548189683310074: 1, 22.523
253883353394: 1, 22.32740773821302: 1, 22.273429896010054: 1, 22.162488
312111382: 1, 22.137611695253486: 1, 22.079094931951573: 1, 22.05203169
210642: 1, 22.035958554139604: 1, 21.996805951510257: 1, 21.95596249763
644: 1, 21.893509543255124: 1, 21.885119384784154: 1, 21.8466969936935
4: 1, 21.665643520877385: 1, 21.62560267128145: 1, 21.535932949986485:
1, 21.4464540467378: 1, 21.439513706064027: 1, 21.430318388658257: 1, 2
1.41447843843295: 1, 21.40728949183619: 1, 21.301757197732606: 1, 21.28
2229421005134: 1, 21.25318723499383: 1, 21.209194924729726: 1, 21.20850
4758135337: 1, 21.19056796031065: 1, 21.15882062962858: 1, 21.096786896
26921: 1, 21.04895320295023: 1, 21.04024595005498: 1, 20.99021038839623
5: 1, 20.96290682813012: 1, 20.896291760992288: 1, 20.895474684268248:
1, 20.840495249158074: 1, 20.838594560093664: 1, 20.754302806820053: 1,
20.734242573137173: 1, 20.71342578901828: 1, 20.707491860636992: 1, 20.
65819985413726: 1, 20.61117555718509: 1, 20.580406034434066: 1, 20.5001
58637580697: 1, 20.490276931128644: 1, 20.48952471704066: 1, 20.4529542
2118553: 1, 20.391162715547058: 1, 20.306284558870086: 1, 20.2381987951
21892: 1, 20.224005112789232: 1, 20.116893471670803: 1, 20.093408425631
345: 1, 20.07495268658852: 1, 20.00129397119804: 1, 19.975639670688665:
1, 19.88130848543608: 1, 19.850199730747676: 1, 19.84241544701067: 1, 1
9.622430546040885: 1, 19.603375674235718: 1, 19.591697183696013: 1, 19.
58256540020509: 1, 19.536104747550503: 1, 19.442724517199228: 1, 19.428
13602902321: 1, 19.402226589746228: 1, 19.335408260860326: 1, 19.297745
471595878: 1, 19.285194955389517: 1, 19.221630354576767: 1, 19.14179526
2518368: 1, 19.131129791628222: 1, 19.101941899132015: 1, 19.0687058964
38512: 1, 19.061835113471364: 1, 19.056166161813323: 1, 19.028720369550
143: 1, 18.936514221432752: 1, 18.915771752586625: 1, 18.9155336438452
5: 1, 18.83076649343112: 1, 18.816362646917568: 1, 18.75230636991202:
1, 18.675804020295214: 1, 18.640339133168382: 1, 18.623136165585358: 1,
18.59531036338168: 1, 18.563842155362583: 1, 18.543761662664657: 1, 18.
432570784736665: 1, 18.400799938331435: 1, 18.368483342081596: 1, 18.34
6628209858302: 1, 18.307548734909894: 1, 18.163015777481906: 1, 18.1357
60313735673: 1, 18.007640593815196: 1, 18.00012809716: 1, 17.9738098055
39613: 1, 17.941893314198364: 1, 17.896934313300402: 1, 17.874281996219
334: 1, 17.845956517748295: 1, 17.810919581842896: 1, 17.80692117198436

```
7: 1, 17.805298143505365: 1, 17.79681309556911: 1, 17.7934931872325: 1,
17.782660043757165: 1, 17.76464366370555: 1, 17.741174805914284: 1, 17.
735526626555323: 1, 17.696084110003014: 1, 17.687584239143007: 1, 17.68
4306560896925: 1, 17.680015711745398: 1, 17.647514536902108: 1, 17.6052
79607282082: 1, 17.576267309876307: 1, 17.542599138151783: 1, 17.541213
010327336: 1, 17.532690695068336: 1, 17.51825964400783: 1, 17.493804484
261794: 1, 17.419212834855635: 1, 17.365412602300285: 1, 17.34138786312
438: 1, 17.338172873761696: 1, 17.33815815528912: 1, 17.19377259801942
5: 1, 17.183353252375593: 1, 17.16728493611862: 1, 17.160703658030005:
1, 17.13502138860578: 1, 17.113446953775174: 1, 17.109643627394576: 1,
17.09901972442902: 1, 17.0757454131002: 1, 17.069318119260064: 1, 17.04
8010851242076: 1, 17.004752243688124: 1, 16.940365832466817: 1, 16.9275
61818998306: 1, 16.802749806857886: 1, 16.80156179733922: 1, 16.7886197
38469862: 1, 16.78207167926122: 1, 16.747685808880064: 1, 16.7414188029
89326: 1, 16.738143132383417: 1, 16.703016373612744: 1, 16.663166755093
386: 1, 16.662210982495907: 1, 16.656266281347797: 1, 16.6167849847504
5: 1, 16.590792126647464: 1, 16.510767216282698: 1, 16.50160869866314:
1, 16.43551014666523: 1, 16.42507871687866: 1, 16.42016461986423: 1, 1
6.408300596566566: 1, 16.382609917037986: 1, 16.377284949100478: 1, 16.
360835068623935: 1, 16.350976350378826: 1, 16.277954258403717: 1, 16.23
3854918419322: 1, 16.224812780787985: 1, 16.224508636750336: 1, 16.1971
1993232095: 1, 16.158329657115292: 1, 16.14132574037359: 1, 16.11636452
2766577: 1, 16.082676174111644: 1, 16.064379219643296: 1, 16.0287049021
67007: 1, 15.988173617836072: 1, 15.98813791192857: 1, 15.9740847902666
27: 1, 15.943435005408304: 1, 15.900993434273577: 1, 15.90010754601306:
1, 15.896524108642554: 1, 15.885023685594586: 1, 15.879366762836707: 1,
15.84342337887603: 1, 15.840582173481538: 1, 15.77595025096992: 1, 15.7
70714083014292: 1, 15.649556310599923: 1, 15.640589168228866: 1, 15.617
715752429348: 1, 15.553858737687753: 1, 15.513608705968643: 1, 15.50784
5748186726: 1, 15.480611337166167: 1, 15.441009861747531: 1, 15.4157470
34961843: 1, 15.397759203122726: 1, 15.326213774305337: 1, 15.314894459
790954: 1, 15.308588703730154: 1, 15.300351197950672: 1, 15.26204243978
6057: 1, 15.240887587757419: 1, 15.231256953498011: 1, 15.1964359626842
94: 1, 15.195078127690309: 1, 15.1754340479902: 1, 15.17125275844945:
1, 15.164747553771731: 1, 15.160685970683344: 1, 15.115571112733566: 1,
15.10843454845793: 1, 15.103413661158621: 1, 15.095731308822492: 1, 15.
093007851254793: 1, 15.083741709131447: 1, 15.038813738533044: 1, 15.01
8927435633273: 1, 14.985983067945872: 1, 14.949199644504631: 1, 14.9459
84431375983: 1, 14.904700552134091: 1, 14.84723143144854: 1, 14.7925605
```

28583696: 1, 14.7916579026633: 1, 14.741265663271683: 1, 14.74068395204
0584: 1, 14.70677140339767: 1, 14.682210195621442: 1, 14.64763312910715
2: 1, 14.644878315927889: 1, 14.63210882615521: 1, 14.630643112983728:
1, 14.618733906033508: 1, 14.60763546390622: 1, 14.596464764002492: 1,
14.571236724594197: 1, 14.541856174655916: 1, 14.540594722912322: 1, 1
4.534028230427722: 1, 14.523330405164954: 1, 14.479981413603287: 1, 14.
441773047412914: 1, 14.41241107390147: 1, 14.411880986741327: 1, 14.408
456924161: 1, 14.353453104246894: 1, 14.352553193577693: 1, 14.30851268
570367: 1, 14.284730573653508: 1, 14.255614900873667: 1, 14.25277697038
9645: 1, 14.23810860441524: 1, 14.22989372558082: 1, 14.21729146996702:
1, 14.210309893070772: 1, 14.191974940421844: 1, 14.1255524794685: 1, 1
4.111518210999604: 1, 14.104547384461274: 1, 14.083979967648764: 1, 14.
045360019373808: 1, 14.023330773863146: 1, 13.998814340449425: 1, 13.98
3459350886328: 1, 13.945689937797322: 1, 13.944673713561883: 1, 13.9421
32768781436: 1, 13.921489329987828: 1, 13.874215204508012: 1, 13.822199
59932836: 1, 13.800817810725722: 1, 13.757152813246847: 1, 13.737156901
888875: 1, 13.727193877488224: 1, 13.721103092947493: 1, 13.70727846827
5236: 1, 13.702629500052478: 1, 13.614473022562114: 1, 13.5906738598703
44: 1, 13.584309465199668: 1, 13.577815620574427: 1, 13.54573673217049
6: 1, 13.541449107737359: 1, 13.53250561831646: 1, 13.532117070836108:
1, 13.511861266592081: 1, 13.468463550490787: 1, 13.458965480449125: 1,
13.457917717597734: 1, 13.37523523758901: 1, 13.349659350635832: 1, 13.
337823239553778: 1, 13.327499428230281: 1, 13.29744526542774: 1, 13.287
81539643561: 1, 13.258198023030968: 1, 13.257826070418183: 1, 13.199896
761561925: 1, 13.130536737624714: 1, 13.128578136162659: 1, 13.07959700
0768521: 1, 13.07794023717107: 1, 13.028764658527905: 1, 13.01538324612
818: 1, 12.999344471825573: 1, 12.962018345377583: 1, 12.93364556965923
7: 1, 12.93280676936484: 1, 12.897035395145204: 1, 12.885117175535289:
1, 12.881374716254788: 1, 12.878397614117299: 1, 12.872128395995691: 1,
12.864051841704129: 1, 12.828094938586247: 1, 12.82186004074975: 1, 12.
815780622009019: 1, 12.793881112050812: 1, 12.755330025945188: 1, 12.72
4603784418553: 1, 12.685291923294777: 1, 12.613081462710298: 1, 12.5945
31921054939: 1, 12.590414272496968: 1, 12.573376478532815: 1, 12.540275
511096674: 1, 12.539228800886022: 1, 12.526489600398671: 1, 12.52595135
6254318: 1, 12.515427960465399: 1, 12.486045769769538: 1, 12.4642727035
03884: 1, 12.439570548521994: 1, 12.43491382372246: 1, 12.4331760662388
26: 1, 12.429741766569066: 1, 12.420317540242506: 1, 12.41320817922755
5: 1, 12.39893597105459: 1, 12.393489978721197: 1, 12.38631520929924:
1, 12.37924689967499: 1, 12.371458717183414: 1, 12.357579411008574: 1,

12.346263064943882: 1, 12.3294544436115: 1, 12.32592456764915: 1, 12.32
4130569459907: 1, 12.31879497885668: 1, 12.302294287103349: 1, 12.29052
3722577701: 1, 12.27028664322707: 1, 12.262748087995478: 1, 12.25905806
9768777: 1, 12.238812888369157: 1, 12.222455309785921: 1, 12.2142724275
72029: 1, 12.207340252837167: 1, 12.206874265707377: 1, 12.128400452716
694: 1, 12.128395919354404: 1, 12.085334297482852: 1, 12.07471842885355
6: 1, 12.057580698698663: 1, 12.050104117856126: 1, 12.02255745297249:
1, 11.967546171004859: 1, 11.921864651808622: 1, 11.919281425931054: 1,
11.91223239445049: 1, 11.905051117829064: 1, 11.89236281534761: 1, 11.8
88178559672536: 1, 11.859683979465885: 1, 11.856867280751331: 1, 11.856
66705412283: 1, 11.840054947846955: 1, 11.826863919029735: 1, 11.817616
402541079: 1, 11.811632895844113: 1, 11.796216503010148: 1, 11.77559722
3436952: 1, 11.752717036461924: 1, 11.750633165410484: 1, 11.7501189856
47605: 1, 11.748006491413236: 1, 11.744454279542126: 1, 11.738832724755
387: 1, 11.720828246027182: 1, 11.679176694447708: 1, 11.6659977160069
1: 1, 11.665026946631775: 1, 11.664296167863728: 1, 11.623208264207173:
1, 11.617932361272642: 1, 11.610952711623513: 1, 11.579373543471702: 1,
11.539133249727914: 1, 11.537158847323637: 1, 11.528762685631353: 1, 1
1.525464452261637: 1, 11.501679495320111: 1, 11.490783501588469: 1, 11.
487611530259112: 1, 11.476684273778362: 1, 11.462214471762458: 1, 11.44
7112787102222: 1, 11.435301938118767: 1, 11.420800717147054: 1, 11.4105
5964803102: 1, 11.4025558817999: 1, 11.379571370170645: 1, 11.359694122
612614: 1, 11.338604205260715: 1, 11.337566766258488: 1, 11.33297851665
79: 1, 11.324822636088378: 1, 11.322969576501416: 1, 11.30444491060107
7: 1, 11.29885897542807: 1, 11.28461392939385: 1, 11.284305359594619:
1, 11.24893123006535: 1, 11.240027482372165: 1, 11.209140511751: 1, 11.
208743666537003: 1, 11.193258706014674: 1, 11.189425076783245: 1, 11.17
0207387424266: 1, 11.152509219964795: 1, 11.140595063985236: 1, 11.1253
34352321563: 1, 11.116971463489639: 1, 11.114366971067613: 1, 11.091557
478154154: 1, 11.084631281032108: 1, 11.041461526220873: 1, 11.03868592
5413178: 1, 11.031679990995581: 1, 11.008883276769005: 1, 11.0010544547
0388: 1, 11.000136396767305: 1, 10.997616204355136: 1, 10.9870093317973
01: 1, 10.986652355805893: 1, 10.985220934616205: 1, 10.98121418656984
2: 1, 10.948781531413161: 1, 10.930392553275977: 1, 10.927897873152858:
1, 10.915427992919659: 1, 10.887909865881875: 1, 10.886674618612018: 1,
10.869281915399942: 1, 10.866949695961134: 1, 10.856055776433664: 1, 1
0.810179045954985: 1, 10.784875916174494: 1, 10.746203824979979: 1, 10.
742053457047753: 1, 10.727907683163334: 1, 10.72055402740349: 1, 10.706
66887999585: 1, 10.702463832679236: 1, 10.69458405083617: 1, 10.6941652

