Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import sys
sys.path.insert(0,'C:\\Users\\nrtsa\\AppData\\Local\\Programs\\Python\\Python37\\Lib\\site-packages')
```

In [240]:
```
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 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 [241]:
```
data = pd.read_csv('training_variants/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[241]:

	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 [242]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text/training_text",sep="\|\|",engine="pytho
n",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[242]:

	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 [243]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [244]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 1375.827107000001 seconds
```

In [245]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[245]:

	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 [246]:
```
result[result.isnull().any(axis=1)]
```

Out[246]:

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

In [247]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [248]:
```
result[result['ID']==1109]
```

Out[248]:

	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 [249]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

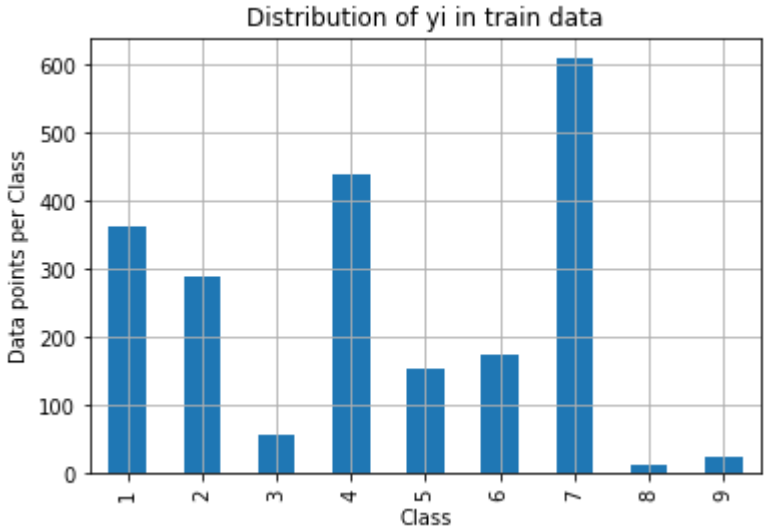
In [250]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

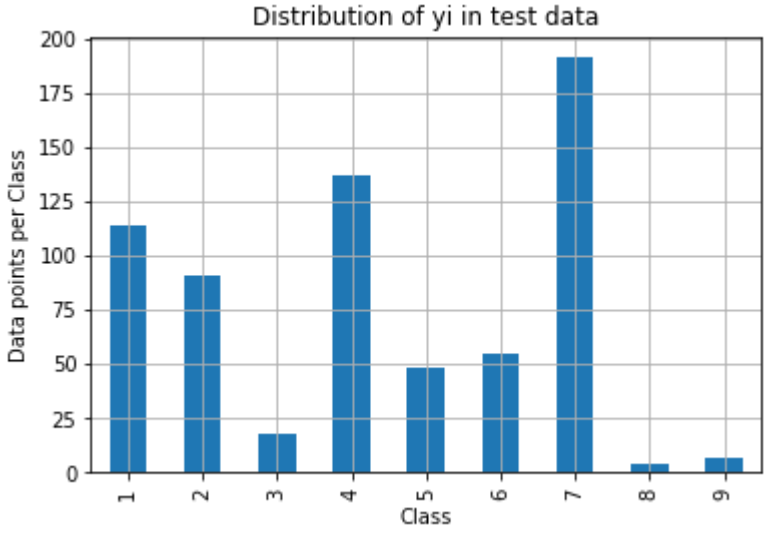
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [251]:
```
# it returns a dict, keys as class labels and values as the number of data poi
nts in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.v
alues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*10
0), 3), '%)')

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

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

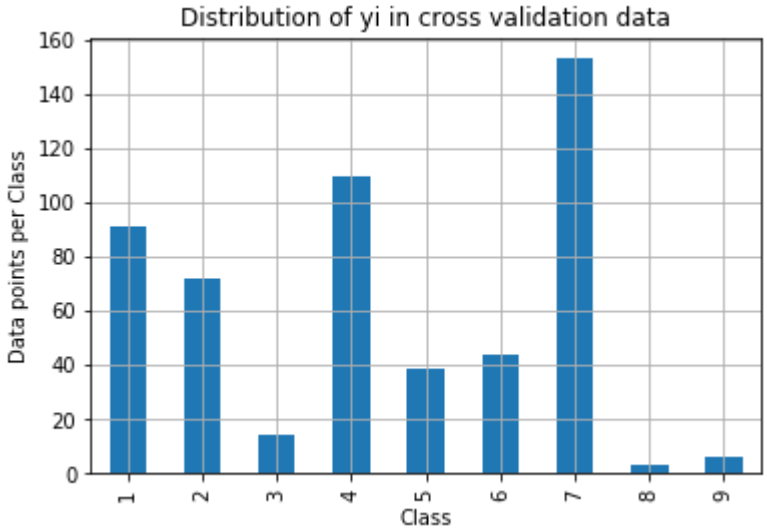
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```




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




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




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

3.2 Prediction using a 'Random' Model

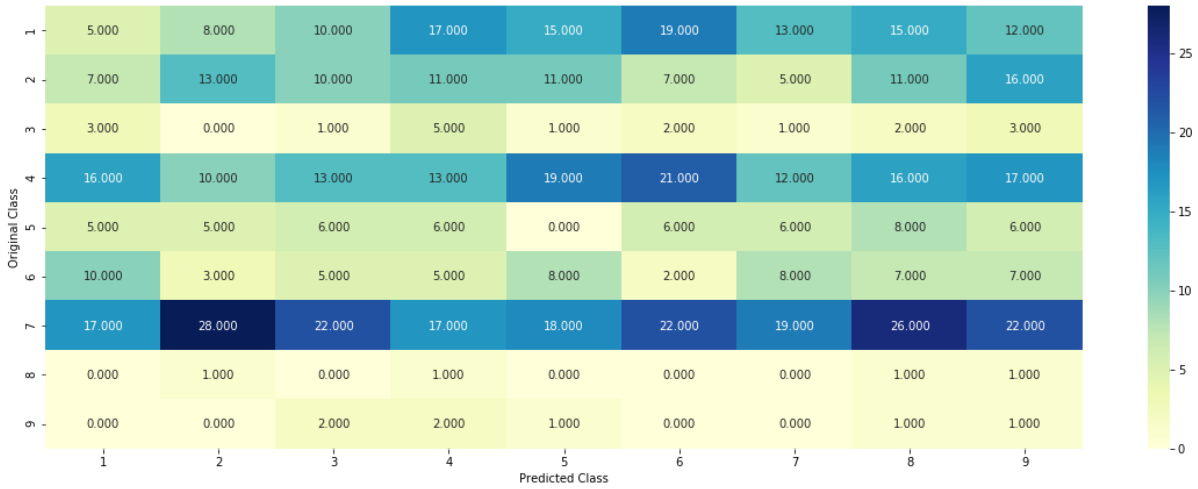
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [252]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to
 rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, y
ticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
```

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

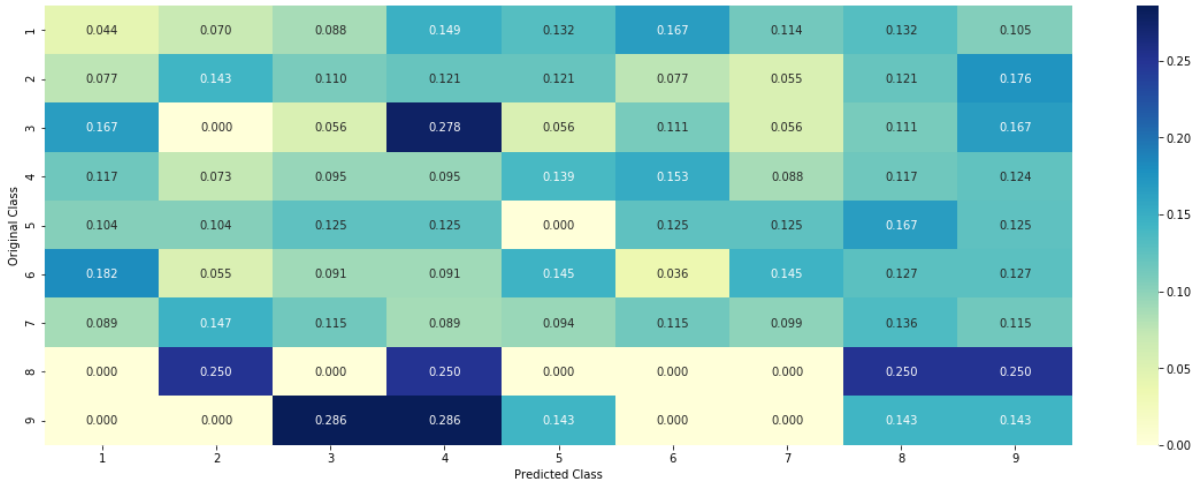
In [253]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
r sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.4825795187873934
Log loss on Test Data using Random Model 2.485474735099201
-------------------- Confusion matrix --------------------
```




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


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



3.3 Univariate Analysis

In [254]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in
 train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in cl
ass1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representat
ion of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

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

```
red in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs
 to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=
='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]
==i)]

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818
18177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.
03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897959
18366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.0510
20408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818
1818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.05
6818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060
6060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069
182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920
5295, 0.07284768211920 5295, 0.066225165562913912, 0.066225165562913912, 0.2715
2317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333
33334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.299
99999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
```

```
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each fea
ture value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is
 there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

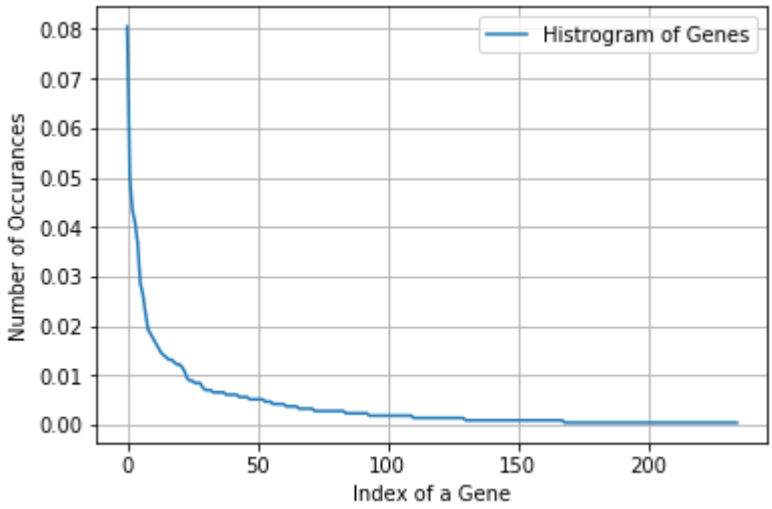
In [255]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 235
BRCA1     171
TP53      106
EGFR       92
BRCA2      87
PTEN       78
KIT        61
BRAF       56
ALK        48
ERBB2      41
PDGFRA     39
Name: Gene, dtype: int64
```

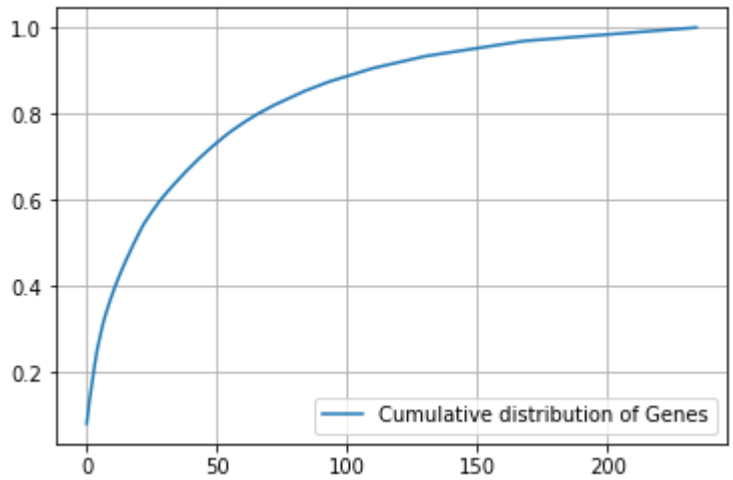
In [256]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

Ans: There are 235 different categories of genes in the train data, and they are distibuted as follows

In [257]:
```
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 [258]:
```
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 [259]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", tra
in_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test
_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df
))
```

In [260]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
```

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

In [261]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(max_features = 1000)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [262]:
```
train_df['Gene'].head()
```

Out[262]:
```
1076     FOXA1
283     NKX2-1
2125     CCND1
962       ESR1
1278      HRAS
Name: Gene, dtype: object
```

In [263]:
```
gene_vectorizer.get_feature_names()
```

Out[263]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid2',
 'arid5b',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
```

```
                'dicer1',
                'dnmt3b',
                'egfr',
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fam58a',
                'fanca',
                'fancc',
                'fat1',
                'fbxw7',
                'fgf19',
                'fgf3',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxo1',
                'foxp1',
                'fubp1',
                'gata3',
                'gna11',
                'gnaq',
                'gnas',
                'h3f3a',
                'hist1h1c',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikzf1',
                'il7r',
                'inpp4b',
                'jak1',
                'jak2',
                'kdm5c',
                'kdm6a',
                'kdr',
```

```
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
```

```
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smo',
'sos1',
'sox9',
'spop',
'src',
'stag2',
'stat3',
'stk11',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
```

```
'vegfa',
'vhl',
'whsc1',
'xpo1',
'xrcc2',
'yap1']
```

In [264]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature:", train_gene_feature_onehotCoding.shap
e)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding m
ethod. The shape of gene feature: (2124, 234)
```

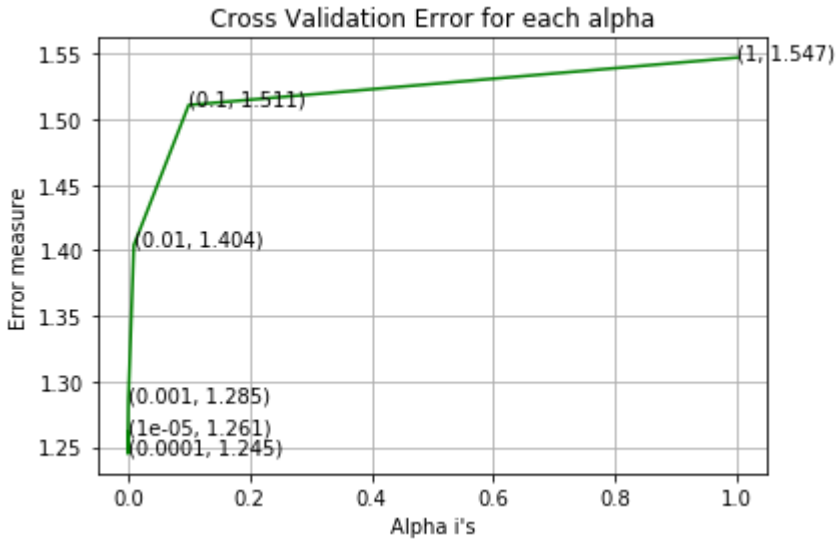
Q4. How good is this gene feature in predicting y_i?

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 [265]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, 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_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.2607914765571266
For values of alpha =  0.0001 The log loss is: 1.2447750915461886
For values of alpha =  0.001 The log loss is: 1.2851870200100242
For values of alpha =  0.01 The log loss is: 1.4038928491406362
For values of alpha =  0.1 The log loss is: 1.5108926636253637
For values of alpha =  1 The log loss is: 1.547101576655368
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9717790082855539
For values of best alpha =  0.0001 The cross validation log loss is: 1.244775
0915461886
For values of best alpha =  0.0001 The test log loss is: 1.1909868799788006
```

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 [266]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",
unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  235  gen
es in train dataset?
Ans
1. In test data 641 out of 665 : 96.39097744360903
2. In cross validation data 521 out of  532 : 97.93233082706767
```

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?

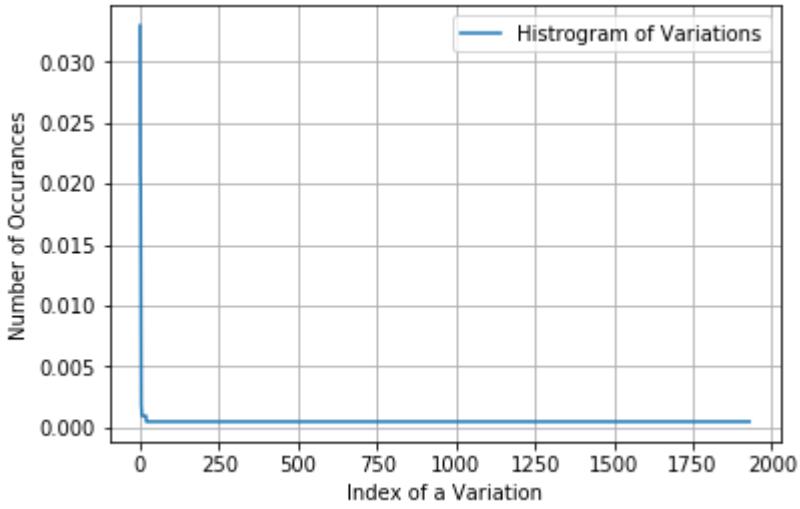
In [267]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1930
Truncating_Mutations    70
Amplification           44
Deletion                42
Fusions                 23
Overexpression           4
T58I                     3
Q61H                     2
R841K                    2
T286A                    2
G12A                     2
Name: Variation, dtype: int64
```

In [268]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

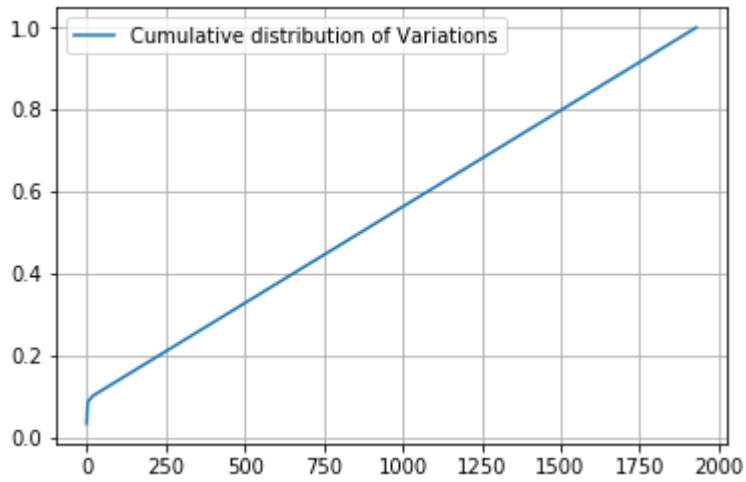
```
Ans: There are 1930 different categories of variations in the train data, and
they are distibuted as follows
```

In [269]:
```
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 [270]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.03295669 0.05367232 0.07344633 ... 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 [271]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [272]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [273]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(max_features = 1000)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [274]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1000)
```

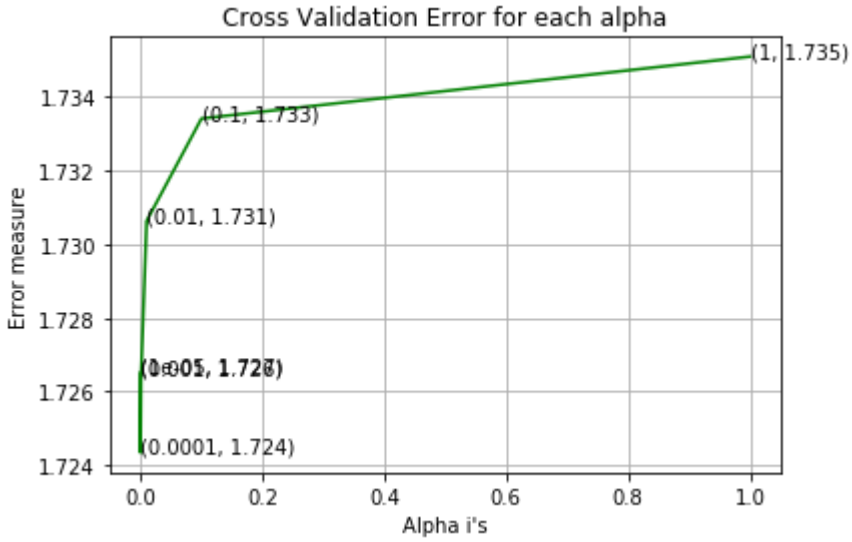
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [275]:
```
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_onehotCoding, y_train)

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

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, 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_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
```

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

```
For values of alpha =  1e-05 The log loss is: 1.7265099018680656
For values of alpha =  0.0001 The log loss is: 1.7243453988349362
For values of alpha =  0.001 The log loss is: 1.7264710124693656
For values of alpha =  0.01 The log loss is: 1.7306287920156882
For values of alpha =  0.1 The log loss is: 1.7334101722968704
For values of alpha =  1 The log loss is: 1.735090882207859
```




```
For values of best alpha =  0.0001 The train log loss is: 1.217085403287781
For values of best alpha =  0.0001 The cross validation log loss is: 1.724345
3988349362
For values of best alpha =  0.0001 The test log loss is: 1.7176768403644722
```