2747408: 1, 10.690012870673668: 1, 10.6839749470779: 1, 10.657525116092
35: 1, 10.636398879019495: 1, 10.619615859305663: 1, 10.60846630029349
8: 1, 10.607229570301183: 1, 10.606516464968683: 1, 10.605254332021945:
1, 10.603665754996646: 1, 10.593650589104719: 1, 10.587275775389498: 1,
10.57513559225201: 1, 10.570310728450016: 1, 10.543935621479987: 1, 10.
541422642221873: 1, 10.538695731343784: 1, 10.527084350852606: 1, 10.51
7073512484444: 1, 10.439857816047917: 1, 10.43786771368984: 1, 10.42477
3165901339: 1, 10.418742741354738: 1, 10.406326718660498: 1, 10.4038227
77051522: 1, 10.395413879937129: 1, 10.383239734716064: 1, 10.381931242
298615: 1, 10.366749353263808: 1, 10.35810636187934: 1, 10.352908492656
683: 1, 10.346662170513802: 1, 10.346327005131437: 1, 10.33415397852972
4: 1, 10.217319541448775: 1, 10.210946691363345: 1, 10.145058925479848:
1, 10.14160439486035: 1, 10.141267712065702: 1, 10.131870537203158: 1,
10.126645407239428: 1, 10.123400236983658: 1, 10.123088323790329: 1, 1
0.103217785352122: 1, 10.100481330231196: 1, 10.090419737987519: 1, 10.
08226892392704: 1, 10.080569703627377: 1, 10.074850045197199: 1, 10.071
535611473609: 1, 10.057659279233143: 1, 10.05383179436949: 1, 10.044424
656204187: 1, 10.040227343662107: 1, 10.022800463806748: 1, 10.02060776
6550409: 1, 10.001793878190401: 1, 9.992596504358016: 1, 9.988975966366
477: 1, 9.979190948459005: 1, 9.975251945779872: 1, 9.970283889577779:
1, 9.959992119527596: 1, 9.959708317501656: 1, 9.957117392818786: 1, 9.
933911569926027: 1, 9.93168461158219: 1, 9.926817223654718: 1, 9.922628
690189045: 1, 9.897230145477291: 1, 9.875111631709172: 1, 9.84738688777
5452: 1, 9.844170544374911: 1, 9.822793125870955: 1, 9.82156052495436:
1, 9.81903445034533: 1, 9.816753305545962: 1, 9.785435644208583: 1, 9.7
72689223090847: 1, 9.767717641052366: 1, 9.757835963376694: 1, 9.740523
049573477: 1, 9.723607168630322: 1, 9.72264407184473: 1, 9.711606964431
464: 1, 9.701623929312651: 1, 9.690786641016231: 1, 9.688830349580973:
1, 9.66794417597226: 1, 9.66382290545063: 1, 9.663592078868064: 1, 9.61
2363695566875: 1, 9.60858166022539: 1, 9.606189159362819: 1, 9.60146269
9257462: 1, 9.596896204207507: 1, 9.567554903072466: 1, 9.5652914690405
19: 1, 9.564124083350297: 1, 9.56338674930483: 1, 9.562760415284687: 1,
9.559045822603698: 1, 9.543931029179266: 1, 9.537180494822596: 1, 9.515
18440806661: 1, 9.510977903269403: 1, 9.505059668106453: 1, 9.497498609
937985: 1, 9.474513217048322: 1, 9.473895929043573: 1, 9.44463202050808
1: 1, 9.435552077212032: 1, 9.434272641232933: 1, 9.43422451510326: 1,
9.434178035504868: 1, 9.412283063949044: 1, 9.40085094603303: 1, 9.3977
40381810944: 1, 9.395875110420144: 1, 9.387809653102295: 1, 9.386471639
95468: 1, 9.37780553016936: 1, 9.377628495027349: 1, 9.346280362610193:

1, 9.329174452702823: 1, 9.322160362123528: 1, 9.321850093546827: 1, 9.
282381556636452: 1, 9.25762051394449: 1, 9.256692613475773: 1, 9.251377
311517805: 1, 9.249345408730761: 1, 9.248303232077665: 1, 9.24354921965
7043: 1, 9.215019740678228: 1, 9.193834068881799: 1, 9.19188274268123:
1, 9.177160644124749: 1, 9.17629959634515: 1, 9.166026087045065: 1, 9.1
60931698684761: 1, 9.143368943116034: 1, 9.142389258142028: 1, 9.132641
125158127: 1, 9.132404096188408: 1, 9.110996615051489: 1, 9.11072877477
8277: 1, 9.095257586833819: 1, 9.0944791547259: 1, 9.094367917132715:
1, 9.08650587027217: 1, 9.081928085556166: 1, 9.062904479760471: 1, 9.0
44546522145131: 1, 9.043272623015897: 1, 9.036786888945754: 1, 9.034362
57370652: 1, 9.032263454266078: 1, 9.030129810059043: 1, 9.010518431130
02: 1, 8.994475244121007: 1, 8.992629542134678: 1, 8.988746491089579:
1, 8.98850095602688: 1, 8.980131722633203: 1, 8.97456179438193: 1, 8.97
1354530489482: 1, 8.966314753088628: 1, 8.963835490562607: 1, 8.9603860
77757597: 1, 8.956736133038708: 1, 8.95627942482858: 1, 8.9445220971577
83: 1, 8.936186790862836: 1, 8.923187958741215: 1, 8.897668766847982:
1, 8.882655218898265: 1, 8.855596138429929: 1, 8.850657862595883: 1, 8.
836225355812404: 1, 8.819179946355806: 1, 8.813621712004196: 1, 8.81059
7882673697: 1, 8.793933195778578: 1, 8.759627635570624: 1, 8.7314048397
80188: 1, 8.728906303970192: 1, 8.714210351652946: 1, 8.69200116894815
8: 1, 8.679310968758067: 1, 8.672722126038062: 1, 8.671358547985971: 1,
8.652051122853143: 1, 8.646638409084543: 1, 8.631377100620103: 1, 8.627
528568798288: 1, 8.623264073893901: 1, 8.621317340275146: 1, 8.61543520
3269692: 1, 8.603198704890762: 1, 8.599359339374388: 1, 8.5954833968752
63: 1, 8.594038994480549: 1, 8.591969703410742: 1, 8.59083487385124: 1,
8.583782589438739: 1, 8.563412030548548: 1, 8.56161564256978: 1, 8.5444
36773550457: 1, 8.538905604826484: 1, 8.531122340757818: 1, 8.507754855
97848: 1, 8.489287486427877: 1, 8.488432289659238: 1, 8.47731158617969
5: 1, 8.46230129965133: 1, 8.42998978862969: 1, 8.4031627142564: 1, 8.3
97944066073483: 1, 8.374670744787: 1, 8.371037959989204: 1, 8.364120004
86927: 1, 8.363628782340552: 1, 8.358170154816852: 1, 8.33608613549021:
1, 8.322812218411856: 1, 8.303147214383777: 1, 8.299103312485956: 1, 8.
289196794956103: 1, 8.285293083811764: 1, 8.26731658065807: 1, 8.263224
713907498: 1, 8.258414241895679: 1, 8.254475575793862: 1, 8.24850579385
3758: 1, 8.245387228024898: 1, 8.220921461633536: 1, 8.216785296090196:
1, 8.200128204016504: 1, 8.198542169622115: 1, 8.18949757762059: 1, 8.1
889709248525: 1, 8.177943176885439: 1, 8.140628055166648: 1, 8.09940130
8169199: 1, 8.08941032889416: 1, 8.070208391161968: 1, 8.06802318712053
6: 1, 8.065277477774616: 1, 8.062855607878726: 1, 8.050667783663933: 1,

```
8.03923306909571: 1, 8.03641437820244: 1, 8.032247210054841: 1, 8.01546
1607143356: 1, 8.014524085120007: 1, 7.9901377666321425: 1, 7.982288984
507987: 1, 7.963004314333506: 1, 7.9590729742141635: 1, 7.9539304471224
53: 1, 7.950294307491677: 1, 7.911776386178232: 1, 7.9069794682333026:
1, 7.900854862733382: 1, 7.899989231039216: 1, 7.884497180390469: 1, 7.
884252257708612: 1, 7.879503058206964: 1, 7.864463377188842: 1, 7.85303
3397912182: 1, 7.841325778272732: 1, 7.819887735701204: 1, 7.8155919097
94346: 1, 7.788668511605602: 1, 7.7809942339835745: 1, 7.77712473365224
3: 1, 7.773173822547524: 1, 7.749305683763201: 1, 7.746483619665634: 1,
7.736889267255843: 1, 7.733380635922334: 1, 7.682726725143473: 1, 7.665
884238377392: 1, 7.622468311667542: 1, 7.621867645258787: 1, 7.61300356
5475011: 1, 7.59764677570219: 1, 7.560186471510164: 1, 7.55254006602923
5: 1, 7.542832221308188: 1, 7.51233230752807: 1, 7.499424418532006: 1,
7.496626670782815: 1, 7.48863882513929: 1, 7.482705092885916: 1, 7.4763
18845086032: 1, 7.46903893694344: 1, 7.411452070060055: 1, 7.3773005344
58522: 1, 7.325109374137442: 1, 7.322491207160345: 1, 7.29405726623539
9: 1, 7.246101833434693: 1, 7.233706471505566: 1, 7.223834528434821: 1,
7.223416888676875: 1, 7.220808429222921: 1, 7.206317098294747: 1, 7.203
959971316927: 1, 7.178600975397407: 1, 7.123348412504785: 1, 7.12078066
9454264: 1, 7.091218478343598: 1, 7.046710700085157: 1, 7.037833765868
9: 1, 6.985825297982155: 1, 6.981530928630024: 1, 6.976005800914068: 1,
6.950416038432628: 1, 6.945625005449126: 1, 6.934104602444603: 1, 6.900
0203208341615: 1, 6.867148342178194: 1, 6.8391267979898585: 1, 6.824468
787987379: 1, 6.820206162942003: 1, 6.722713089486043: 1, 6.67448524443
4324: 1, 6.560869658140529: 1, 6.497945134738575: 1, 6.017157514512125:
1})
```

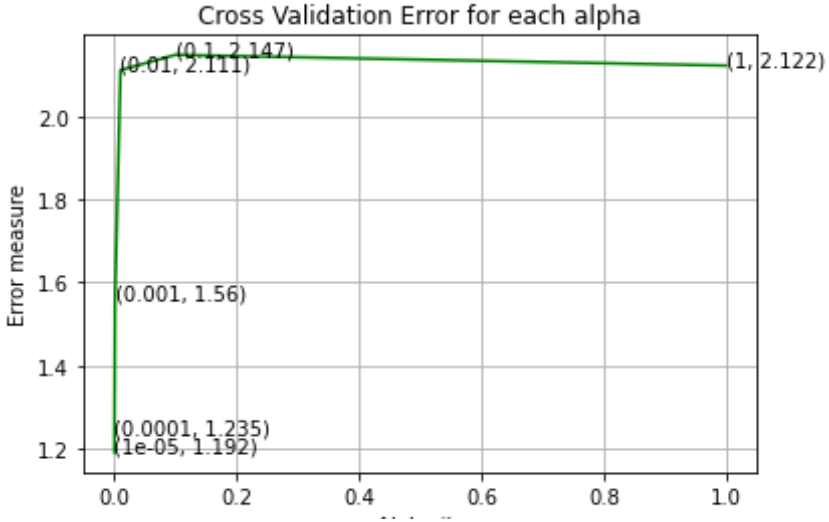
In [55]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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_TfidfVec, y_train)

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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_TfidfVec, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_TfidfVec, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.1918159537736366
For values of alpha =  0.0001 The log loss is: 1.2345921088141831
For values of alpha =  0.001 The log loss is: 1.5596825488992005
For values of alpha =  0.01 The log loss is: 2.1107535535866653
For values of alpha =  0.1 The log loss is: 2.1474246221009965
For values of alpha =  1 The log loss is: 2.1216416792001866
```



Alpha i's

```
For values of best alpha =  1e-05 The train log loss is: 0.680085315175
3827
For values of best alpha =  1e-05 The cross validation log loss is: 1.1
918159537736366
For values of best alpha =  1e-05 The test log loss is: 1.0805936588869
725
```

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

Ans. Yes, it seems like!

In [56]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [57]:
```
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 appe
ared in train data")
```

```
94.6 % of word of test data appeared in train data
94.1 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [35]:
```
#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 probabilit
ies belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [36]:
```
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 [60]:
```
# 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 = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)
```

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

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

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

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

Stacking the three types of features

In [61]:
```
# 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_TfidfVec = hstack((train_gene_feature_TfidfVec,train_var
iation_feature_TfidfVec))
test_gene_var_TfidfVec = hstack((test_gene_feature_TfidfVec,test_variat
ion_feature_TfidfVec))
cv_gene_var_TfidfVec = hstack((cv_gene_feature_TfidfVec,cv_variation_fe
ature_TfidfVec))

train_x_TfidfVec = hstack((train_gene_var_TfidfVec, train_text_feature_
TfidfVec)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_TfidfVec = hstack((test_gene_var_TfidfVec, test_text_feature_Tfi
dfVec)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_TfidfVec = hstack((cv_gene_var_TfidfVec, cv_text_feature_TfidfVec
)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, trai
n_text_feature_responseCoding))
```

```
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_t
ext_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [62]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_TfidfVec.shape)
print("(number of data points * number of features) in test data = ", t
est_x_TfidfVec.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_TfidfVec.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 31
94)
(number of data points * number of features) in test data =  (665, 319
4)
(number of data points * number of features) in cross validation data =
(532, 3194)
```

In [63]:
```
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 = ", t
est_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(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

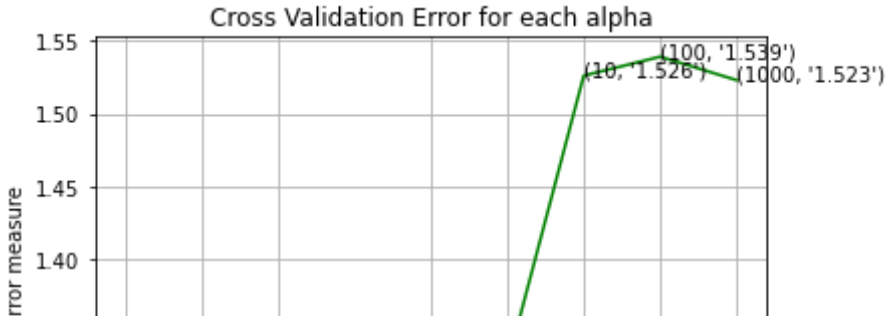
4.1.1.1. Hyper parameter tuning

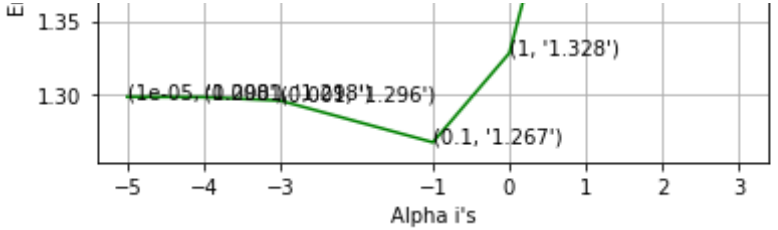
In [75]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to
 X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test v
ector 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
```

```
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# ----------------------


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

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


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

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

```
for alpha = 1e-05
Log Loss : 1.2980334512683402
for alpha = 0.0001
Log Loss : 1.2978479844033606
for alpha = 0.001
Log Loss : 1.2955077139332112
for alpha = 0.1
Log Loss : 1.2669010087452954
for alpha = 1
Log Loss : 1.3279683602824348
for alpha = 10
Log Loss : 1.5260625001627106
for alpha = 100
Log Loss : 1.5390693319335647
for alpha = 1000
Log Loss : 1.522992432481274
```





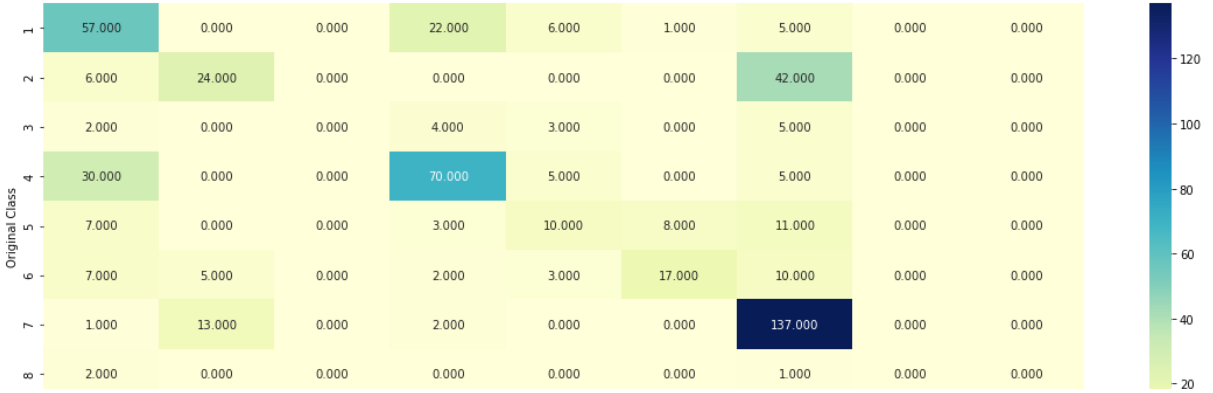

```
For values of best alpha =  0.1 The train log loss is: 0.53654962482270
35
For values of best alpha =  0.1 The cross validation log loss is: 1.266
9010087452954
For values of best alpha =  0.1 The test log loss is: 1.211332380749039
5
```

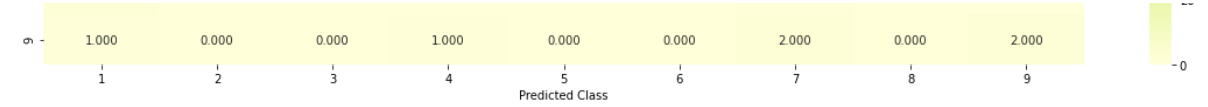
4.1.1.2. Testing the model with best hyper paramters

In [76]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to
 X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test v
ector 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
```

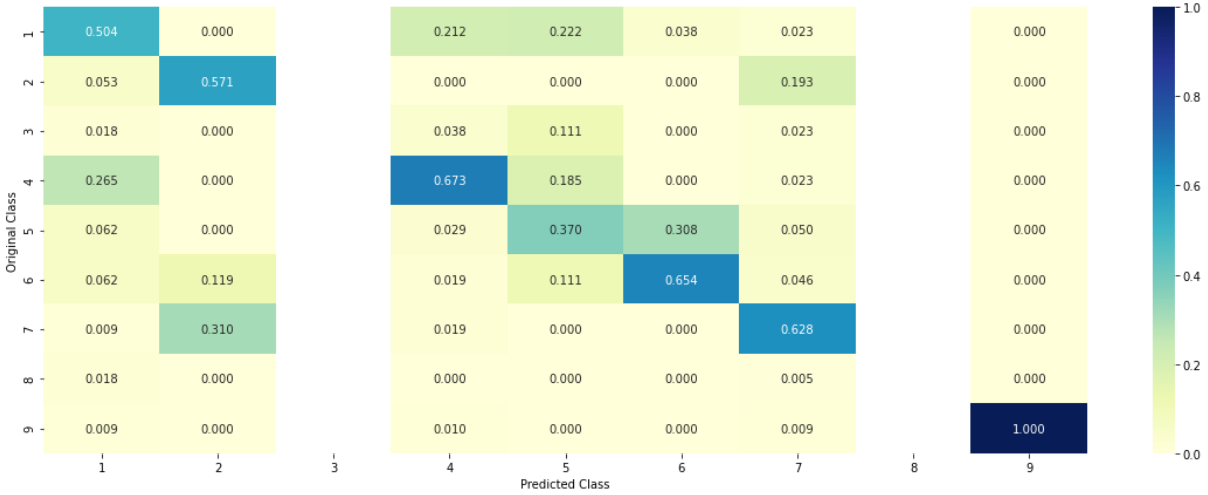
```
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------

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

```
Log Loss : 1.2669010087452954
Number of missclassified point : 0.4041353383458647
-------------------- Confusion matrix --------------------
```

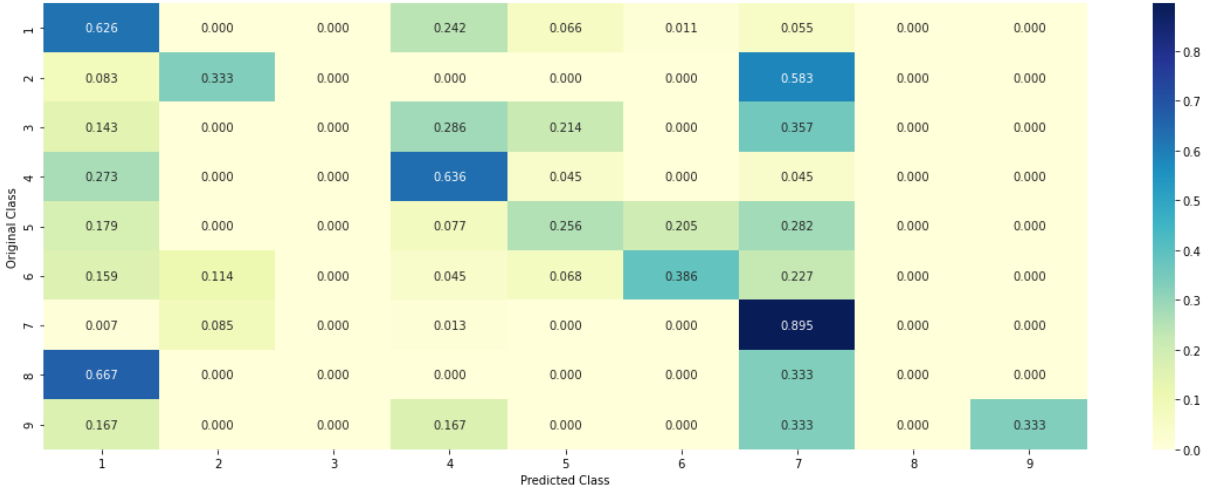




---------- Precision matrix (Columm Sum=1) ----------



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



4.1.1.3. Feature Importance, Correctly classified point

In [77]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_TfidfVec[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_TfidfVec[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0467 0.0447 0.0161 0.0504 0.0319 0.0
322 0.7685 0.0057 0.0037]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [67]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_TfidfVec[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_TfidfVec[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7563 0.0394 0.0171 0.0571 0.0345 0.0
346 0.051  0.0061 0.0039]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

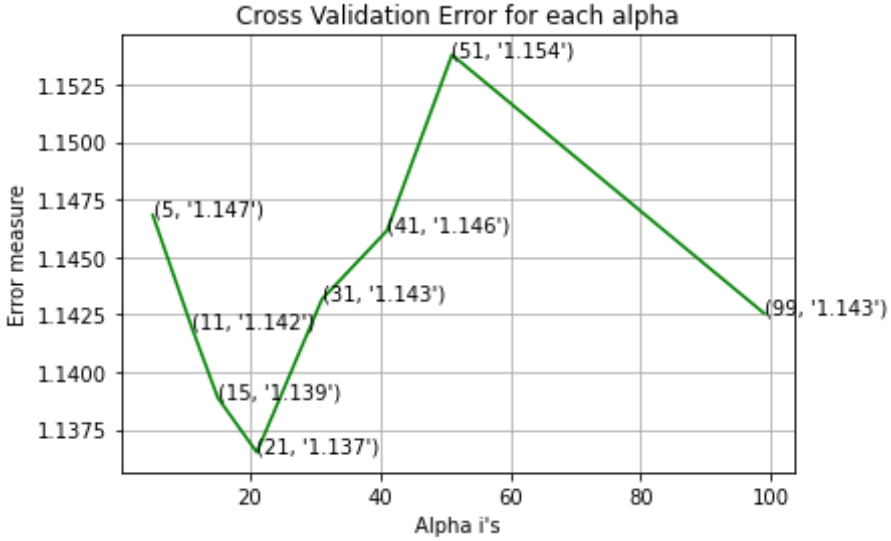
4.2.1. Hyper parameter tuning

In [68]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# 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-examp
le-1/
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
```

```
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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


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

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

```
for alpha = 5
Log Loss : 1.1468297616416518
for alpha = 11
Log Loss : 1.1419361248536133
for alpha = 15
Log Loss : 1.1388822906161522
for alpha = 21
Log Loss : 1.1365119679204274
for alpha = 31
Log Loss : 1.1431607301498505
for alpha = 41
Log Loss : 1.146145642327557
for alpha = 51
Log Loss : 1.1537897828892354
for alpha = 99
Log Loss : 1.1425509925340125
```



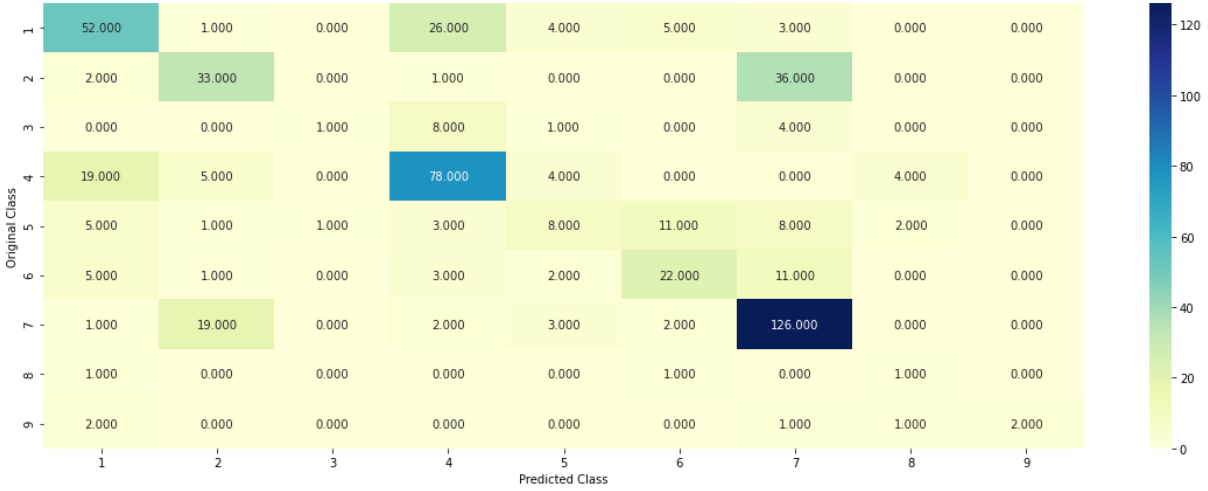

```
For values of best alpha =  21 The train log loss is: 0.732575276658581
9
For values of best alpha =  21 The cross validation log loss is: 1.1365
119679204274
For values of best alpha =  21 The test log loss is: 1.041445507489089
```

4.2.2. Testing the model with best hyper paramters

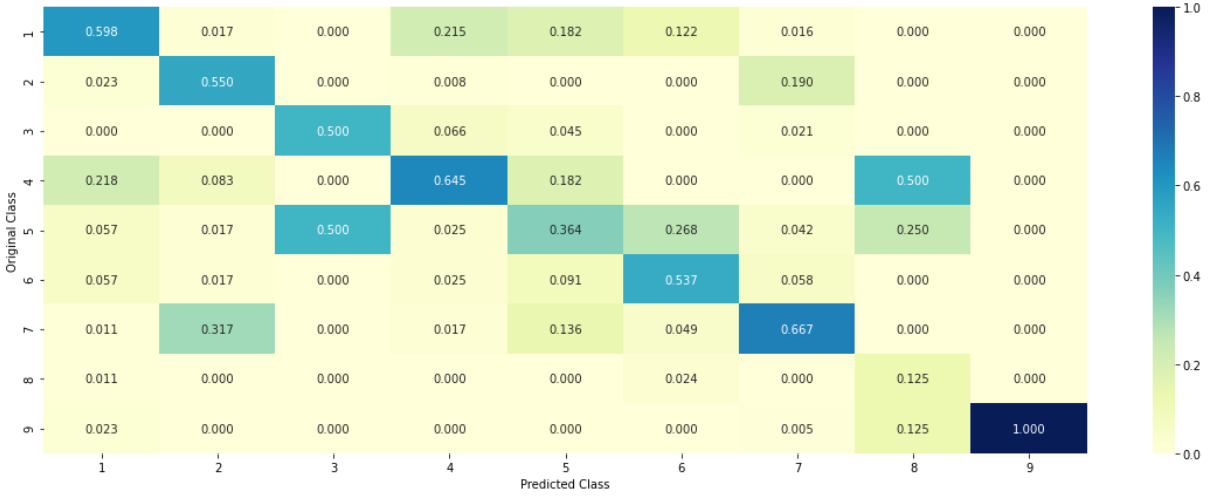
In [69]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
```

```
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# 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-examp
le-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x
_responseCoding, cv_y, clf)
```