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 [276]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'
]))))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1930  genes in test and cross
validation data sets?
Ans
1. In test data 68 out of 665 : 10.225563909774436
2. In cross validation data 54 out of  532 : 10.150375939849624
```

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 [277]:
```
# 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 [278]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_di
ct.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(
row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [279]:
```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(max_features = 1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [280]:
```
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 [281]:
```
#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 [282]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train
_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [283]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=0)

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

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

In [284]:
```
#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 [285]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({254.43724077989026: 1, 180.75801031654228: 1, 135.6698250232942: 1,
131.06158228516995: 1, 126.72138804661232: 1, 120.37837923031616: 1, 120.1336
6292381725: 1, 117.38219113347685: 1, 112.48557765798671: 1, 109.073605081192
6: 1, 106.09707303908375: 1, 91.85404267252967: 1, 89.8296525852699: 1, 85.89
830961388228: 1, 81.98282050878447: 1, 81.30981102382073: 1, 79.5898671368013
3: 1, 78.30203852285089: 1, 76.62758382692955: 1, 76.31978672816477: 1, 73.97
797168464976: 1, 73.73718911904652: 1, 70.95303340458227: 1, 69.0902855163839
4: 1, 67.42508163679685: 1, 67.40977636329413: 1, 66.79153923110236: 1, 65.46
577850650777: 1, 64.44683885540273: 1, 64.2377442470851: 1, 64.2174305995174
6: 1, 63.848464866305584: 1, 62.192013650233214: 1, 60.24866216052387: 1, 58.
74019885323736: 1, 56.286873855362: 1, 56.1762316809323: 1, 54.4537466957101
6: 1, 54.23898449313715: 1, 51.0420490692118: 1, 50.83004400930236: 1, 50.350
31183274496: 1, 49.49721504481006: 1, 49.49206130541962: 1, 48.4599158272517
9: 1, 47.994311491699406: 1, 46.064555257868676: 1, 45.86027840403149: 1, 45.
76776128489948: 1, 44.967558704038254: 1, 44.31375580507636: 1, 43.4371302816
7759: 1, 43.334014486083795: 1, 43.0711894815711: 1, 43.02385440260525: 1, 4
2.94048250007119: 1, 42.62902457537603: 1, 42.30047473059397: 1, 42.199951587
12582: 1, 42.02859919991917: 1, 41.90078734066745: 1, 41.65003039241487: 1, 4
1.61867494251025: 1, 41.10210296388188: 1, 40.43832638021387: 1, 40.349628476
74777: 1, 39.83840759501087: 1, 39.74821542444531: 1, 39.73915230174761: 1, 3
9.3490531427697: 1, 39.01069711840668: 1, 38.49545776469339: 1, 38.4126077649
1183: 1, 38.38582055078087: 1, 37.254575881805124: 1, 36.96276098283079: 1, 3
6.62235761664208: 1, 36.53722099414352: 1, 36.41237325232565: 1, 36.069006406
659426: 1, 36.01348047523715: 1, 35.84567008951477: 1, 35.416829740491536: 1,
35.03438728992844: 1, 34.966088673767175: 1, 34.590808100013874: 1, 34.412589
810604175: 1, 34.281032806452124: 1, 34.23410279239548: 1, 34.13494078567615:
1, 33.80707526796277: 1, 33.52724874566354: 1, 32.721735315935014: 1, 32.6570
18708860065: 1, 32.63018048828395: 1, 32.40690081937138: 1, 32.39531917698257
5: 1, 32.31675089167369: 1, 32.286479064799416: 1, 32.096545532869094: 1, 32.
01650896091411: 1, 31.799062805832968: 1, 31.748162200290167: 1, 31.738946052
861003: 1, 31.687309499608727: 1, 31.637609334063825: 1, 31.42326456114082:
1, 31.206710484513966: 1, 31.085071199647782: 1, 30.994817172581435: 1, 30.99
037927345199: 1, 30.85400589513561: 1, 30.840214800866985: 1, 30.611028921589
618: 1, 30.5301016163881: 1, 30.309067471639576: 1, 30.147989826611475: 1, 3
0.09548082366658: 1, 29.994732862522337: 1, 29.790507909188726: 1, 29.6485216
5590859: 1, 29.490940925032156: 1, 29.42069471026214: 1, 29.407633170949172:
1, 29.400751833873375: 1, 29.324580442060988: 1, 28.919130154593315: 1, 28.90
1762524737094: 1, 28.759378530001726: 1, 28.603915198340854: 1, 28.5831875712
34312: 1, 28.030458288868758: 1, 27.70806733490654: 1, 27.69345222177571: 1,
27.682608393886966: 1, 27.67427655303743: 1, 27.66951180853718: 1, 27.3079210
7425332: 1, 27.2475825815999: 1, 27.047270642028188: 1, 26.997636457900736:
1, 26.70857464979773: 1, 26.69951416979122: 1, 26.451172659128712: 1, 26.4257
11689831722: 1, 26.39073694255188: 1, 26.319336762899702: 1, 26.1823627002540
68: 1, 26.080858686218363: 1, 25.7475252935411: 1, 25.70292023730297: 1, 25.5
73709763197783: 1, 25.523256546793622: 1, 25.394101847673532: 1, 25.106043077
411037: 1, 25.043296219609797: 1, 24.95610471447471: 1, 24.89129087392046: 1,
24.876210091816716: 1, 24.851770423282094: 1, 24.80413972703683: 1, 24.573946
819648423: 1, 24.555947198987436: 1, 24.48736877526404: 1, 24.41117633735405
3: 1, 24.40624754520678: 1, 24.363126400620672: 1, 24.305198991151098: 1, 24.
29972594722353: 1, 24.13736413470293: 1, 23.96182204823934: 1, 23.93368814797
5506: 1, 23.750095199027292: 1, 23.41913888131249: 1, 23.385432378776695: 1,
23.36711279971813: 1, 23.337617885776282: 1, 23.30495933940468: 1, 23.2708937
11484913: 1, 23.233773886318442: 1, 23.179041557560655: 1, 23.16993262327177
3: 1, 23.098296724618084: 1, 23.093282959038298: 1, 23.078944853851947: 1, 2
2.98841860493069: 1, 22.954248301212832: 1, 22.891470206225677: 1, 22.8168082
5209578: 1, 22.803296641773898: 1, 22.73535162352879: 1, 22.720736755448478:
1, 22.642317384113433: 1, 22.624350309735284: 1, 22.612654885314146: 1, 22.60
```

2023203642123: 1, 22.5124693261455: 1, 22.511175306473568: 1, 22.490726405729
415: 1, 22.434074025881063: 1, 22.429982451020994: 1, 22.36826394898403: 1, 2
2.32974243541341: 1, 22.28968982481478: 1, 22.234573521278335: 1, 22.22405444
9265584: 1, 22.139895010832934: 1, 22.126369287892718: 1, 21.953654498725285:
1, 21.879219598742605: 1, 21.75481413154622: 1, 21.741446678714876: 1, 21.675
383579346132: 1, 21.672017310598374: 1, 21.670763765102773: 1, 21.60428835783
0132: 1, 21.589626882343882: 1, 21.579918747200683: 1, 21.576430558576803: 1,
21.516981240847155: 1, 21.466551629836964: 1, 21.428506414127845: 1, 21.32027
5816592925: 1, 21.26338258603642: 1, 21.22831816909978: 1, 21.17689123677643
8: 1, 21.14682335059504: 1, 21.13509336545264: 1, 21.11730811772532: 1, 21.10
7656675492038: 1, 21.094301460383104: 1, 21.064085871196184: 1, 20.9123594386
95702: 1, 20.88195912643592: 1, 20.828801419429055: 1, 20.762237849725025: 1,
20.720975482603315: 1, 20.611077659116084: 1, 20.522928945586617: 1, 20.42353
3458226814: 1, 20.33895351968425: 1, 20.261248636464227: 1, 20.1382964328116
7: 1, 20.130803557693138: 1, 20.12941568338321: 1, 20.108657437974365: 1, 20.
054637058145737: 1, 20.043162076863386: 1, 19.95486966510669: 1, 19.887949963
81882: 1, 19.74662163020888: 1, 19.66878716889485: 1, 19.622504454672875: 1,
19.599314066129317: 1, 19.557467600943436: 1, 19.487476718868457: 1, 19.48026
2264420578: 1, 19.467908621603346: 1, 19.461470385424757: 1, 19.4481953278210
43: 1, 19.402309561983778: 1, 19.30516047920824: 1, 19.23174021680403: 1, 19.
2159476804407: 1, 19.182987211996483: 1, 19.12883162938933: 1, 19.12512205676
243: 1, 19.102908434069032: 1, 19.096086512248995: 1, 19.079360974995247: 1,
19.044830797216854: 1, 19.030924469241814: 1, 19.025386243550788: 1, 19.01471
838362724: 1, 18.95863849119765: 1, 18.92277673333053: 1, 18.86950505103169:
1, 18.825017022749062: 1, 18.805446151345635: 1, 18.77336501407797: 1, 18.766
632702761388: 1, 18.754989944941507: 1, 18.746037109243307: 1, 18.72438375540
2844: 1, 18.68928533140973: 1, 18.68898952131861: 1, 18.675638437171035: 1, 1
8.62508263927266: 1, 18.618578520552422: 1, 18.58261182173211: 1, 18.55700684
179162: 1, 18.55490680382832: 1, 18.545008227455977: 1, 18.537717696310438:
1, 18.40419777112467: 1, 18.309211906977968: 1, 18.174381907058617: 1, 18.156
50769027578: 1, 18.136157326854555: 1, 18.127434637195226: 1, 18.019628234214
487: 1, 17.999347581144548: 1, 17.99161465431819: 1, 17.979556248086578: 1, 1
7.959452564356177: 1, 17.956333632640206: 1, 17.949360204406396: 1, 17.885798
393403853: 1, 17.879755079590982: 1, 17.876919737125817: 1, 17.84688271494148
3: 1, 17.824634407943574: 1, 17.778099826702647: 1, 17.77747929039029: 1, 17.
736685256262863: 1, 17.727958006433102: 1, 17.685108677567367: 1, 17.67790016
452204: 1, 17.644685708679063: 1, 17.631018031795175: 1, 17.576542345772904:
1, 17.527171086381795: 1, 17.47778477942734: 1, 17.4525382086268: 1, 17.43779
9404735237: 1, 17.38643525802771: 1, 17.364968691413385: 1, 17.3512620235097:
1, 17.313666746654874: 1, 17.302602530414706: 1, 17.272208712703318: 1, 17.23
8651431658752: 1, 17.169204197821237: 1, 17.157962416443173: 1, 17.1395936187
5235: 1, 17.135271258693226: 1, 17.121915841719055: 1, 17.084712842745972: 1,
17.03340836783196: 1, 17.026308146004155: 1, 16.991982787999742: 1, 16.950217
593302753: 1, 16.949732844319183: 1, 16.913467454845982: 1, 16.85341383925668
5: 1, 16.8298521227837: 1, 16.782658296544476: 1, 16.701520597541613: 1, 16.6
8951154255465: 1, 16.658046417470082: 1, 16.649707136710376: 1, 16.6485566999
56384: 1, 16.64646173910655: 1, 16.646379238499026: 1, 16.633300425027905: 1,
16.591366023868527: 1, 16.58777929898807: 1, 16.564558304271948: 1, 16.562310
909499693: 1, 16.53372082356613: 1, 16.526946422408077: 1, 16.51736798705385:
1, 16.508025760917498: 1, 16.503843652511293: 1, 16.469026901769457: 1, 16.43
148928562557: 1, 16.376598745309636: 1, 16.34936240223864: 1, 16.337092668637
666: 1, 16.283761612049698: 1, 16.250074769942557: 1, 16.24145394252826: 1, 1
6.230836127572058: 1, 16.189299264025006: 1, 16.130194738474618: 1, 16.129372
77067567: 1, 16.069791423894657: 1, 16.062494084229655: 1, 16.05352618426543:
1, 16.02495462253771: 1, 16.01578712520345: 1, 15.979083621801482: 1, 15.9362
7334484074: 1, 15.875810869279439: 1, 15.849443251107605: 1, 15.8116501867481
55: 1, 15.786910322160226: 1, 15.775204400226354: 1, 15.754389269812046: 1, 1

5.742774529994218: 1, 15.693354055653854: 1, 15.677423851878482: 1, 15.665378
406884818: 1, 15.58636847401991: 1, 15.550244930936035: 1, 15.53915807847520
4: 1, 15.496315040448014: 1, 15.43907048188269: 1, 15.394583615300915: 1, 15.
35771961556613: 1, 15.35181010577117: 1, 15.307453081185464: 1, 15.2796992405
93226: 1, 15.236762730001953: 1, 15.231594104165074: 1, 15.212718708277272:
1, 15.197325487940544: 1, 15.194313802984528: 1, 15.171726256251183: 1, 15.13
4311077417685: 1, 15.117546721163894: 1, 15.105967908293149: 1, 15.0750338736
04387: 1, 15.014424370723878: 1, 14.998194757533009: 1, 14.978955980594357:
1, 14.964357193955275: 1, 14.964160043149702: 1, 14.94159160512808: 1, 14.941
413950309972: 1, 14.923070841795907: 1, 14.909671562284384: 1, 14.90330422051
4203: 1, 14.897459062839443: 1, 14.89036728775002: 1, 14.873275820134905: 1,
14.84115102685039: 1, 14.83991572260489: 1, 14.838905900717235: 1, 14.8293388
70351117: 1, 14.820636935455056: 1, 14.807593384001013: 1, 14.79190905946412
1: 1, 14.778724313792901: 1, 14.773336718868403: 1, 14.744541777367289: 1, 1
4.743270823184035: 1, 14.727262289172662: 1, 14.71961720175579: 1, 14.7117205
21163215: 1, 14.681910022594838: 1, 14.659658684236785: 1, 14.65494690208803:
1, 14.62932540015895: 1, 14.616345518693377: 1, 14.582838767569125: 1, 14.569
258809891808: 1, 14.51801831754388: 1, 14.473567937575064: 1, 14.464517039988
259: 1, 14.453326665722026: 1, 14.452826136950629: 1, 14.44207959314757: 1, 1
4.405523867982533: 1, 14.39191520491819: 1, 14.35350369849793: 1, 14.33419321