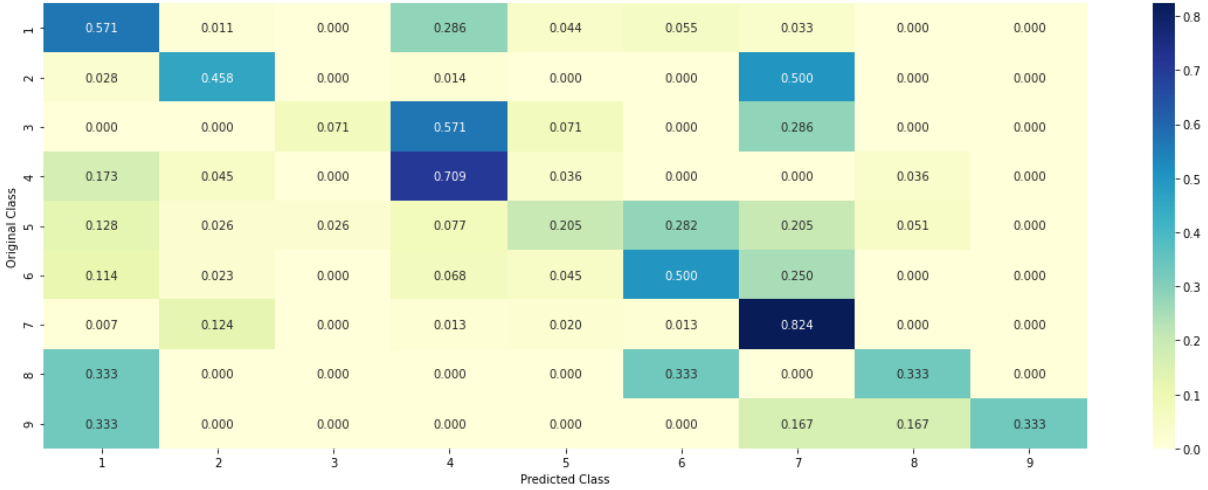
```
Log loss : 1.1365119679204274
Number of mis-classified points : 0.39285714285714285
-------------------- Confusion matrix --------------------
```




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



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



4.2.3.Sample Query point -1

In [70]:
```
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].resh
ape(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  21  nearest neighbours of the test points belongs to classes [7 6
7 2 5 7 5 2 7 7 6 2 7 5 7 7 7 2 2 7 2]
Fequency of nearest points : Counter({7: 10, 2: 6, 5: 3, 6: 2})
```

4.2.4. Sample Query Point-2

In [71]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].resh
ape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
```

```
bours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 4
the k value for knn is 21 and the nearest neighbours of the test points
belongs to classes [1 1 1 1 1 1 1 1 4 1 4 4 1 6 1 1 1 1 1 5 1]
Fequency of nearest points : Counter({1: 16, 4: 3, 6: 1, 5: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [78]:

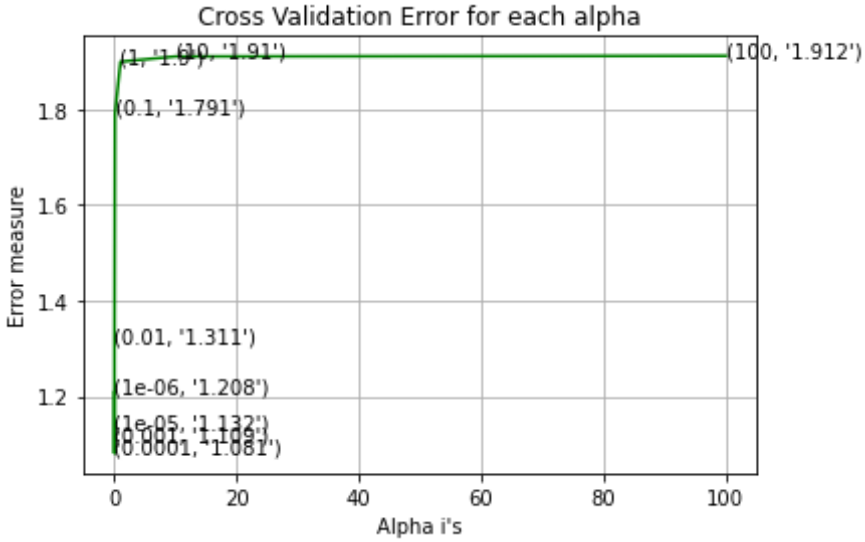
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_TfidfVec, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TfidfVec, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.2083220750442236
for alpha = 1e-05
Log Loss : 1.1322390660765593
for alpha = 0.0001
Log Loss : 1.0809107671761267
for alpha = 0.001
Log Loss : 1.1090395324079734
for alpha = 0.01
Log Loss : 1.3113676460775343
for alpha = 0.1
Log Loss : 1.7910250406429875
for alpha = 1
Log Loss : 1.899518211779723
for alpha = 10
```

```
Log Loss : 1.9104789668812276
for alpha = 100
Log Loss : 1.9117916922980163
```



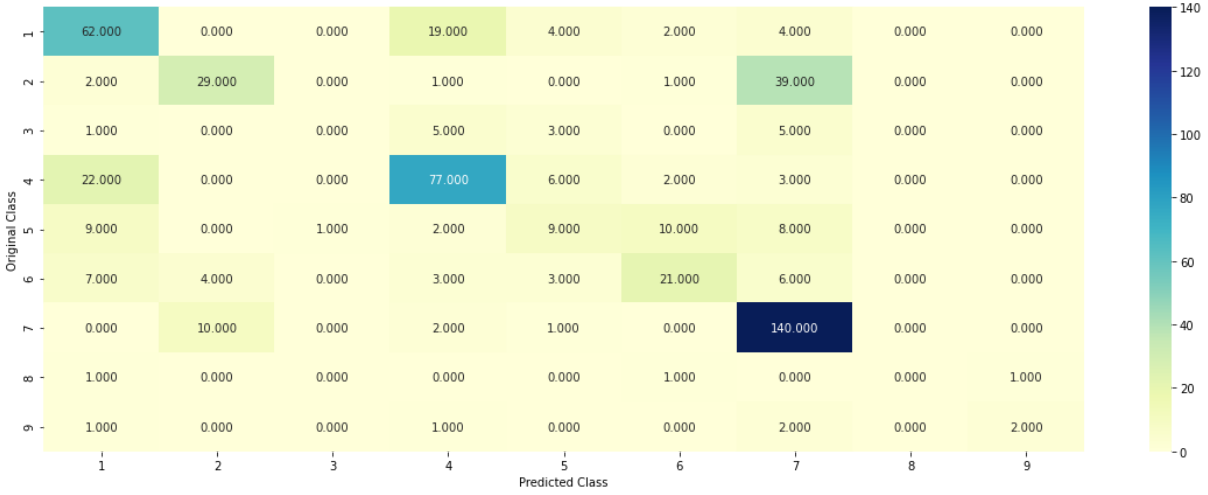

```
For values of best alpha =  0.0001 The train log loss is: 0.39676296596
38532
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0809107671761267
For values of best alpha =  0.0001 The test log loss is: 0.970330806525
5117
```

4.3.1.2. Testing the model with best hyper paramters

In [79]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
```

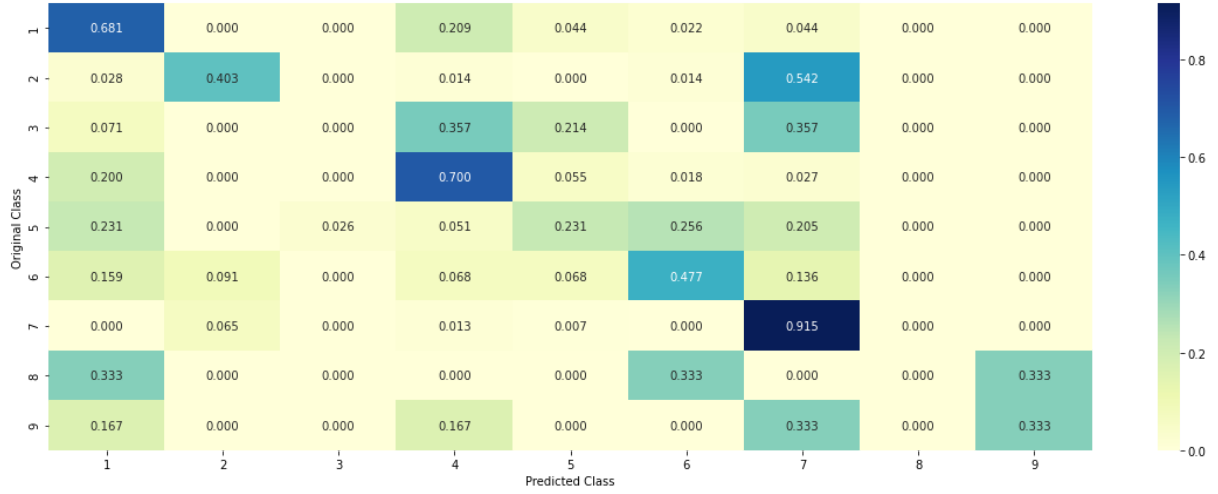
```
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with S
tochastic 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], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_TfidfVec, train_y, cv_x_Tfidf
Vec, cv_y, clf)
```

```
Log loss : 1.0809107671761267
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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

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



4.3.1.3. Feature Importance

In [80]:
```
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_TfidfVec.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
ry point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

4.3.1.3.1. Correctly Classified point

In [81]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[3.500e-03 6.030e-02 4.200e-03 2.250e-0
2 2.310e-02 1.100e-03 8.744e-01
  1.050e-02 3.000e-04]]
Actual Class : 2
--------------------------------------------------
29 Text feature [constitutive] present in test data point [True]
46 Text feature [codon] present in test data point [True]
62 Text feature [activation] present in test data point [True]
65 Text feature [activated] present in test data point [True]
92 Text feature [insertion] present in test data point [True]
114 Text feature [downstream] present in test data point [True]
147 Text feature [missense] present in test data point [True]
214 Text feature [inhibitor] present in test data point [True]
229 Text feature [3b] present in test data point [True]
234 Text feature [oncogenic] present in test data point [True]
240 Text feature [2a] present in test data point [True]
275 Text feature [transforming] present in test data point [True]
277 Text feature [approximately] present in test data point [True]
288 Text feature [substrate] present in test data point [True]
309 Text feature [basal] present in test data point [True]
324 Text feature [signaling] present in test data point [True]
328 Text feature [increased] present in test data point [True]
331 Text feature [genomic] present in test data point [True]
370 Text feature [signalling] present in test data point [True]
```

```
378 Text feature [included] present in test data point [True]

394 Text feature [carcinomas] present in test data point [True]
398 Text feature [activating] present in test data point [True]
414 Text feature [acid] present in test data point [True]
443 Text feature [function] present in test data point [True]
444 Text feature [contrast] present in test data point [True]
445 Text feature [unable] present in test data point [True]
453 Text feature [14] present in test data point [True]
470 Text feature [conserved] present in test data point [True]
476 Text feature [terminal] present in test data point [True]
480 Text feature [affected] present in test data point [True]
489 Text feature [serum] present in test data point [True]
496 Text feature [consequences] present in test data point [True]
499 Text feature [42] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7246 0.0182 0.014  0.111  0.0621 0.0
21  0.0399 0.005  0.0042]]
Actual Class : 4
--------------------------------------------------
106 Text feature [encoding] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2. Without Class balancing

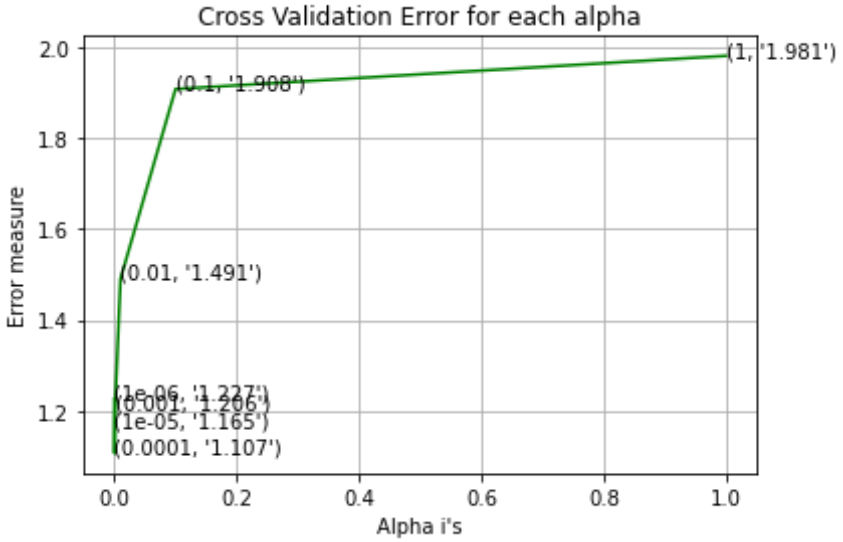
4.3.2.1. Hyper paramter tuning

In [83]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
```

```
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#---------------------------------------
# video link:
#---------------------------------------

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_x_TfidfVec, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_TfidfVec, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_TfidfVec)
    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_TfidfVec, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TfidfVec, train_y)
```

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

```
for alpha = 1e-06
Log Loss : 1.2270185051396545
for alpha = 1e-05
Log Loss : 1.16535611760859
for alpha = 0.0001
Log Loss : 1.106939696667221
for alpha = 0.001
Log Loss : 1.2059858098294955
for alpha = 0.01
Log Loss : 1.4905806512740336
for alpha = 0.1
Log Loss : 1.9076822542849723
for alpha = 1
Log Loss : 1.9806696807110584
```




```
For values of best alpha =  0.0001 The train log loss is: 0.388025232
8954419
For values of best alpha =  0.0001 The cross validation log loss is:
1.106939696667221
For values of best alpha =  0.0001 The test log loss is: 0.9928916311
383522
```

4.3.2.2. Testing model with best hyper parameters

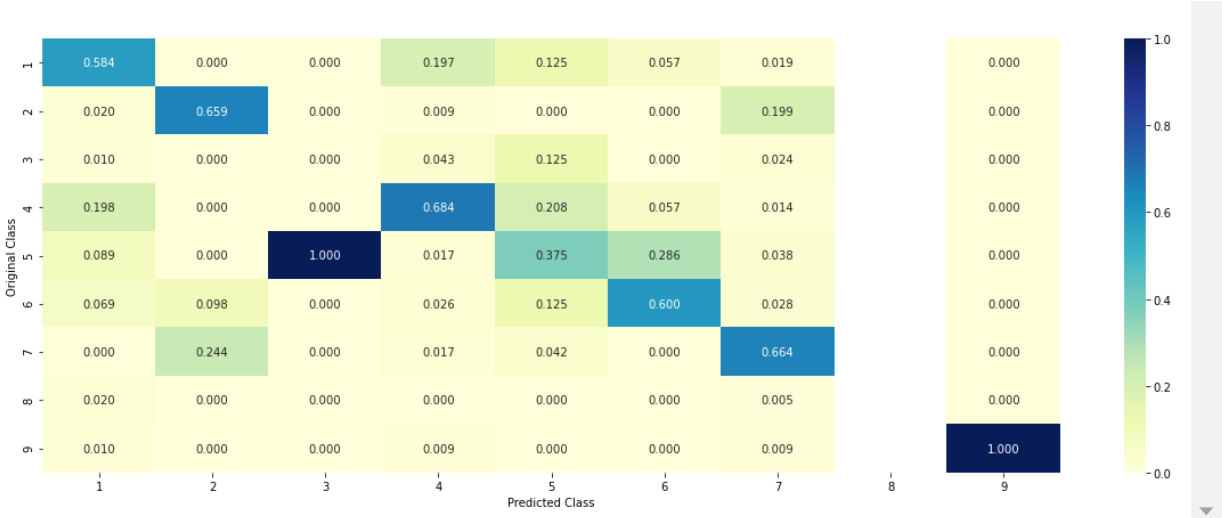
In [84]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
```

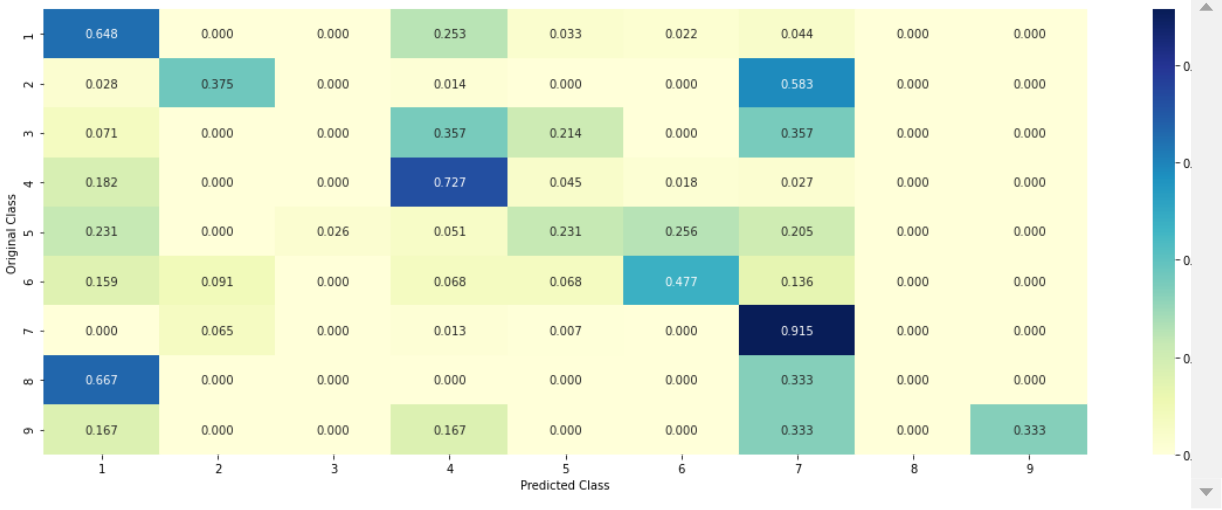
```
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

```
Log loss : 1.106939696667221
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```


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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [85]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_TfidfVec,train_y)
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0037 0.0565 0.0024 0.0304 0.0161 0.0
01  0.8782 0.0118 0.    ]]
Actual Class : 2
--------------------------------------------------
45 Text feature [constitutive] present in test data point [True]
49 Text feature [codon] present in test data point [True]
100 Text feature [activation] present in test data point [True]
113 Text feature [activated] present in test data point [True]
119 Text feature [insertion] present in test data point [True]
153 Text feature [downstream] present in test data point [True]
242 Text feature [missense] present in test data point [True]
244 Text feature [3b] present in test data point [True]
249 Text feature [inhibitor] present in test data point [True]
251 Text feature [2a] present in test data point [True]
286 Text feature [approximately] present in test data point [True]
301 Text feature [increased] present in test data point [True]
304 Text feature [signalling] present in test data point [True]
320 Text feature [basal] present in test data point [True]
334 Text feature [oncogenic] present in test data point [True]
342 Text feature [genomic] present in test data point [True]
352 Text feature [substrate] present in test data point [True]
439 Text feature [transforming] present in test data point [True]
448 Text feature [activating] present in test data point [True]
449 Text feature [contrast] present in test data point [True]
461 Text feature [acid] present in test data point [True]
462 Text feature [carcinomas] present in test data point [True]
```

```
467 Text feature [included] present in test data point [True]
473 Text feature [signaling] present in test data point [True]
488 Text feature [unable] present in test data point [True]
Out of the top  500  features  25 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.742  0.018  0.0123 0.0975 0.0609 0.0
222 0.0373 0.0056 0.0041]]
Actual Class : 4
--------------------------------------------------
104 Text feature [encoding] present in test data point [True]
494 Text feature [sporadic] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

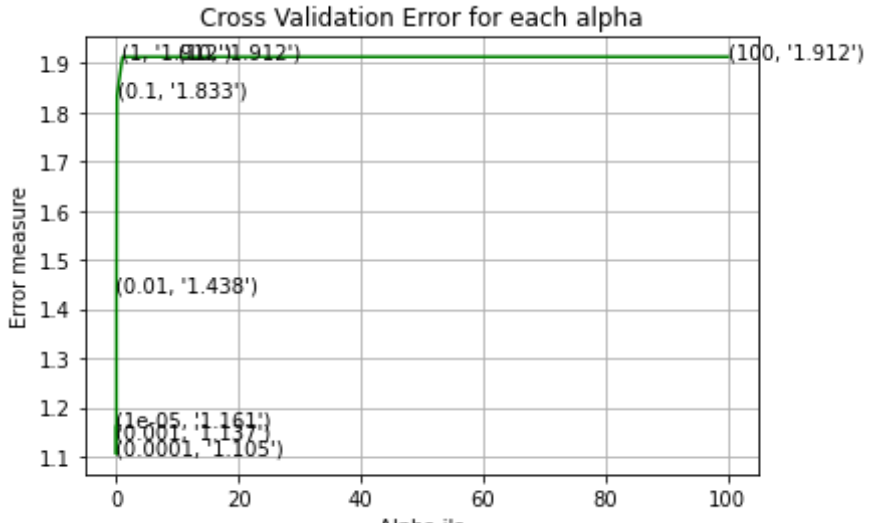
In [87]:
```
# read more about support vector machines with linear kernals here htt
```

```
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
```

```
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='bal
anced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2'
, loss='hinge', random_state=42)
    clf.fit(train_x_TfidfVec, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_TfidfVec, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_TfidfVec)
    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='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_TfidfVec, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TfidfVec, train_y)

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

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

```
for C = 1e-05
Log Loss : 1.1608450558819465
for C = 0.0001
Log Loss : 1.104620151928524
for C = 0.001
Log Loss : 1.1371248167258716
for C = 0.01
Log Loss : 1.4384346994931365
for C = 0.1
Log Loss : 1.833208649736233
for C = 1
Log Loss : 1.9121862788892723
for C = 10
Log Loss : 1.9121922807669507
for C = 100
Log Loss : 1.9121954401834242
```



Alpha i's

```
For values of best alpha =  0.0001 The train log loss is: 0.31233591885
78119
For values of best alpha =  0.0001 The cross validation log loss is: 1.
104620151928524
For values of best alpha =  0.0001 The test log loss is: 0.993347673554
4319
```

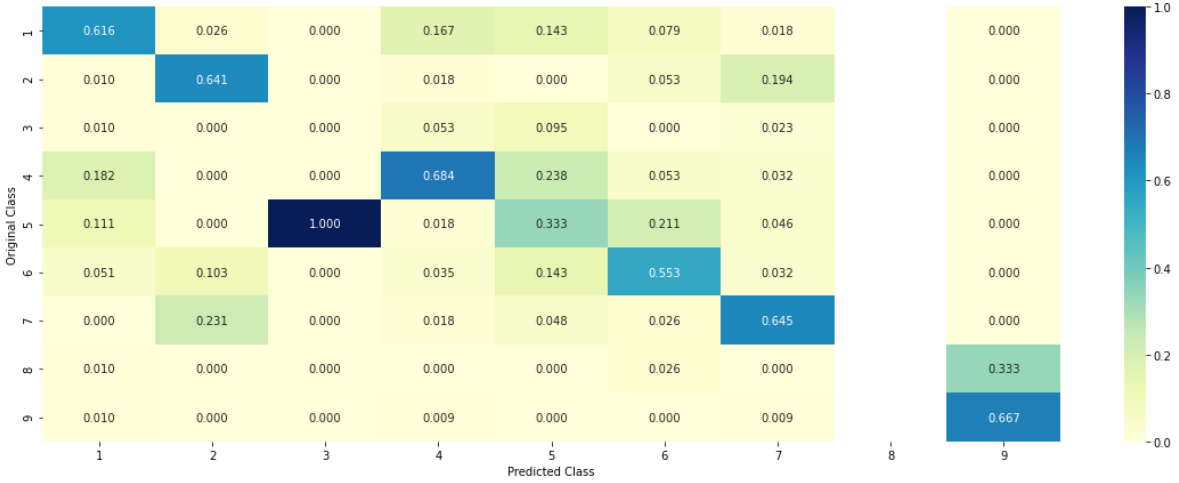
4.4.2. Testing model with best hyper parameters

In [88]:

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

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

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


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

```
Log loss : 1.104620151928524
Number of mis-classified points : 0.37218045112781956
-------------------- 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 [89]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_TfidfVec,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_TfidfVec[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_TfidfVec[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0155 0.0394 0.0173 0.1665 0.0226 0.0
01  0.7308 0.0059 0.001 ]]
Actual Class : 2
--------------------------------------------------
96 Text feature [codon] present in test data point [True]
190 Text feature [3b] present in test data point [True]
192 Text feature [constitutive] present in test data point [True]
203 Text feature [downstream] present in test data point [True]
355 Text feature [insertion] present in test data point [True]
360 Text feature [approximately] present in test data point [True]
367 Text feature [basal] present in test data point [True]
368 Text feature [increased] present in test data point [True]
371 Text feature [signalling] present in test data point [True]
385 Text feature [2a] present in test data point [True]
386 Text feature [note] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [90]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_TfidfVec[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_TfidfVec[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7598 0.0175 0.0226 0.0327 0.1073 0.0
144 0.0368 0.0049 0.0039]]
Actual Class : 4
--------------------------------------------------
```

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

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [91]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
```

```
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
```

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


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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2475929880016963
for n_estimators = 100 and max depth =  10
Log Loss : 1.2566422492241127
for n_estimators = 200 and max depth =  5
Log Loss : 1.2360317289712128
for n_estimators = 200 and max depth =  10
Log Loss : 1.245450400759458
for n_estimators = 500 and max depth =  5
Log Loss : 1.2362661549152487
for n_estimators = 500 and max depth =  10
Log Loss : 1.2437740065655585
for n_estimators = 1000 and max depth =  5
```