6139683: 1, 14.330421164382216: 1, 14.322650859014258: 1, 14.298640232530532:
1, 14.27514651385005: 1, 14.257481078856996: 1, 14.25227269947761: 1, 14.2410
40452214735: 1, 14.224763900948293: 1, 14.205493559532806: 1, 14.081999470187
69: 1, 14.039655056350338: 1, 14.037522933553413: 1, 14.028129440373995: 1, 1
3.985754738971012: 1, 13.95787592086994: 1, 13.950363863972042: 1, 13.9396112
10154757: 1, 13.918823844913389: 1, 13.887883451395583: 1, 13.88586555538001
8: 1, 13.878554061351476: 1, 13.854701406499256: 1, 13.804174348871019: 1, 1
3.76356186766743: 1, 13.75739326196597: 1, 13.747946153126072: 1, 13.66313089
1258039: 1, 13.66269197887456: 1, 13.572050650864213: 1, 13.55239497343505:
1, 13.544914913956529: 1, 13.53700526175076: 1, 13.516768618538315: 1, 13.446
06811219464: 1, 13.411341807221532: 1, 13.388853473079058: 1, 13.344969002720
308: 1, 13.328256595550412: 1, 13.32441219715363: 1, 13.299452132004976: 1, 1
3.296670435795537: 1, 13.274322517859641: 1, 13.216849026675025: 1, 13.189081
670451284: 1, 13.187517194714005: 1, 13.16606559783336: 1, 13.14561318639595:
1, 13.138546585734028: 1, 13.136425999472378: 1, 13.122696941775114: 1, 13.07
1162005560884: 1, 13.028567164354902: 1, 13.022603026306298: 1, 13.0026119258
91248: 1, 12.996827746315011: 1, 12.982067026867725: 1, 12.957896036211793:
1, 12.950399657519256: 1, 12.937417535868368: 1, 12.91070152510663: 1, 12.901
549555816175: 1, 12.888821950262598: 1, 12.883980843908525: 1, 12.88393550952
1224: 1, 12.867587849272551: 1, 12.820189556215949: 1, 12.818282489174369: 1,
12.79880390518542: 1, 12.78979263770619: 1, 12.76934292102794: 1, 12.68726186
874182: 1, 12.648705047243544: 1, 12.583673144367129: 1, 12.581870428182933:
1, 12.581576842939599: 1, 12.5421872254836: 1, 12.526582953888564: 1, 12.5124
09028803502: 1, 12.50325039888697: 1, 12.472897868981635: 1, 12.4626562438394
58: 1, 12.453036178400135: 1, 12.42619440077089: 1, 12.416537397287694: 1, 1
2.372184310679987: 1, 12.34567547644726: 1, 12.342028382614437: 1, 12.3202174
06346115: 1, 12.290355728191031: 1, 12.24546855096995: 1, 12.216707654441981:
1, 12.193321590926036: 1, 12.175786914160264: 1, 12.144673308103965: 1, 12.11
033378395041: 1, 12.099278840454556: 1, 12.070509404011931: 1, 12.06590169580
7396: 1, 12.052808919440235: 1, 12.030537891432544: 1, 12.010811184687732: 1,
12.010089778599509: 1, 12.009390005364924: 1, 11.972748404906339: 1, 11.97029
0690810774: 1, 11.896248430459993: 1, 11.895995528354883: 1, 11.8858840521702
17: 1, 11.881209531771976: 1, 11.87276351409194: 1, 11.844017783836517: 1, 1
1.836031189788518: 1, 11.829799981324562: 1, 11.826385908230217: 1, 11.817942
007683389: 1, 11.815370881714678: 1, 11.815220722998532: 1, 11.81458131352350
8: 1, 11.81079062333359: 1, 11.804509328417613: 1, 11.785891802802487: 1, 11.
767379814290733: 1, 11.752964455170272: 1, 11.739526104280701: 1, 11.72899985

4053603: 1, 11.673942440176663: 1, 11.671858061867184: 1, 11.665935857230684:
1, 11.659255012239177: 1, 11.646431215174083: 1, 11.639053709233865: 1, 11.63
5857173867603: 1, 11.611774961318728: 1, 11.599138862398872: 1, 11.5967922750
34506: 1, 11.577040808123456: 1, 11.574508894411307: 1, 11.568923100399765:
1, 11.567451206404218: 1, 11.556166873800354: 1, 11.549461228098275: 1, 11.54
805962192491: 1, 11.546918318168107: 1, 11.530447716427693: 1, 11.52812164917
6347: 1, 11.522244894497206: 1, 11.506689343808244: 1, 11.447937580472573: 1,
11.444834710615257: 1, 11.439495848120503: 1, 11.41378795813346: 1, 11.405282
571290623: 1, 11.386508879123864: 1, 11.385141928698298: 1, 11.3755335953263
7: 1, 11.33448285444614: 1, 11.330735695731043: 1, 11.328471670641479: 1, 11.
311707778579095: 1, 11.290822423479197: 1, 11.285200082496624: 1, 11.24337620
5667051: 1, 11.240636929553263: 1, 11.208679378960317: 1, 11.19662161662036:
1, 11.168281982183624: 1, 11.166630083943675: 1, 11.141928116731464: 1, 11.08
7033556748919: 1, 11.081686384513128: 1, 11.042643204518573: 1, 11.0403388258
43788: 1, 11.039536448574486: 1, 11.029482477802906: 1, 11.001101038992173:
1, 10.974827533198326: 1, 10.970105730302437: 1, 10.9430874358195: 1, 10.9210
21366524016: 1, 10.917260495793666: 1, 10.915778585107745: 1, 10.908304677901
889: 1, 10.905195963050343: 1, 10.860015384435027: 1, 10.852296131777942: 1,
10.840270134851224: 1, 10.838717311611198: 1, 10.835160019792337: 1, 10.83224
8181730545: 1, 10.818275642298751: 1, 10.807095730552124: 1, 10.7907630508289
22: 1, 10.788720796814486: 1, 10.78643511169873: 1, 10.767829125166037: 1, 1
0.74978214174368: 1, 10.748938498250052: 1, 10.74830401119868: 1, 10.74655033
3187484: 1, 10.739226666383736: 1, 10.721343686449206: 1, 10.721328582831704:
1, 10.702113930290619: 1, 10.701278488631127: 1, 10.70048024188204: 1, 10.667
553160838066: 1, 10.65130765252057: 1, 10.646604135355352: 1, 10.641449795926
793: 1, 10.63575579065587: 1, 10.634500266297437: 1, 10.630158531164417: 1, 1
0.588306869119373: 1, 10.561083157190149: 1, 10.55816832074098: 1, 10.5548401
87698296: 1, 10.552771325683318: 1, 10.55175460334814: 1, 10.549105901271076:
1, 10.547162173521171: 1, 10.516584411726084: 1, 10.489934609096673: 1, 10.48
4431371179841: 1, 10.466130309778295: 1, 10.464449469148423: 1, 10.4024107681
31105: 1, 10.391577845084772: 1, 10.379736605217188: 1, 10.378604110202465:
1, 10.374903609899706: 1, 10.356979042445586: 1, 10.332028457672651: 1, 10.32
4873497746598: 1, 10.31443862573766: 1, 10.308781972982564: 1, 10.29648873597
6629: 1, 10.284955317824378: 1, 10.240245877500858: 1, 10.233522625553443: 1,
10.218571330497479: 1, 10.159488442725552: 1, 10.15682720140524: 1, 10.148664
71604711: 1, 10.139876712408123: 1, 10.132042309419493: 1, 10.12530048280538:
1, 10.123827699512743: 1, 10.119204852490038: 1, 10.11679934867588: 1, 10.114
152575753565: 1, 10.104115410102095: 1, 10.071639216271238: 1, 10.05922532607
1248: 1, 10.058629910591852: 1, 10.058537398191481: 1, 10.029069912477281: 1,
10.018329801441448: 1, 10.009398305459369: 1, 9.9898261922: 1, 9.985694956147
63: 1, 9.981410151942507: 1, 9.964319962965968: 1, 9.964187567043828: 1, 9.95
9829891548775: 1, 9.952559207898357: 1, 9.929919087502316: 1, 9.9267340096510
4: 1, 9.895397276193657: 1, 9.893698031400707: 1, 9.892864745536059: 1, 9.882
03014070883: 1, 9.876673329082639: 1, 9.870658836608566: 1, 9.86490964821553
7: 1, 9.85944755934318: 1, 9.85811885545327: 1, 9.849187643487317: 1, 9.84275
1079924732: 1, 9.829945255165867: 1, 9.827983199870046: 1, 9.825618408014668:
1, 9.813369979223603: 1, 9.80236066262833: 1, 9.80172658165223: 1, 9.78947753
4872809: 1, 9.786815793565593: 1, 9.778737527141791: 1, 9.777763147238632: 1,
9.761332370110425: 1, 9.7538402612787: 1, 9.751327343058538: 1, 9.74906962503
863: 1, 9.748043393519582: 1, 9.74525319397831: 1, 9.735343577550735: 1, 9.72
783326104565: 1, 9.659400749274674: 1, 9.657703397399247: 1, 9.64862834210577
7: 1, 9.643125546064036: 1, 9.635423039182557: 1, 9.62278172793548: 1, 9.6181
49170408461: 1, 9.588069773983923: 1, 9.586437021516161: 1, 9.57119432305463
4: 1, 9.556219854094545: 1, 9.55575339377011: 1, 9.540659024635785: 1, 9.5298
98860203968: 1, 9.529260377709596: 1, 9.525646543343507: 1, 9.52462497825165
3: 1, 9.492660036980222: 1, 9.489419943416125: 1, 9.475860742236996: 1, 9.454
874574383142: 1, 9.451493923100474: 1, 9.451146344720485: 1, 9.45077420448705

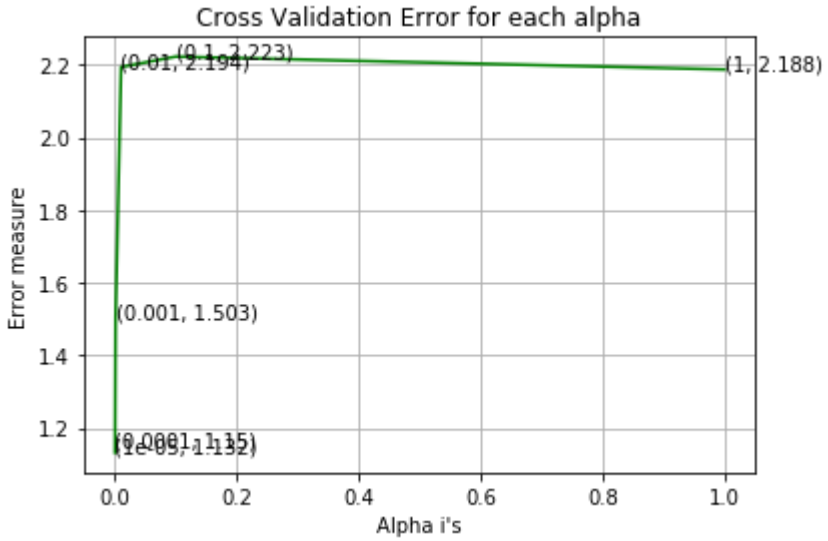
6: 1, 9.448676672073038: 1, 9.43339161303966: 1, 9.412362484153949: 1, 9.3956
77714202975: 1, 9.383650200735659: 1, 9.371931397034196: 1, 9.36485050496733
8: 1, 9.346549449177514: 1, 9.340819414572893: 1, 9.339970876617103: 1, 9.338
342551522725: 1, 9.323781475729108: 1, 9.322957397752255: 1, 9.31192860123252
3: 1, 9.309452513722126: 1, 9.309209086349064: 1, 9.307768053889172: 1, 9.288
661298961145: 1, 9.28578632287749: 1, 9.251338939739092: 1, 9.25023422125509
1: 1, 9.236438226719901: 1, 9.235294043178403: 1, 9.230312472532507: 1, 9.223
85218598756: 1, 9.217387016172957: 1, 9.214560716162861: 1, 9.20729884006329
5: 1, 9.197081552054467: 1, 9.191705038247825: 1, 9.179969130597192: 1, 9.171
733266258926: 1, 9.168417541504574: 1, 9.142411313834575: 1, 9.1342353575199
2: 1, 9.124968266795296: 1, 9.122511055982182: 1, 9.116986913324391: 1, 9.114
786431257986: 1, 9.093180655903597: 1, 9.092272753665306: 1, 9.0922095511512
9: 1, 9.071601097353632: 1, 9.05815077538072: 1, 9.033653405693414: 1, 9.0238
16042028836: 1, 9.01751537259764: 1, 9.015169045296561: 1, 9.009647236700818:
1, 9.008636719217751: 1, 9.001052122016146: 1, 8.995040232853157: 1, 8.960082
185255965: 1, 8.951989181008782: 1, 8.94501196298493: 1, 8.908740158522809:
1, 8.908539813620543: 1, 8.893515289841638: 1, 8.885207629723972: 1, 8.881299
603110492: 1, 8.854089893065897: 1, 8.834495856520887: 1, 8.832683903865208:
1, 8.828206840808198: 1, 8.819031785058417: 1, 8.802962742013651: 1, 8.801653
528845568: 1, 8.798544992277089: 1, 8.797617070738976: 1, 8.794336579318598:
1, 8.774819979924336: 1, 8.763170130713238: 1, 8.70909037499309: 1, 8.7070088
68869694: 1, 8.700789723232475: 1, 8.686903590896149: 1, 8.682642444632338:
1, 8.655698297916917: 1, 8.654255337407056: 1, 8.650559755571734: 1, 8.647122
951077293: 1, 8.639004082667714: 1, 8.635073789033026: 1, 8.632129974051866:
1, 8.602880267487189: 1, 8.581616815819082: 1, 8.581030852993454: 1, 8.560587
484219328: 1, 8.55076549877526: 1, 8.53500546441645: 1, 8.534895126269662: 1,
8.528367943345557: 1, 8.521025938530663: 1, 8.50648271819273: 1, 8.5029332448
898: 1, 8.497065085492869: 1, 8.483476890545992: 1, 8.482133689847739: 1, 8.4
76505808578201: 1, 8.460814909241227: 1, 8.43836859647207: 1, 8.4093700247820
83: 1, 8.40021533125622: 1, 8.398253036984645: 1, 8.380047861598522: 1, 8.343
335670938979: 1, 8.323641757325916: 1, 8.317305564358186: 1, 8.3143094517832
2: 1, 8.309546560833722: 1, 8.294264032128487: 1, 8.293706948177926: 1, 8.287
521340391432: 1, 8.26462446625491: 1, 8.24895605751441: 1, 8.232564633043472:
1, 8.223257793576536: 1, 8.203009674469746: 1, 8.197135156295396: 1, 8.194684
14600405: 1, 8.174727163970923: 1, 8.164559784364553: 1, 8.159441847131175:
1, 8.146925581025487: 1, 8.134203610744272: 1, 8.126641650111955: 1, 8.122838
86761959: 1, 8.12188815395335: 1, 8.105394736028465: 1, 8.102167985935182: 1,
8.09691626000542: 1, 8.096163317976815: 1, 8.095951024148846: 1, 8.0929449526
7592: 1, 8.085536182492708: 1, 8.069617155586657: 1, 8.058383683946522: 1, 8.
05679127565863: 1, 8.050951254032434: 1, 8.047141077584032: 1, 8.044255683320
822: 1, 8.039114559614903: 1, 8.024248779852861: 1, 7.995178437853961: 1, 7.9
894588222221445: 1, 7.986666153536008: 1, 7.970346394916617: 1, 7.96975094783
5646: 1, 7.955766887740374: 1, 7.949016512648956: 1, 7.935275142983067: 1, 7.
920383109240712: 1, 7.861674748309392: 1, 7.860466699526948: 1, 7.85575251423
98035: 1, 7.848039119915359: 1, 7.8462385084751745: 1, 7.825390568926308: 1,
7.817388138641146: 1, 7.803997644525179: 1, 7.798772610162897: 1, 7.787833633
21734: 1, 7.7736401449493195: 1, 7.7266833253329015: 1, 7.706701928028447: 1,
7.702462724017798: 1, 7.67284525749774: 1, 7.671149258228216: 1, 7.6682410693
51113: 1, 7.607013921100653: 1, 7.606685386217568: 1, 7.60643014255216: 1, 7.
605986203227384: 1, 7.588944642004582: 1, 7.558245685268743: 1, 7.55067983440
9957: 1, 7.542065854530562: 1, 7.541471268900396: 1, 7.5099671438763815: 1,
7.489644559223634: 1, 7.478873630559474: 1, 7.476945984184204: 1, 7.454354509
197147: 1, 7.4534885681948975: 1, 7.4332226490285125: 1, 7.414284485379659:
1, 7.3999001771047395: 1, 7.3889180870033515: 1, 7.378332440187493: 1, 7.3165
78881645736: 1, 7.29473735754095: 1, 7.291190468396402: 1, 7.289384444123678:
1, 7.285884651968864: 1, 7.276851465422521: 1, 7.275856494001829: 1, 7.261462
4849931655: 1, 7.210107743397758: 1, 7.187611846892729: 1, 7.180671842413738:

```
1, 7.170472349498599: 1, 7.169049176152391: 1, 7.1631910311286795: 1, 7.15777
93605009665: 1, 7.1087463498677765: 1, 7.100273820423581: 1, 7.09936568995090
9: 1, 7.083214827155227: 1, 7.043311692888054: 1, 7.032221369395034: 1, 7.020
561322410519: 1, 6.993144680328319: 1, 6.960157301448914: 1, 6.9008722557223
6: 1, 6.873307047340329: 1, 6.856623170502155: 1, 6.732587179658522: 1, 6.650
728965163763: 1, 6.55378987820564: 1, 6.532085268448021: 1, 6.45537476761725:
1, 5.962024324322491: 1, 5.87694389389109: 1})
```

In [286]:
```
# Train a Logistic regression+Calibration model using text features whicha re
 on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
```

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

```
For values of alpha =  1e-05 The log loss is: 1.132309743800044
For values of alpha =  0.0001 The log loss is: 1.1500715715576066
For values of alpha =  0.001 The log loss is: 1.5033421388127168
For values of alpha =  0.01 The log loss is: 2.194291667607671
For values of alpha =  0.1 The log loss is: 2.222984196013969
For values of alpha =  1 The log loss is: 2.1875523810179227
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7057366393162506
For values of best alpha =  1e-05 The cross validation log loss is: 1.1323097
43800044
For values of best alpha =  1e-05 The test log loss is: 1.1729615767798298
```

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

Ans. Yes, it seems like!

In [287]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features = 1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [288]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train
 data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in
train data")
```

```
94.5 % of word of test data appeared in train data
94.2 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [289]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities bel
ongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_
y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [290]:
```
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 [291]:
```
# 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(max_features = 1000)
    var_count_vec = TfidfVectorizer(max_features = 1000)
    text_count_vec = TfidfVectorizer(max_features = 1000)

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

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

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

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

Stacking the three types of features

In [292]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,t
rain_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,tes
t_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_vari
ation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_
feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_fea
ture_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_r
esponseCoding))
```

In [293]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2234)
(number of data points * number of features) in test data =  (665, 2234)
(number of data points * number of features) in cross validation data = (532, 2234)
```

In [294]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

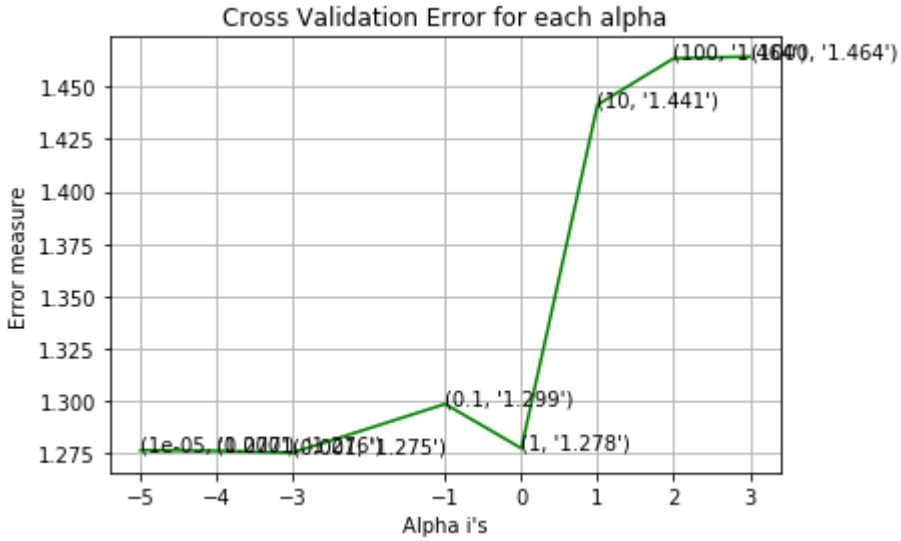
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [295]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_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(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
```

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


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


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

```
for alpha = 1e-05
Log Loss : 1.276613840599791
for alpha = 0.0001
Log Loss : 1.276346328699647
for alpha = 0.001
Log Loss : 1.2754470148386734
for alpha = 0.1
Log Loss : 1.298655265292885
for alpha = 1
Log Loss : 1.2775183018835277
for alpha = 10
Log Loss : 1.441429428775323
for alpha = 100
Log Loss : 1.4635961891371512
for alpha = 1000
Log Loss : 1.4644204483539731
```

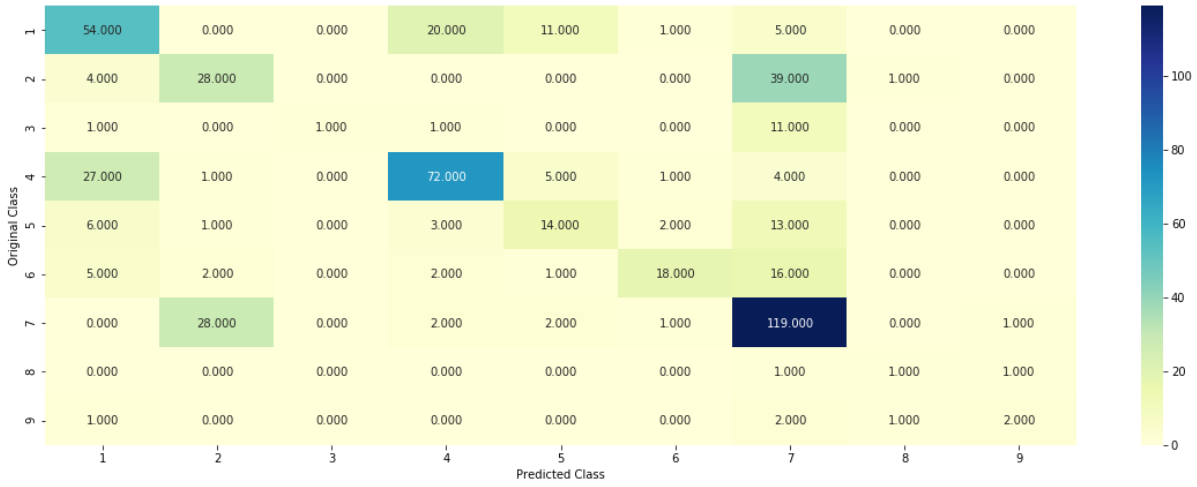



```
For values of best alpha =  0.001 The train log loss is: 0.7242335065597271
For values of best alpha =  0.001 The cross validation log loss is: 1.2754470
148386734
For values of best alpha =  0.001 The test log loss is: 1.2165881875100049
```

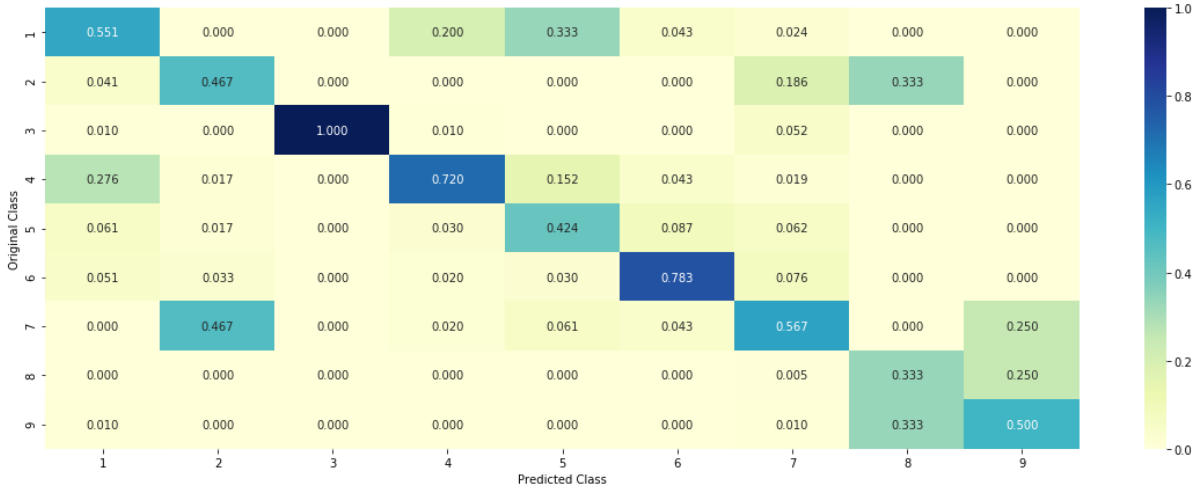
4.1.1.2. Testing the model with best hyper paramters

In [296]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

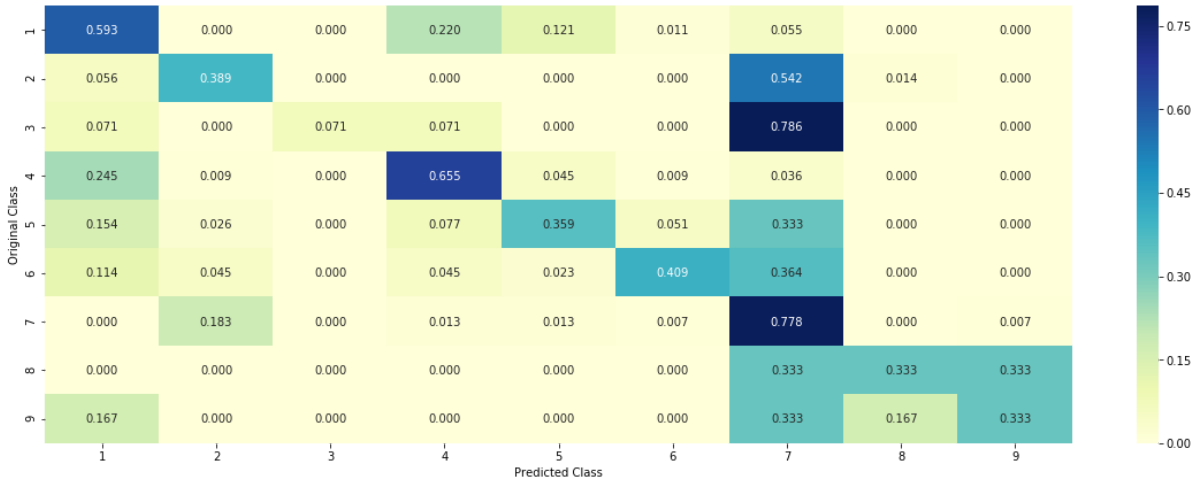
```
Log Loss : 1.2754470148386734
Number of missclassified point : 0.4191729323308271
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Incorrectly classified point

In [297]:
```
test_point_index = 2
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.129  0.0471 0.0147 0.6669 0.0342 0.0337 0.
0698 0.0028 0.0018]]
Actual Class : 4
--------------------------------------------------
8 Text feature [activity] present in test data point [True]
15 Text feature [protein] present in test data point [True]
16 Text feature [experiments] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [whereas] present in test data point [True]
20 Text feature [acid] present in test data point [True]
21 Text feature [results] present in test data point [True]
22 Text feature [amino] present in test data point [True]
24 Text feature [shown] present in test data point [True]
25 Text feature [type] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [wild] present in test data point [True]
28 Text feature [catalytic] present in test data point [True]
30 Text feature [important] present in test data point [True]
32 Text feature [determined] present in test data point [True]
34 Text feature [described] present in test data point [True]
35 Text feature [two] present in test data point [True]
36 Text feature [although] present in test data point [True]
37 Text feature [purified] present in test data point [True]
38 Text feature [may] present in test data point [True]
43 Text feature [indicate] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
46 Text feature [discussion] present in test data point [True]
47 Text feature [indicated] present in test data point [True]
51 Text feature [containing] present in test data point [True]
56 Text feature [show] present in test data point [True]
57 Text feature [three] present in test data point [True]
61 Text feature [30] present in test data point [True]
62 Text feature [bind] present in test data point [True]
63 Text feature [similar] present in test data point [True]
64 Text feature [previously] present in test data point [True]
65 Text feature [thus] present in test data point [True]
66 Text feature [associated] present in test data point [True]
67 Text feature [generated] present in test data point [True]
69 Text feature [however] present in test data point [True]
72 Text feature [see] present in test data point [True]
73 Text feature [analysis] present in test data point [True]
74 Text feature [residues] present in test data point [True]
77 Text feature [using] present in test data point [True]
80 Text feature [substitutions] present in test data point [True]
81 Text feature [effects] present in test data point [True]
83 Text feature [addition] present in test data point [True]
84 Text feature [could] present in test data point [True]
87 Text feature [contribute] present in test data point [True]
88 Text feature [several] present in test data point [True]
89 Text feature [one] present in test data point [True]
90 Text feature [mutants] present in test data point [True]
91 Text feature [10] present in test data point [True]
92 Text feature [data] present in test data point [True]
93 Text feature [assay] present in test data point [True]
94 Text feature [used] present in test data point [True]
95 Text feature [site] present in test data point [True]
```