```
Log Loss : 1.2357440308847174
for n_estimators = 1000 and max depth =  10
Log Loss : 1.244429918348473
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2331942168015781
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2442204704959519
For values of best estimator =  2000 The train log loss is: 0.828837546
2705457
For values of best estimator =  2000 The cross validation log loss is:
1.2331942168015781
For values of best estimator =  2000 The test log loss is: 1.1350510112
705638
```

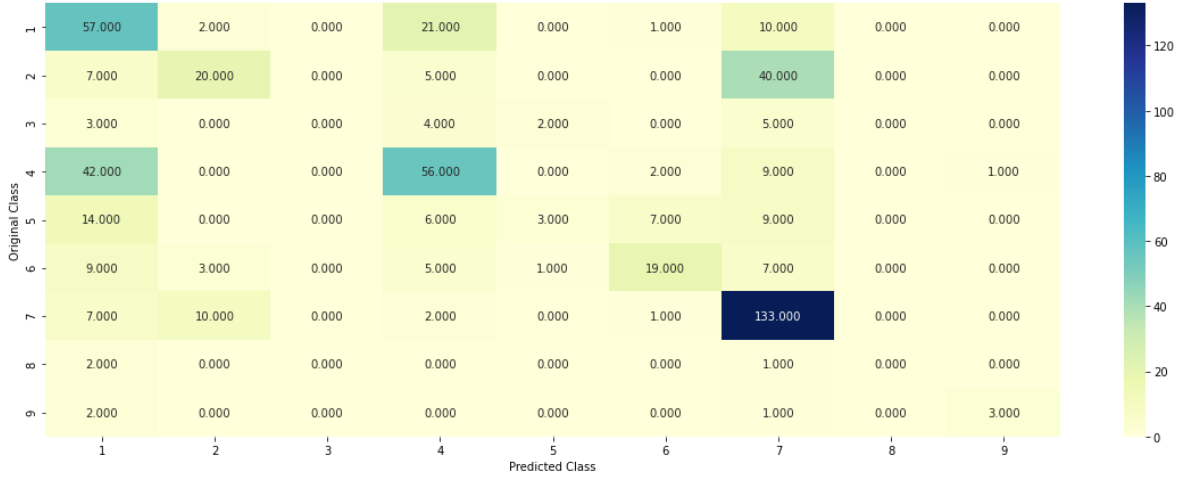
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [92]:

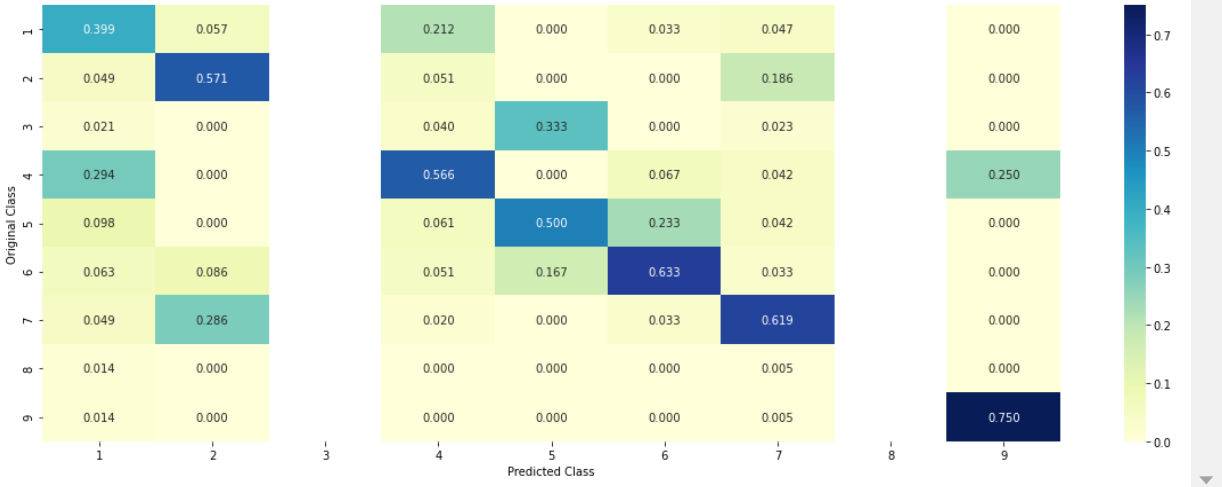
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
```

```
online/lessons/random-forest-and-their-construction-2/
# ------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_TfidfVec, train_y,cv_x_TfidfV
ec,cv_y, clf)
```

```
Log loss : 1.2331942168015781
Number of mis-classified points : 0.45300751879699247
-------------------- 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 [93]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_TfidfVec, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_TfidfVec, train_y)

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.09   0.1593 0.0319 0.1083 0.0751 0.0
642 0.4483 0.0156 0.0075]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [tyrosine] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [treatment] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
9 Text feature [missense] present in test data point [True]
10 Text feature [activated] present in test data point [True]
11 Text feature [phosphorylation] present in test data point [True]
12 Text feature [receptor] present in test data point [True]
14 Text feature [inhibitor] present in test data point [True]
15 Text feature [transforming] present in test data point [True]
18 Text feature [protein] present in test data point [True]
```

```
20 Text feature [functional] present in test data point [True]
21 Text feature [oncogenic] present in test data point [True]

30 Text feature [signaling] present in test data point [True]
33 Text feature [cells] present in test data point [True]
36 Text feature [erk] present in test data point [True]
47 Text feature [proteins] present in test data point [True]
53 Text feature [3t3] present in test data point [True]
56 Text feature [cell] present in test data point [True]
59 Text feature [kinases] present in test data point [True]
63 Text feature [downstream] present in test data point [True]
66 Text feature [mek] present in test data point [True]
72 Text feature [ovarian] present in test data point [True]
73 Text feature [activity] present in test data point [True]
74 Text feature [sequence] present in test data point [True]
79 Text feature [assays] present in test data point [True]
84 Text feature [variant] present in test data point [True]
85 Text feature [expressing] present in test data point [True]
87 Text feature [serum] present in test data point [True]
88 Text feature [dna] present in test data point [True]
94 Text feature [tagged] present in test data point [True]
98 Text feature [expected] present in test data point [True]
Out of the top  100  features  35 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [94]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_TfidfVec[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_TfidfVec[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_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2525 0.0283 0.0213 0.4436 0.0775 0.1
307 0.0364 0.0042 0.0057]]
Actuall Class : 4
--------------------------------------------------
9 Text feature [missense] present in test data point [True]
18 Text feature [protein] present in test data point [True]
20 Text feature [functional] present in test data point [True]
26 Text feature [pathogenic] present in test data point [True]
33 Text feature [cells] present in test data point [True]
38 Text feature [variants] present in test data point [True]
56 Text feature [cell] present in test data point [True]
84 Text feature [variant] present in test data point [True]
Out of the top  100  features  8 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [95]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# -------------------------------
```

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
```

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

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.194761832835216
for n_estimators = 10 and max depth =  3
Log Loss : 1.728974834124511
for n_estimators = 10 and max depth =  5
Log Loss : 1.508462494983981
```

```
for n_estimators = 10 and max depth =  10
Log Loss : 2.027664208135141
for n_estimators = 50 and max depth =  2
Log Loss : 1.72092954113363
for n_estimators = 50 and max depth =  3
Log Loss : 1.4271693210034293
for n_estimators = 50 and max depth =  5
Log Loss : 1.4599309853756604
for n_estimators = 50 and max depth =  10
Log Loss : 1.7803072533346604
for n_estimators = 100 and max depth =  2
Log Loss : 1.630068188028538
for n_estimators = 100 and max depth =  3
Log Loss : 1.4631750980740965
for n_estimators = 100 and max depth =  5
Log Loss : 1.4271308737428536
for n_estimators = 100 and max depth =  10
Log Loss : 1.7808101338398858
for n_estimators = 200 and max depth =  2
Log Loss : 1.59715386293404
for n_estimators = 200 and max depth =  3
Log Loss : 1.4690348309053258
for n_estimators = 200 and max depth =  5
Log Loss : 1.4812131185051027
for n_estimators = 200 and max depth =  10
Log Loss : 1.7980295324031823
for n_estimators = 500 and max depth =  2
Log Loss : 1.6531798198245558
for n_estimators = 500 and max depth =  3
Log Loss : 1.5058795453379643
for n_estimators = 500 and max depth =  5
Log Loss : 1.4530573793952253
for n_estimators = 500 and max depth =  10
Log Loss : 1.814972011916507
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6159450032120914
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5039717293100554
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4388986134567927
```

```
Log Loss : 1.4388986134567927
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8159147495830523
For values of best alpha =  100 The train log loss is: 0.06956851429310
7
For values of best alpha =  100 The cross validation log loss is: 1.427
1308737428534

For values of best alpha =  100 The test log loss is: 1.334765330938606
5
```

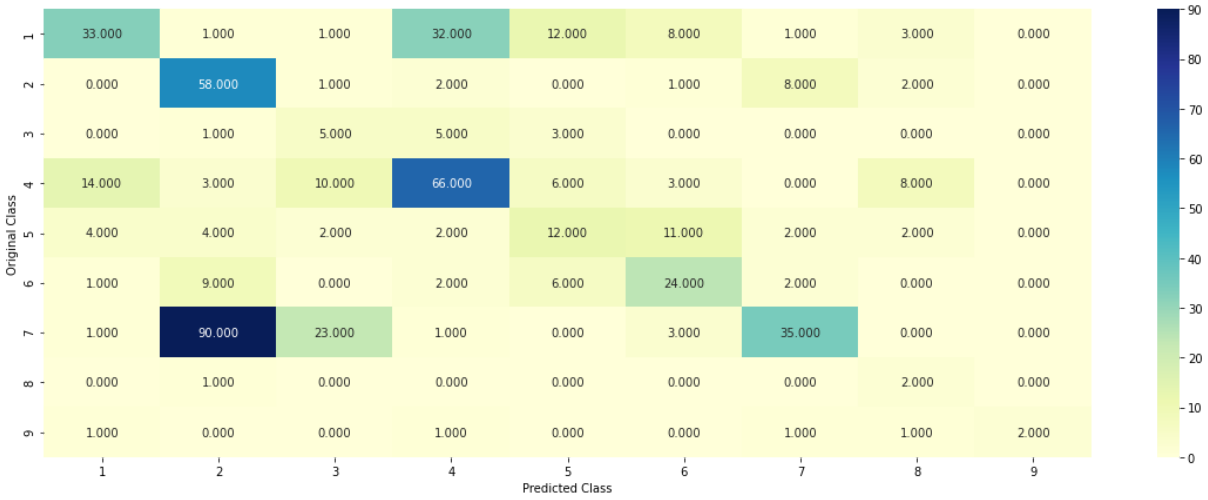
4.5.4. Testing model with best hyper parameters (Response Coding)

In [96]:

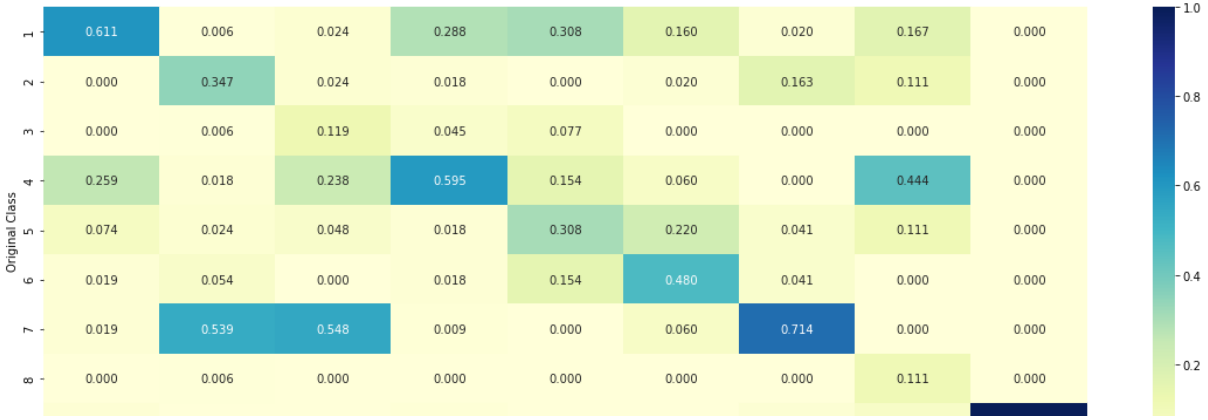
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

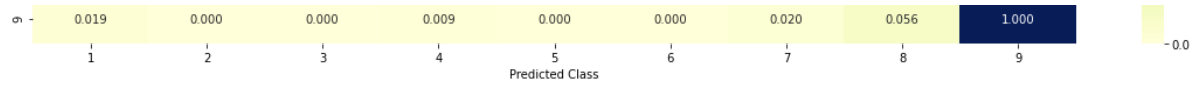
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='au
to',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_
responseCoding,cv_y, clf)
```

```
Log loss : 1.4271308737428536
Number of mis-classified points : 0.5545112781954887
-------------------- Confusion matrix --------------------
```

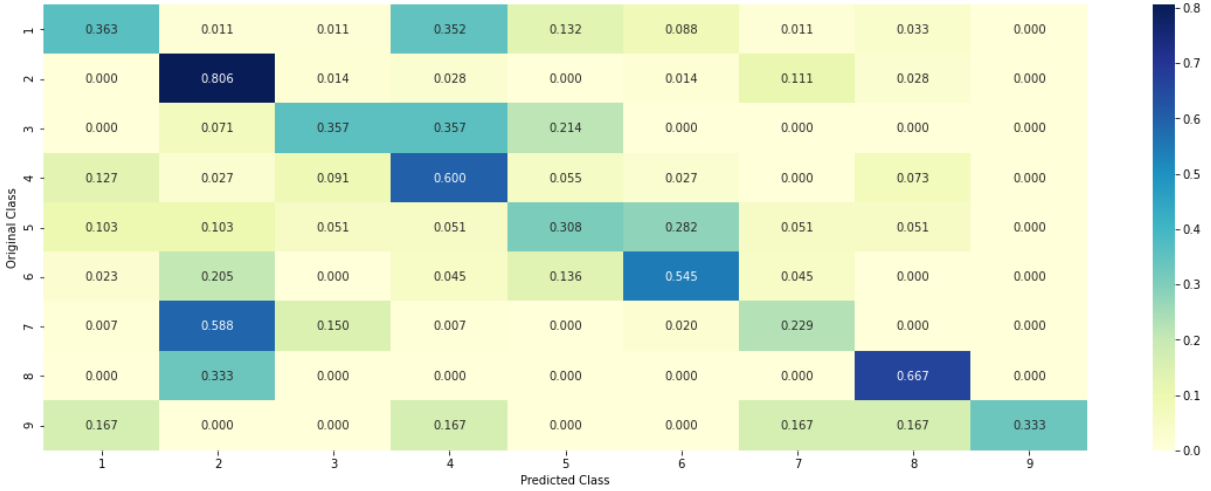



```
-------------------- Precision matrix (Columm Sum=1) ----------------
--
```





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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [97]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='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]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0296 0.3495 0.1858 0.0341 0.081  0.0
605 0.2122 0.0309 0.0163]]
```

```
Actual Class : 2

--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [98]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
```

```
test_x_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: [[0.3444 0.0544 0.0997 0.1863 0.055  0.0
948 0.0229 0.1045 0.0382]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
```

```
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [99]:

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

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
```

```
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

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


# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFo
restClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

```
# --------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(train_x_TfidfVec, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

sig_clf1.fit(train_x_TfidfVec, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_cl
f1.predict_proba(cv_x_TfidfVec))))
sig_clf2.fit(train_x_TfidfVec, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig
_clf2.predict_proba(cv_x_TfidfVec))))
sig_clf3.fit(train_x_TfidfVec, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predic
t_proba(cv_x_TfidfVec))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3
], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_TfidfVec, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_TfidfVec))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_TfidfVec))
```