```
97 Text feature [performed] present in test data point [True]
99 Text feature [involved] present in test data point [True]
Out of the top  100  features  54 are present in query point
```

4.1.1.4. Feature Importance, Correctly classified point

In [298]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3179 0.0509 0.0159 0.4607 0.0372 0.0364 0.
076  0.0031 0.002 ]]
Actual Class : 4
--------------------------------------------------
8 Text feature [activity] present in test data point [True]
9 Text feature [proteins] present in test data point [True]
15 Text feature [protein] present in test data point [True]
16 Text feature [experiments] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [whereas] present in test data point [True]
20 Text feature [acid] present in test data point [True]
21 Text feature [results] present in test data point [True]
22 Text feature [amino] present in test data point [True]
24 Text feature [shown] present in test data point [True]
25 Text feature [type] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [wild] present in test data point [True]
29 Text feature [missense] present in test data point [True]
30 Text feature [important] present in test data point [True]
31 Text feature [functional] present in test data point [True]
32 Text feature [determined] present in test data point [True]
34 Text feature [described] present in test data point [True]
35 Text feature [two] present in test data point [True]
36 Text feature [although] present in test data point [True]
38 Text feature [may] present in test data point [True]
39 Text feature [whether] present in test data point [True]
42 Text feature [mammalian] present in test data point [True]
43 Text feature [indicate] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [therefore] present in test data point [True]
46 Text feature [discussion] present in test data point [True]
47 Text feature [indicated] present in test data point [True]
48 Text feature [reduced] present in test data point [True]
50 Text feature [lower] present in test data point [True]
51 Text feature [containing] present in test data point [True]
55 Text feature [either] present in test data point [True]
57 Text feature [three] present in test data point [True]
58 Text feature [loss] present in test data point [True]
59 Text feature [levels] present in test data point [True]
60 Text feature [determine] present in test data point [True]
63 Text feature [similar] present in test data point [True]
64 Text feature [previously] present in test data point [True]
66 Text feature [associated] present in test data point [True]
68 Text feature [stability] present in test data point [True]
69 Text feature [however] present in test data point [True]
70 Text feature [vivo] present in test data point [True]
71 Text feature [buffer] present in test data point [True]
72 Text feature [see] present in test data point [True]
73 Text feature [analysis] present in test data point [True]
76 Text feature [affect] present in test data point [True]
77 Text feature [using] present in test data point [True]
78 Text feature [indicates] present in test data point [True]
80 Text feature [substitutions] present in test data point [True]
81 Text feature [effects] present in test data point [True]
83 Text feature [addition] present in test data point [True]
84 Text feature [could] present in test data point [True]
```

```
85 Text feature [effect] present in test data point [True]
86 Text feature [acids] present in test data point [True]
87 Text feature [contribute] present in test data point [True]
88 Text feature [several] present in test data point [True]
89 Text feature [one] present in test data point [True]
91 Text feature [10] present in test data point [True]
92 Text feature [data] present in test data point [True]
93 Text feature [assay] present in test data point [True]
94 Text feature [used] present in test data point [True]
95 Text feature [site] present in test data point [True]
96 Text feature [cells] present in test data point [True]
97 Text feature [performed] present in test data point [True]
98 Text feature [result] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

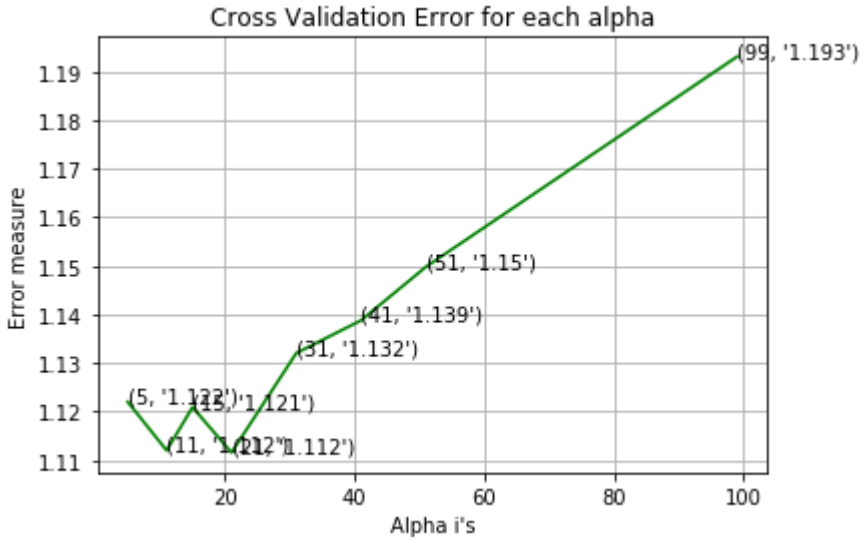
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [299]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


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

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

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(train_y, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 log loss is:",log_loss(cv_y, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(test_y, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.1220444712838222
for alpha = 11
Log Loss : 1.1120257702339105
for alpha = 15
Log Loss : 1.1209389851939933
for alpha = 21
Log Loss : 1.1115809774947647
for alpha = 31
Log Loss : 1.1320663965177107
for alpha = 41
Log Loss : 1.1388027727918437
for alpha = 51
Log Loss : 1.1498209978976601
for alpha = 99
Log Loss : 1.1930932492434763
```

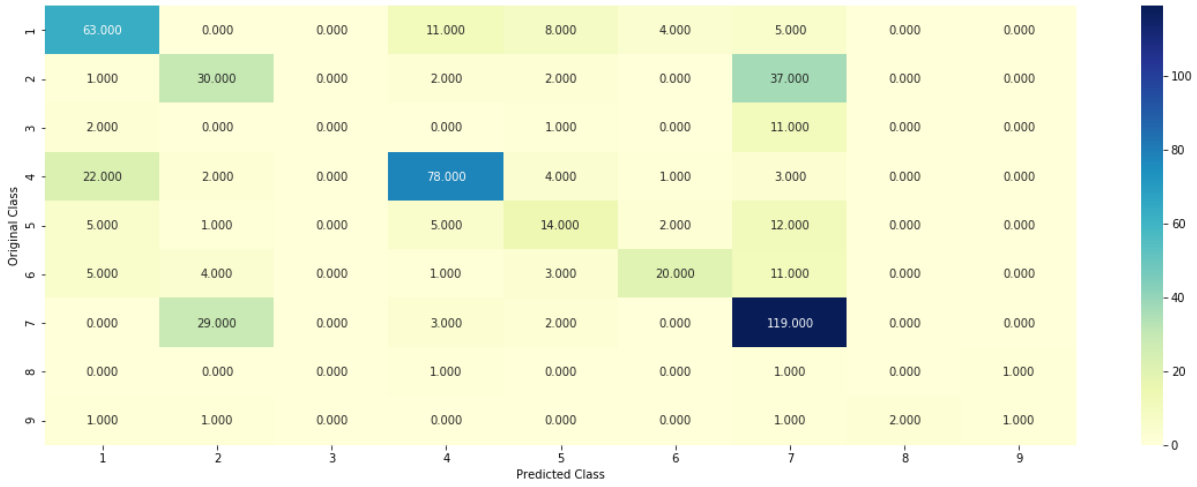



```
For values of best alpha =  21 The train log loss is: 0.7319327135311923
For values of best alpha =  21 The cross validation log loss is: 1.1115809774
947647
For values of best alpha =  21 The test log loss is: 1.1175523552270537
```

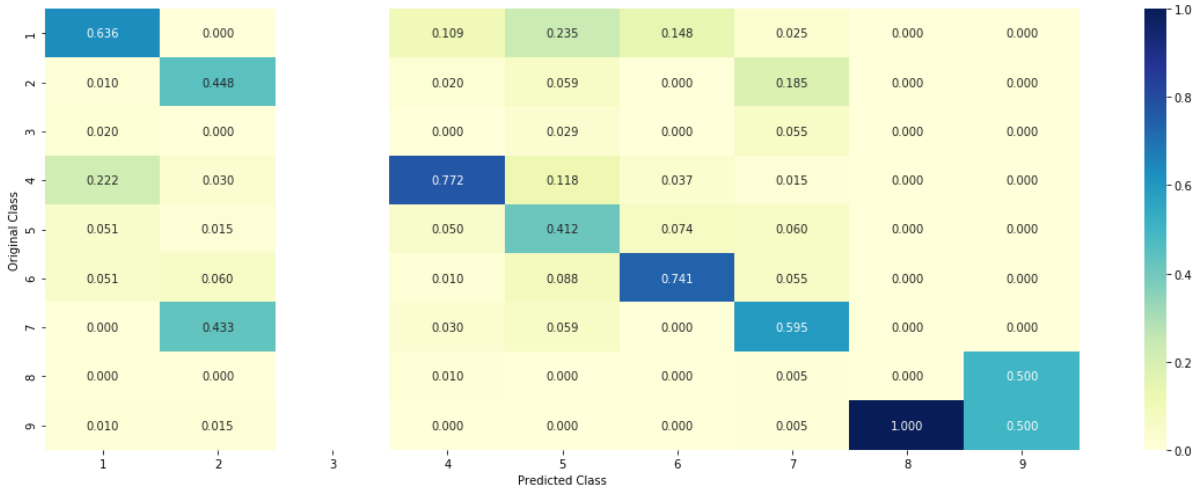
4.2.2. Testing the model with best hyper paramters

In [300]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

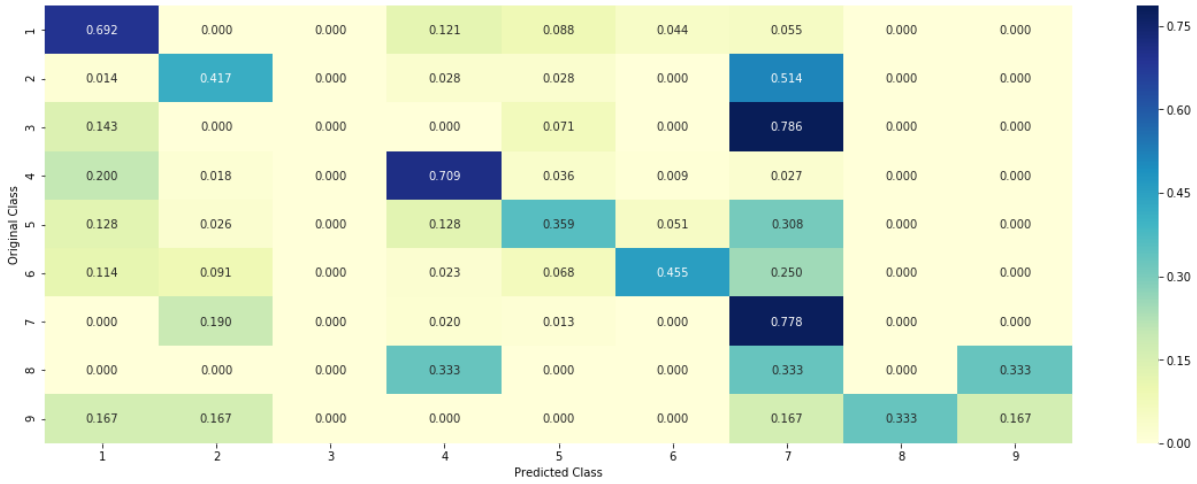
```
Log loss : 1.1115809774947647
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [301]:
```
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 = 3
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1,
-1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs
to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 4
The  21  nearest neighbours of the test points belongs to classes [1 1 1 1 1
1 4 4 1 1 1 4 1 1 9 1 4 1 1 4 1]
Fequency of nearest points : Counter({1: 15, 4: 5, 9: 1})
```

4.2.4. Sample Query Point-2

In [302]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