```
    if best_alpha > log_error:
        best_alpha = log_error
```

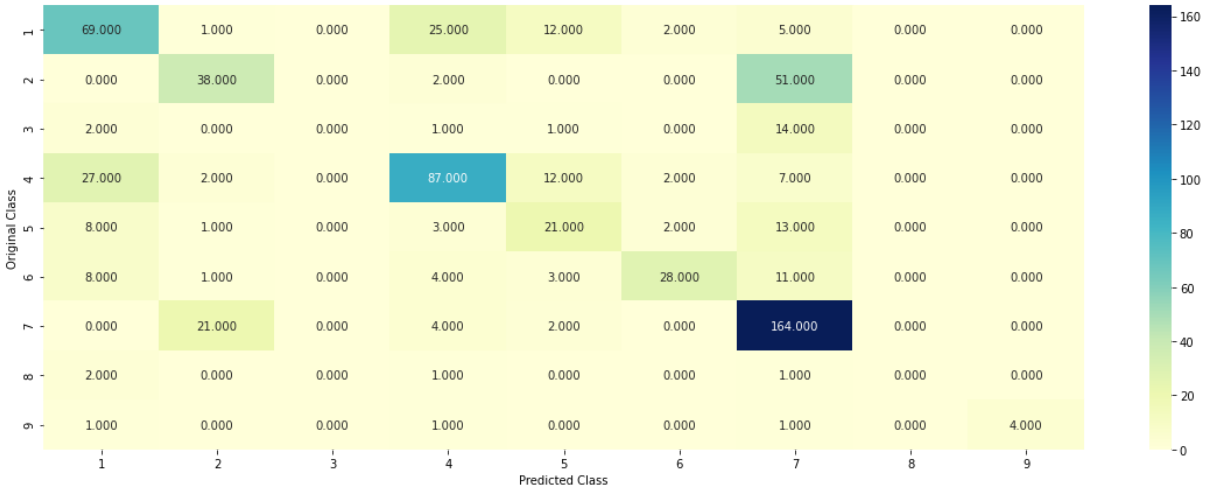
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.91
Naive Bayes : Log Loss: 1.30
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.714
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.334
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.308
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.723
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 2.215
```

4.7.2 testing the model with the best hyper parameters

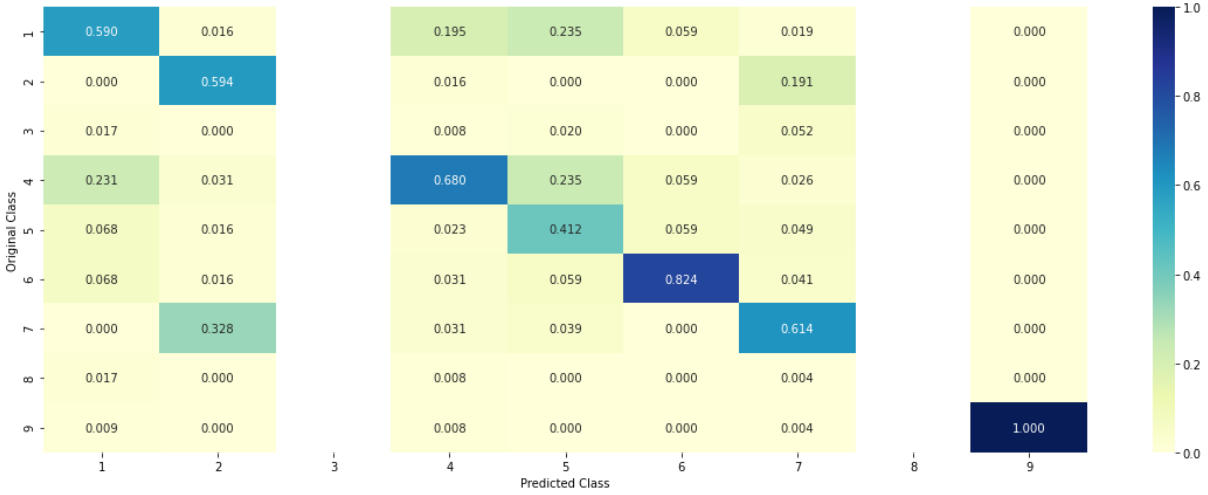
In [100]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_TfidfVec, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.32982476127315724
Log loss (CV) on the stacking classifier : 1.307743155658356
Log loss (test) on the stacking classifier : 1.2329048400788347
```

```
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```




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



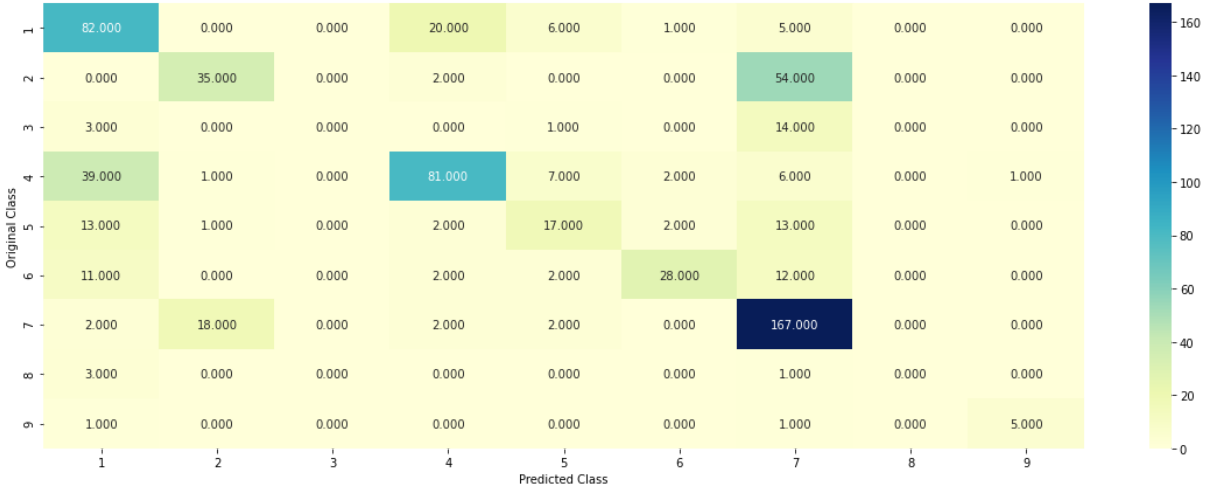

```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.7.3 Maximum Voting classifier

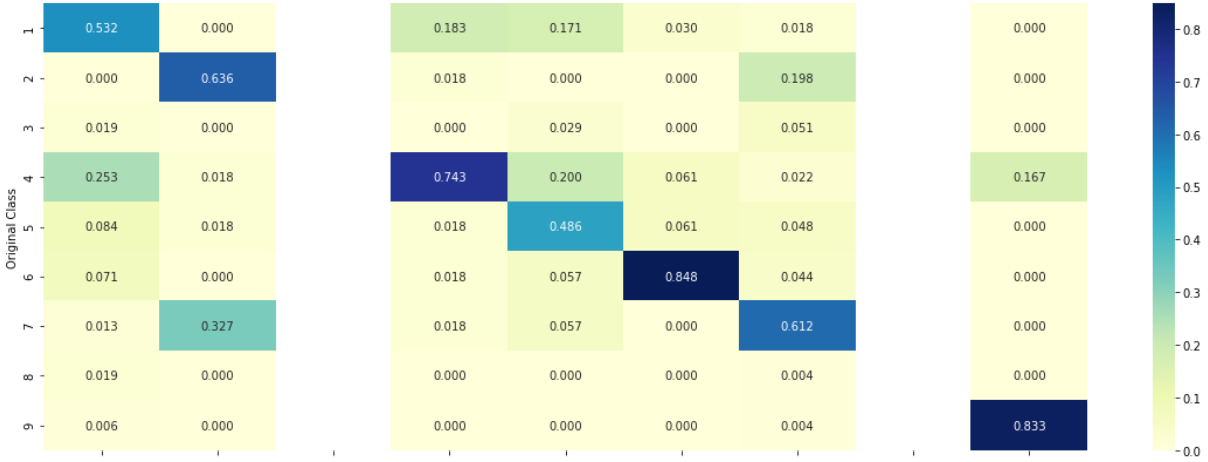
In [101]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_TfidfVec, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_TfidfVec)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_TfidfVec)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_TfidfVec)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_TfidfVec)- test_y))/test_y.shape[0])
```

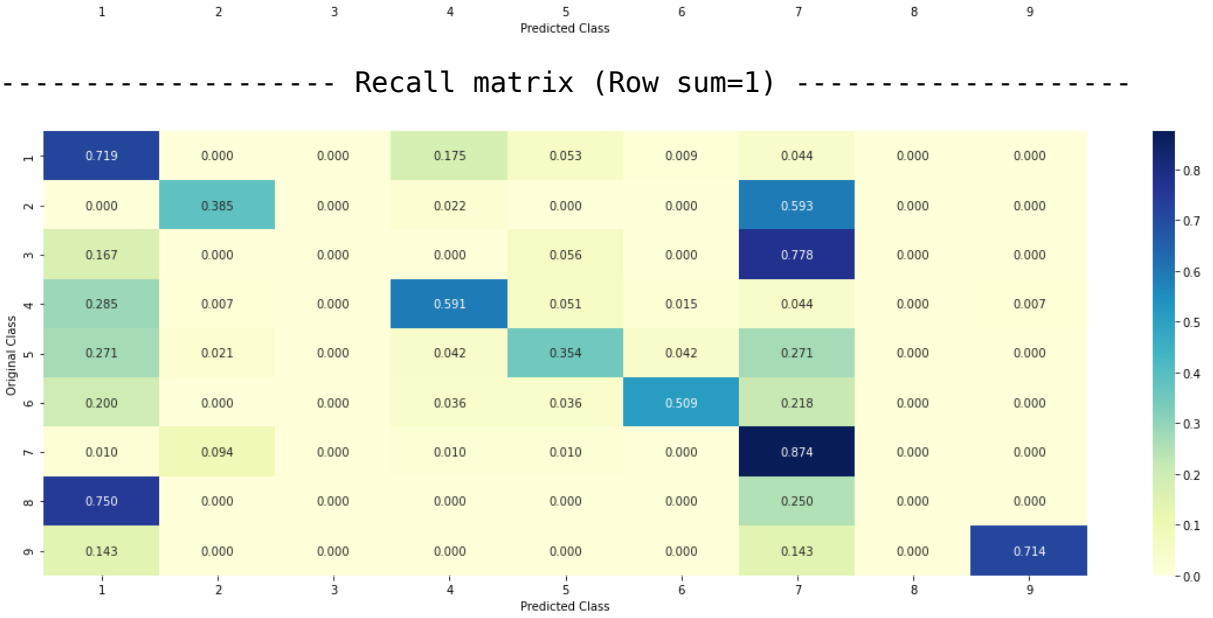
```
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_TfidfVec))
```

```
Log loss (train) on the VotingClassifier : 0.7836559936350218
Log loss (CV) on the VotingClassifier : 1.2735075290687627
Log loss (test) on the VotingClassifier : 1.1833642522282948
Number of missclassified point : 0.37593984962406013
--------------------- Confusion matrix --------------------
```




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





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

Summary of all the models with TFIDF vectorization

In [39]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names=["Model Name","Train","CV","Test","% Misclassified Point
s"]
```

```
x.add_row(["Naive Bayes","0.536","1.266","1.211","40"])
x.add_row(["KNN","0.732","1.136","1.041","39"])
x.add_row(["Logistic Regression W CB","0.388","1.015","1.023","36"])
x.add_row(["Logistic Regression W/O CB","0.396","1.08","0.978","36"])
x.add_row(["Linear SVM","0.312","1.104","0.993","37"])
x.add_row(["Random Forest Classifier  OHE","0.828","1.233","1.35","45"
])
x.add_row(["Random Forest Classifier RC","0.06","1.427","1.334","55"])
x.add_row(["Stack Models:LR+NB+SVM","0.329","1.307","1.232","38"])
x.add_row(["Maximum Voting classifier","0.738","1.273","1.183","37"])
print(x)
```

```
+-------------------------------+-------+-------+-------+-------------
----------+
|           Model Name          | Train |   CV  |  Test | % Misclassifi
ed Points |
+-------------------------------+-------+-------+-------+-------------
----------+
|          Naive Bayes          | 0.536 | 1.266 | 1.211 |           40
          |
|              KNN              | 0.732 | 1.136 | 1.041 |           39
          |
|    Logistic Regression W CB   | 0.388 | 1.015 | 1.023 |           36
          |
|   Logistic Regression W/O CB  | 0.396 |  1.08 | 0.978 |           36
          |
|           Linear SVM          | 0.312 | 1.104 | 0.993 |           37
          |
| Random Forest Classifier  OHE | 0.828 | 1.233 |  1.35 |           45
          |
|  Random Forest Classifier RC  |  0.06 | 1.427 | 1.334 |           55
          |
|     Stack Models:LR+NB+SVM    | 0.329 | 1.307 | 1.232 |           38
          |
|   Maximum Voting classifier   | 0.738 | 1.273 | 1.183 |           37
          |
+-------------------------------+-------+-------+-------+-------------
----------+
```

WithTFIDF vectorizer it is observed Logistic regression model is doing a good job

Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams

Gene

In [103]:
```
#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, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [104]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

#Normalise features
train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCo
ding, axis=0)
test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCodi
ng, axis=0)
cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding,
axis=0)
```

Variation

In [105]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [106]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])

train_variation_feature_onehotCoding = normalize(train_variation_featur
e_onehotCoding, axis=0)
test_variation_feature_onehotCoding = normalize(test_variation_feature_
onehotCoding, axis=0)
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_oneh
otCoding, axis=0)
```

Text Feature

In [107]:
```
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1, 2))
```

```
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])

# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of 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 : 770963
```

In [108]:
```
#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)

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

In [109]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
```

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

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

Stack above three features

In [110]:

```
# merging all there features gene, variance and text features

# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]
# building train, test and cross validation data sets
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_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_fea
ture_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))
```

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, trai
n_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_t
ext_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [111]:
```
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 = ", t
est_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 77
3255)
(number of data points * number of features) in test data =  (665, 7732
55)
(number of data points * number of features) in cross validation data =
(532, 773255)
```

In [112]:
```
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 = ", t
est_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(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)
```