```
Predicted Class : 1
Actual Class : 4
the k value for knn is 21 and the nearest neighbours of the test points belon
gs to classes [1 1 1 4 1 1 1 4 1 4 1 4 4 4 1 4 4 7 1 4 6]
Fequency of nearest points : Counter({1: 10, 4: 9, 7: 1, 6: 1})
```

4.3. Logistic Regression

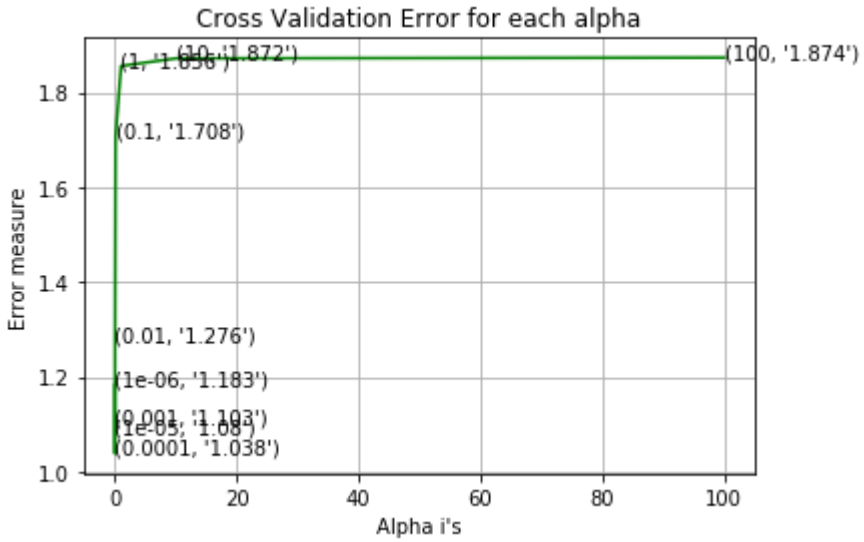
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [303]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

```
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

```
for alpha = 1e-06
Log Loss : 1.183461561268613
for alpha = 1e-05
Log Loss : 1.0802841606447489
for alpha = 0.0001
Log Loss : 1.0377619532836317
for alpha = 0.001
Log Loss : 1.102753640452913
for alpha = 0.01
Log Loss : 1.2756496529168857
for alpha = 0.1
Log Loss : 1.7077939074804773
for alpha = 1
Log Loss : 1.855824969915548
for alpha = 10
Log Loss : 1.8718678755034446
for alpha = 100
Log Loss : 1.873547344772738
```

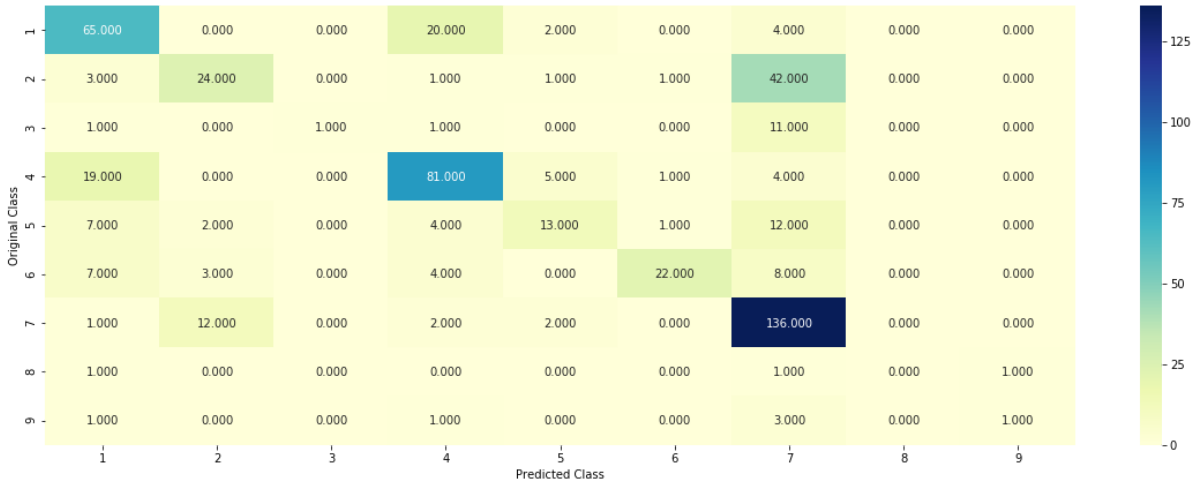



```
For values of best alpha =  0.0001 The train log loss is: 0.551403319132685
For values of best alpha =  0.0001 The cross validation log loss is: 1.037761
9532836317
For values of best alpha =  0.0001 The test log loss is: 1.047534905120887
```

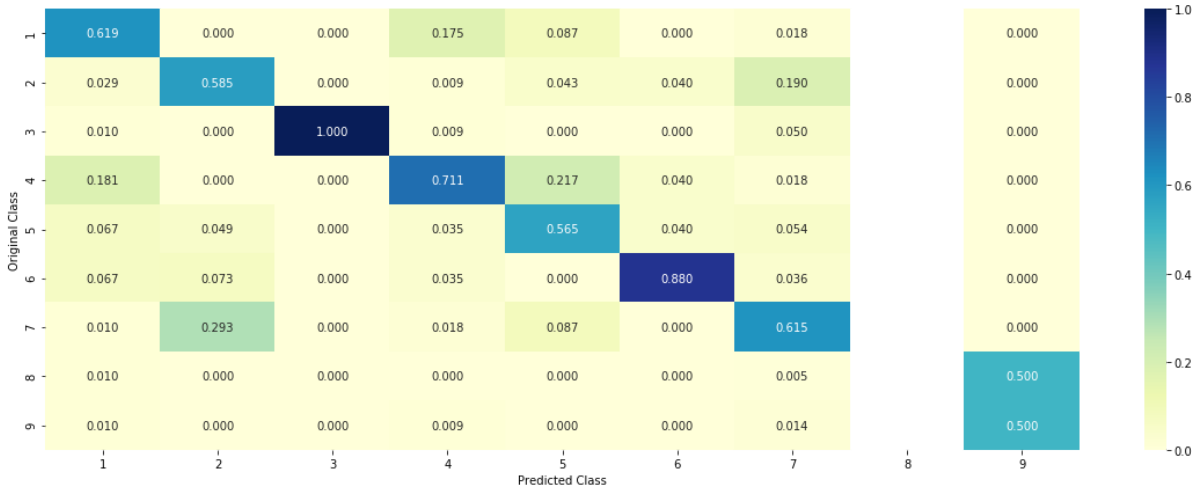
4.3.1.2. Testing the model with best hyper paramters

In [304]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

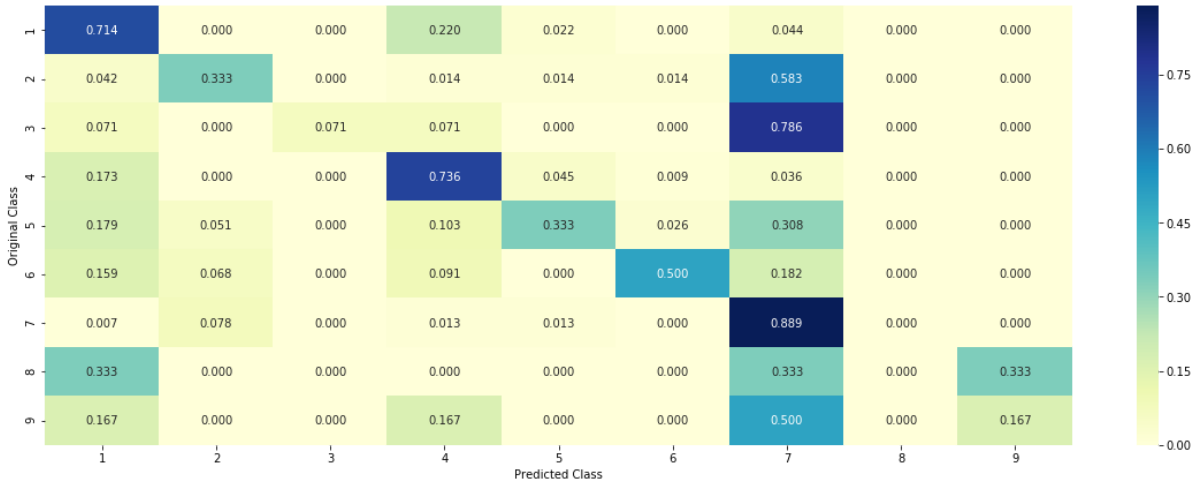
```
Log loss : 1.0377619532836317
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Feature Importance

In [305]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], ye
s_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query poin
t")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," cl
ass:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or
Not']))
```

4.3.1.3.1. Correctly Classified point

In [306]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3448 0.0065 0.0038 0.6046 0.0042 0.0296 0.
0043 0.0014 0.0008]]
Actual Class : 4
--------------------------------------------------
77 Text feature [see] present in test data point [True]
83 Text feature [inactivation] present in test data point [True]
110 Text feature [show] present in test data point [True]
112 Text feature [unable] present in test data point [True]
124 Text feature [protein] present in test data point [True]
143 Text feature [4a] present in test data point [True]
156 Text feature [nature] present in test data point [True]
165 Text feature [high] present in test data point [True]
178 Text feature [representative] present in test data point [True]
184 Text feature [ref] present in test data point [True]
185 Text feature [specifically] present in test data point [True]
188 Text feature [recognition] present in test data point [True]
193 Text feature [deletion] present in test data point [True]
194 Text feature [yeast] present in test data point [True]
209 Text feature [substrate] present in test data point [True]
214 Text feature [purified] present in test data point [True]
226 Text feature [catalytic] present in test data point [True]
227 Text feature [cannot] present in test data point [True]
254 Text feature [indicate] present in test data point [True]
265 Text feature [differences] present in test data point [True]
277 Text feature [3b] present in test data point [True]
280 Text feature [mm] present in test data point [True]
301 Text feature [contribute] present in test data point [True]
303 Text feature [rate] present in test data point [True]
306 Text feature [particular] present in test data point [True]
307 Text feature [described] present in test data point [True]
308 Text feature [29] present in test data point [True]
309 Text feature [bind] present in test data point [True]
313 Text feature [standard] present in test data point [True]
323 Text feature [suggested] present in test data point [True]
326 Text feature [represent] present in test data point [True]
327 Text feature [assay] present in test data point [True]
333 Text feature [1b] present in test data point [True]
336 Text feature [larger] present in test data point [True]
340 Text feature [regions] present in test data point [True]
341 Text feature [family] present in test data point [True]
342 Text feature [bound] present in test data point [True]
353 Text feature [recent] present in test data point [True]
360 Text feature [performed] present in test data point [True]
365 Text feature [2a] present in test data point [True]
367 Text feature [reaction] present in test data point [True]
369 Text feature [led] present in test data point [True]
383 Text feature [consequences] present in test data point [True]
392 Text feature [200] present in test data point [True]
397 Text feature [decreased] present in test data point [True]
402 Text feature [several] present in test data point [True]
404 Text feature [blue] present in test data point [True]
406 Text feature [terminal] present in test data point [True]
407 Text feature [significantly] present in test data point [True]
417 Text feature [characterized] present in test data point [True]
420 Text feature [key] present in test data point [True]
424 Text feature [recombinant] present in test data point [True]
```