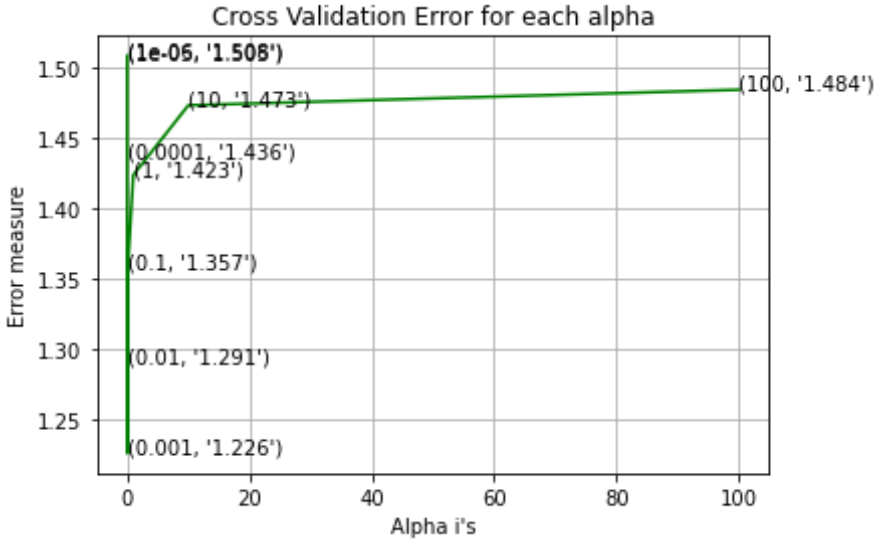
Apply Logistic Regression with CB

In [113]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
plt.show()


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.5077506215342098
for alpha = 1e-05
Log Loss : 1.5050994771257833
for alpha = 0.0001
Log Loss : 1.4358033083146124
for alpha = 0.001
Log Loss : 1.2262074796014903
for alpha = 0.01
Log Loss : 1.2906654323726066
for alpha = 0.1
Log Loss : 1.3572896451021357
for alpha = 1
```

```
Log Loss : 1.4232693293498189
for alpha = 10
Log Loss : 1.4727096482557385
for alpha = 100
Log Loss : 1.4837306069814684
```

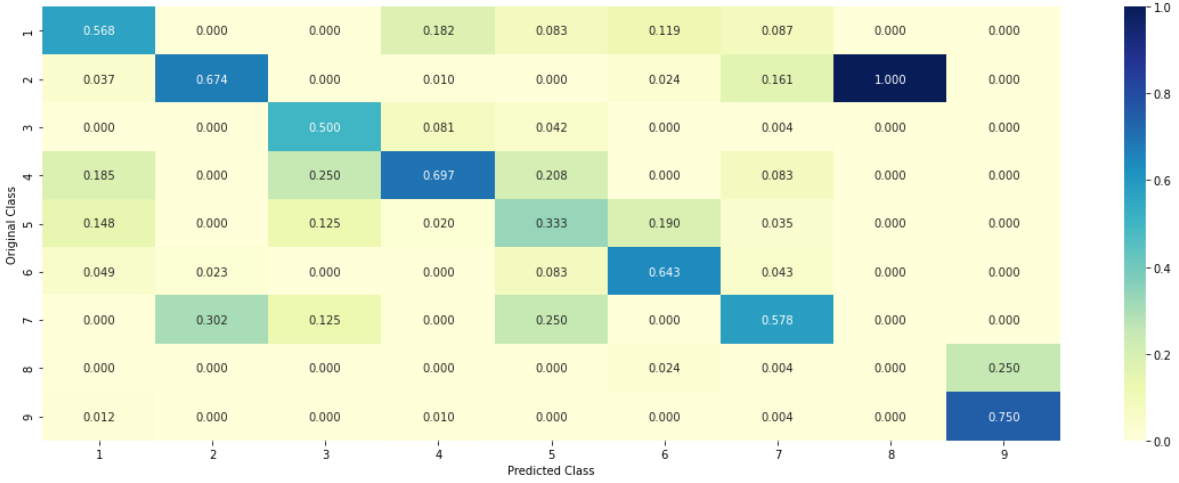



```
For values of best alpha =  0.001 The train log loss is: 0.698049916364
3049
For values of best alpha =  0.001 The cross validation log loss is: 1.2
262074796014903
For values of best alpha =  0.001 The test log loss is: 1.1648399845990
682
```

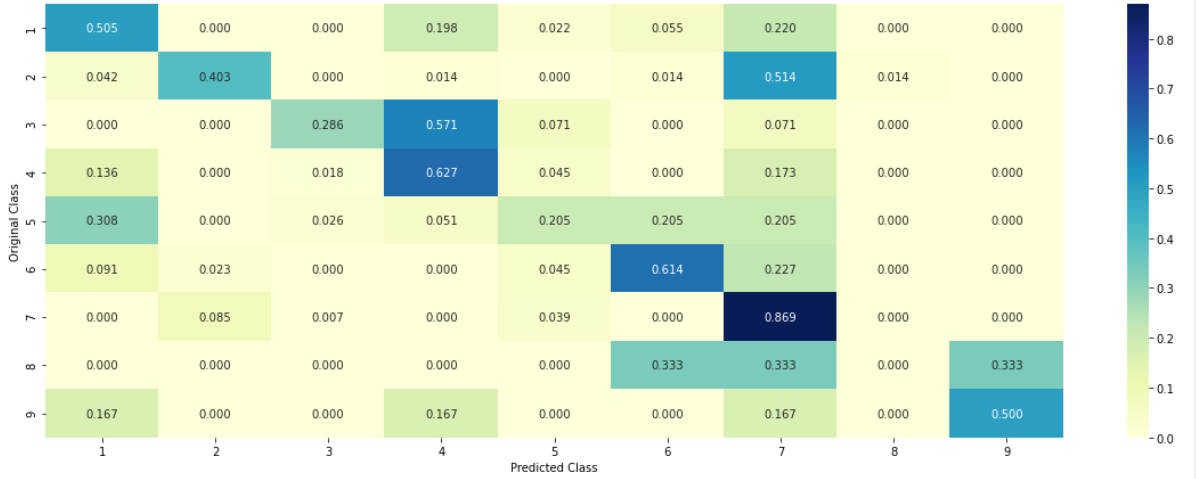
In [114]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

```
Log loss : 1.2262074796014903
Number of mis-classified points : 0.40037593984962405
-------------------- Confusion matrix --------------------
```

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



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



In [ ]:
```
x.field_names=["Model Name","Train","CV","Test","% Misclassified Points"]
x.add_row(["Naive Bayes","0.444","1.188","1.192","39"])
```

Model Name	Train	CV	Test	% Misclassified Points
Logistic Regression	0.698	1.226	1.164	40

Using Count vectorizer with unigram and bigram it is observed that log loss and misclassification errors is not decreased greatly so will move on to feature engineering

Feature Engineering

Gene Feature

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, "Gen
e", train_df))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))

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

In [21]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

Variation

In [22]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [23]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])

train_variation_feature_onehotCoding = normalize(train_variation_featur
e_onehotCoding, axis=0)
test_variation_feature_onehotCoding = normalize(test_variation_feature_
onehotCoding, axis=0)
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_oneh
otCoding, axis=0)
```

Text Feature

In [24]:
```
def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary


import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(t
otal_dict.get(word,0)+90)))
```

```
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [25]:
```
# building a TfidfVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

```
Total number of unique words in train data : 126949
```

In [26]:
```
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 [28]:
```
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
```

Final Features after feature engineering

In [29]:
```
# Putting genes and variations data into a single list
gene_vari = []

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

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

In [30]:
```
tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_vari)
gene_vari_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(train_df['TEXT'])
test_text = tfidfVectorizer.transform(test_df['TEXT'])
cv_text = tfidfVectorizer.transform(cv_df['TEXT'])
```

Stack above three features

In [31]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))

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

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

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCo
ding)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [32]:
```
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 = ", t
est_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 13
0242)
(number of data points * number of features) in test data =  (665, 1302
```

```
42)
(number of data points * number of features) in cross validation data =
(532, 130242)
```

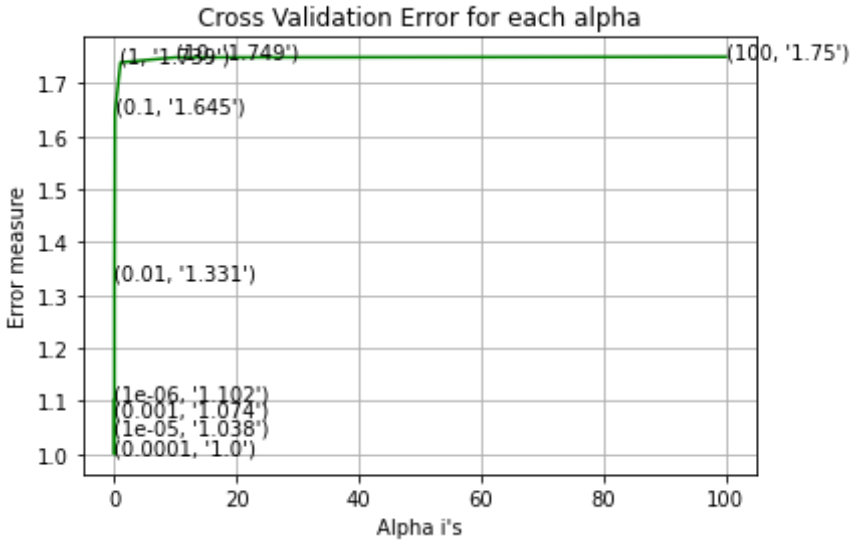
Apply Logistic Regression

In [33]:

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

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


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

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

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

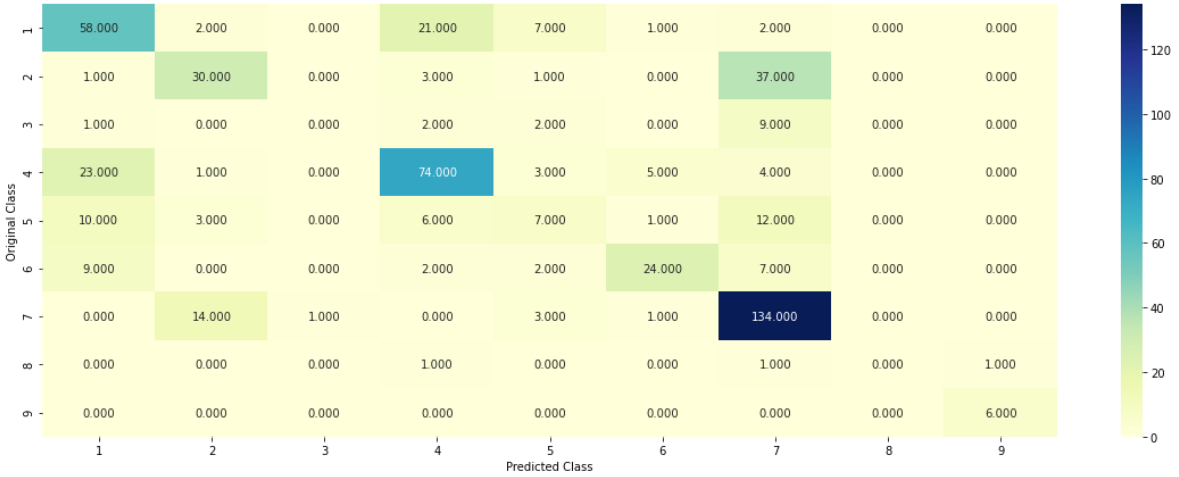
```
for alpha = 1e-06
Log Loss : 1.1021228653459532
for alpha = 1e-05
Log Loss : 1.038210229480967
for alpha = 0.0001
Log Loss : 0.9996844663163629
for alpha = 0.001
Log Loss : 1.0741123090891918
for alpha = 0.01
Log Loss : 1.3307229669332767
for alpha = 0.1
Log Loss : 1.6451081119816384
for alpha = 1
Log Loss : 1.7392609937110843
for alpha = 10
Log Loss : 1.7488691582134304
for alpha = 100
Log Loss : 1.7498749608307396
```




```
For values of best alpha =  0.0001 The train log loss is: 0.42152265742
382694
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9996844663163629
For values of best alpha =  0.0001 The test log loss is: 0.958989436781
9195
```

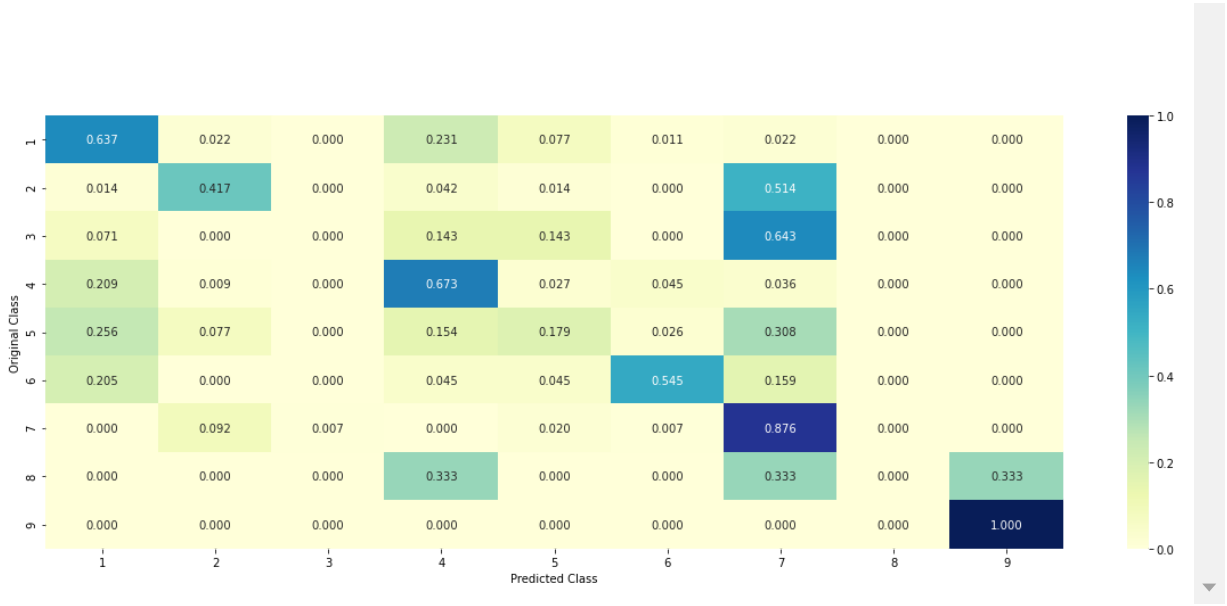
In [37]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

```
Log loss : 0.9996844663163629
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```



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

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



Model Name	Train	CV	Test	% Misclassified Points
Logistic Regression	0.421	0.999	0.958	37

After feature engineering it is observed that log loss and misclassified points has decreased a little and doing reasonabally good job