```
427 Text feature [shows] present in test data point [True]
428 Text feature [co] present in test data point [True]
429 Text feature [complete] present in test data point [True]
434 Text feature [motif] present in test data point [True]
438 Text feature [experiments] present in test data point [True]
440 Text feature [supplementary] present in test data point [True]
441 Text feature [phase] present in test data point [True]
443 Text feature [formation] present in test data point [True]
447 Text feature [information] present in test data point [True]
453 Text feature [absence] present in test data point [True]
454 Text feature [strand] present in test data point [True]
458 Text feature [residue] present in test data point [True]
460 Text feature [considered] present in test data point [True]
469 Text feature [1a] present in test data point [True]
471 Text feature [respectively] present in test data point [True]
473 Text feature [primary] present in test data point [True]
477 Text feature [negative] present in test data point [True]
478 Text feature [observed] present in test data point [True]
479 Text feature [affinity] present in test data point [True]
487 Text feature [mutants] present in test data point [True]
489 Text feature [product] present in test data point [True]
491 Text feature [figure] present in test data point [True]
492 Text feature [despite] present in test data point [True]
496 Text feature [length] present in test data point [True]
Out of the top  500  features  76 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [307]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4856 0.0151 0.0119 0.4097 0.0202 0.0156 0.
0331 0.0057 0.0032]]
Actual Class : 4
--------------------------------------------------
19 Text feature [hydrophobic] present in test data point [True]
91 Text feature [function] present in test data point [True]
92 Text feature [encoding] present in test data point [True]
136 Text feature [structure] present in test data point [True]
146 Text feature [21] present in test data point [True]
152 Text feature [indicated] present in test data point [True]
153 Text feature [affecting] present in test data point [True]
160 Text feature [signal] present in test data point [True]
167 Text feature [subjected] present in test data point [True]
172 Text feature [therefore] present in test data point [True]
173 Text feature [signals] present in test data point [True]
177 Text feature [reduced] present in test data point [True]
179 Text feature [coding] present in test data point [True]
182 Text feature [codon] present in test data point [True]
183 Text feature [region] present in test data point [True]
189 Text feature [mean] present in test data point [True]
191 Text feature [loss] present in test data point [True]
192 Text feature [next] present in test data point [True]
193 Text feature [page] present in test data point [True]
194 Text feature [05] present in test data point [True]
201 Text feature [population] present in test data point [True]
207 Text feature [less] present in test data point [True]
211 Text feature [role] present in test data point [True]
212 Text feature [showing] present in test data point [True]
215 Text feature [2001] present in test data point [True]
221 Text feature [2006] present in test data point [True]
225 Text feature [pa] present in test data point [True]
228 Text feature [assess] present in test data point [True]
232 Text feature [assessment] present in test data point [True]
233 Text feature [directed] present in test data point [True]
238 Text feature [constructs] present in test data point [True]
244 Text feature [screening] present in test data point [True]
247 Text feature [value] present in test data point [True]
249 Text feature [even] present in test data point [True]
254 Text feature [essential] present in test data point [True]
257 Text feature [www] present in test data point [True]
258 Text feature [remaining] present in test data point [True]
260 Text feature [one] present in test data point [True]
261 Text feature [binding] present in test data point [True]
264 Text feature [construct] present in test data point [True]
265 Text feature [change] present in test data point [True]
276 Text feature [affect] present in test data point [True]
278 Text feature [s1] present in test data point [True]
280 Text feature [nucleotide] present in test data point [True]
281 Text feature [reporter] present in test data point [True]
282 Text feature [previous] present in test data point [True]
290 Text feature [another] present in test data point [True]
299 Text feature [19] present in test data point [True]
303 Text feature [tested] present in test data point [True]
307 Text feature [seven] present in test data point [True]
312 Text feature [analyzed] present in test data point [True]
314 Text feature [antibodies] present in test data point [True]
```

```
321 Text feature [2008] present in test data point [True]
327 Text feature [assessed] present in test data point [True]
330 Text feature [ratio] present in test data point [True]
333 Text feature [sds] present in test data point [True]
335 Text feature [criteria] present in test data point [True]
336 Text feature [medium] present in test data point [True]
340 Text feature [based] present in test data point [True]
343 Text feature [pathogenic] present in test data point [True]
345 Text feature [diagnosis] present in test data point [True]
348 Text feature [protein] present in test data point [True]
349 Text feature [carrying] present in test data point [True]
351 Text feature [effect] present in test data point [True]
365 Text feature [http] present in test data point [True]
370 Text feature [across] present in test data point [True]
372 Text feature [resulted] present in test data point [True]
373 Text feature [provide] present in test data point [True]
378 Text feature [small] present in test data point [True]
379 Text feature [possible] present in test data point [True]
380 Text feature [within] present in test data point [True]
382 Text feature [gene] present in test data point [True]
388 Text feature [furthermore] present in test data point [True]
389 Text feature [individual] present in test data point [True]
390 Text feature [vivo] present in test data point [True]
391 Text feature [2002] present in test data point [True]
394 Text feature [significantly] present in test data point [True]
395 Text feature [evidence] present in test data point [True]
397 Text feature [selected] present in test data point [True]
400 Text feature [strong] present in test data point [True]
402 Text feature [corresponding] present in test data point [True]
403 Text feature [relative] present in test data point [True]
405 Text feature [added] present in test data point [True]
408 Text feature [type] present in test data point [True]
409 Text feature [total] present in test data point [True]
412 Text feature [least] present in test data point [True]
413 Text feature [frame] present in test data point [True]
422 Text feature [cell] present in test data point [True]
423 Text feature [17] present in test data point [True]
424 Text feature [limited] present in test data point [True]
425 Text feature [wild] present in test data point [True]
427 Text feature [derived] present in test data point [True]
430 Text feature [whether] present in test data point [True]
435 Text feature [mutation] present in test data point [True]
437 Text feature [splicing] present in test data point [True]
438 Text feature [sequence] present in test data point [True]
440 Text feature [37] present in test data point [True]
441 Text feature [vector] present in test data point [True]
443 Text feature [incubated] present in test data point [True]
445 Text feature [three] present in test data point [True]
448 Text feature [line] present in test data point [True]
449 Text feature [localization] present in test data point [True]
454 Text feature [genetic] present in test data point [True]
455 Text feature [assays] present in test data point [True]
457 Text feature [mutagenesis] present in test data point [True]
460 Text feature [following] present in test data point [True]
461 Text feature [red] present in test data point [True]
476 Text feature [39] present in test data point [True]
481 Text feature [inactivation] present in test data point [True]
```

```
483 Text feature [co] present in test data point [True]
489 Text feature [either] present in test data point [True]
491 Text feature [sequencing] present in test data point [True]
497 Text feature [human] present in test data point [True]
Out of the top  500  features  113 are present in query point
```

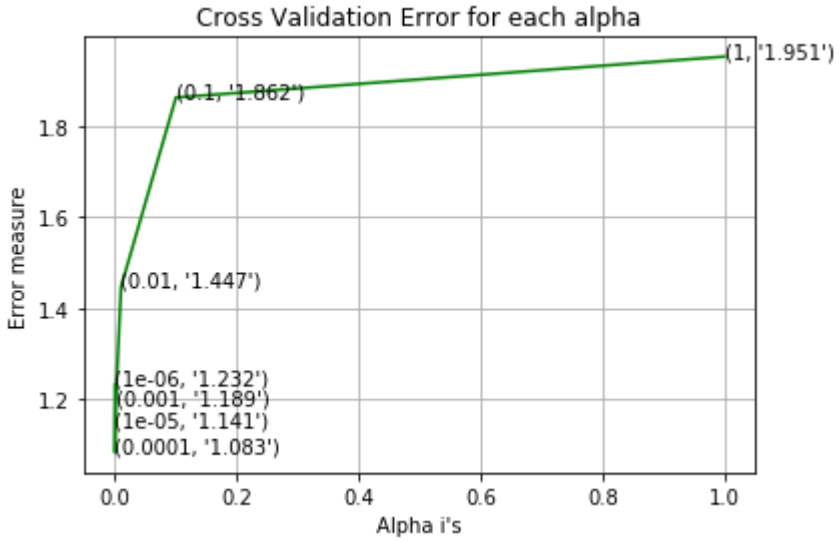
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [308]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2319853730156791
for alpha = 1e-05
Log Loss : 1.1405388366332843
for alpha = 0.0001
Log Loss : 1.083242810616072
for alpha = 0.001
Log Loss : 1.1885178022438982
for alpha = 0.01
Log Loss : 1.4470263002186137
for alpha = 0.1
Log Loss : 1.861708341683483
for alpha = 1
Log Loss : 1.951353746909
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5407787291452905
For values of best alpha =  0.0001 The cross validation log loss is: 1.083242
810616072
For values of best alpha =  0.0001 The test log loss is: 1.068020509856545
```

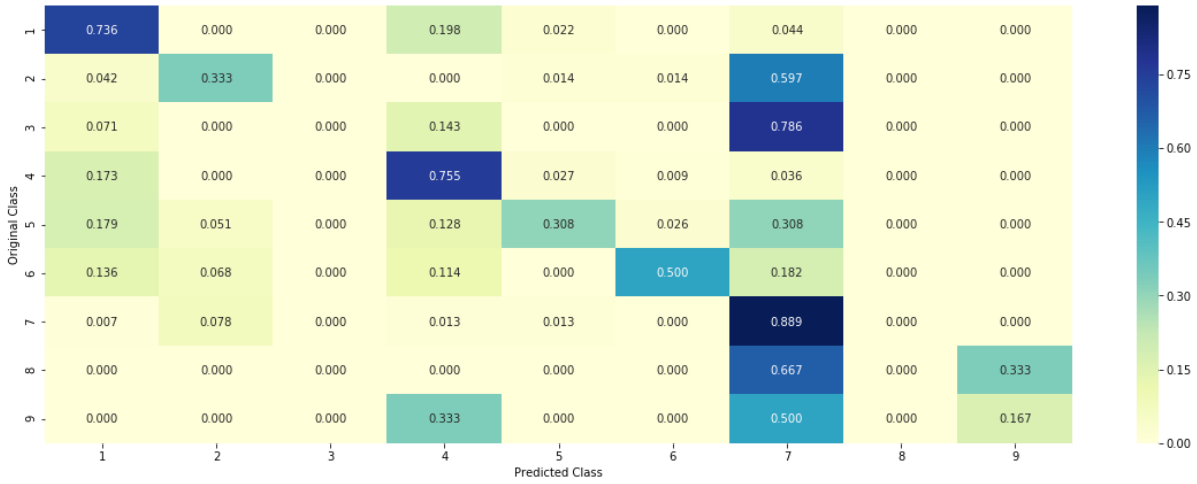
4.3.2.2. Testing model with best hyper parameters

In [309]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.083242810616072
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```


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


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



4.3.2.3. Feature Importance, InCorrectly Classified point

In [310]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[3.865e-01 7.200e-03 1.600e-03 5.681e-01 3.50
0e-03 2.740e-02 5.100e-03
  4.000e-04 3.000e-04]]
Actual Class : 4
--------------------------------------------------
61 Text feature [see] present in test data point [True]
89 Text feature [unable] present in test data point [True]
93 Text feature [protein] present in test data point [True]
98 Text feature [show] present in test data point [True]
108 Text feature [inactivation] present in test data point [True]
113 Text feature [high] present in test data point [True]
117 Text feature [4a] present in test data point [True]
124 Text feature [representative] present in test data point [True]
155 Text feature [yeast] present in test data point [True]
157 Text feature [nature] present in test data point [True]
159 Text feature [specifically] present in test data point [True]
173 Text feature [cannot] present in test data point [True]
183 Text feature [differences] present in test data point [True]
192 Text feature [deletion] present in test data point [True]
196 Text feature [ref] present in test data point [True]
203 Text feature [substrate] present in test data point [True]
244 Text feature [3b] present in test data point [True]
250 Text feature [catalytic] present in test data point [True]
251 Text feature [bind] present in test data point [True]
256 Text feature [recognition] present in test data point [True]
258 Text feature [indicate] present in test data point [True]
264 Text feature [described] present in test data point [True]
269 Text feature [29] present in test data point [True]
272 Text feature [particular] present in test data point [True]
275 Text feature [suggested] present in test data point [True]
276 Text feature [purified] present in test data point [True]
296 Text feature [rate] present in test data point [True]
298 Text feature [contribute] present in test data point [True]
299 Text feature [assay] present in test data point [True]
301 Text feature [represent] present in test data point [True]
302 Text feature [1b] present in test data point [True]
308 Text feature [bound] present in test data point [True]
312 Text feature [larger] present in test data point [True]
315 Text feature [led] present in test data point [True]
318 Text feature [recent] present in test data point [True]
320 Text feature [standard] present in test data point [True]
323 Text feature [family] present in test data point [True]
326 Text feature [regions] present in test data point [True]
361 Text feature [mm] present in test data point [True]
368 Text feature [performed] present in test data point [True]
374 Text feature [key] present in test data point [True]
375 Text feature [significantly] present in test data point [True]
376 Text feature [terminal] present in test data point [True]
378 Text feature [blue] present in test data point [True]
387 Text feature [several] present in test data point [True]
388 Text feature [2a] present in test data point [True]
390 Text feature [recombinant] present in test data point [True]
393 Text feature [formation] present in test data point [True]
396 Text feature [decreased] present in test data point [True]
397 Text feature [cancers] present in test data point [True]
400 Text feature [characterized] present in test data point [True]
```

```
407 Text feature [absence] present in test data point [True]
410 Text feature [reaction] present in test data point [True]
413 Text feature [phase] present in test data point [True]
419 Text feature [information] present in test data point [True]
420 Text feature [co] present in test data point [True]
423 Text feature [200] present in test data point [True]
426 Text feature [complete] present in test data point [True]
440 Text feature [consequences] present in test data point [True]
444 Text feature [figure] present in test data point [True]
446 Text feature [observed] present in test data point [True]
450 Text feature [experiments] present in test data point [True]
454 Text feature [residue] present in test data point [True]
459 Text feature [least] present in test data point [True]
460 Text feature [supplementary] present in test data point [True]
461 Text feature [primary] present in test data point [True]
462 Text feature [affinity] present in test data point [True]
465 Text feature [strand] present in test data point [True]
481 Text feature [length] present in test data point [True]
482 Text feature [1a] present in test data point [True]
485 Text feature [shows] present in test data point [True]
491 Text feature [despite] present in test data point [True]
492 Text feature [form] present in test data point [True]
495 Text feature [respectively] present in test data point [True]
Out of the top  500  features  74 are present in query point
```

4.3.2.4. Feature Importance, Correctly Classified point

In [311]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5267 0.0144 0.0093 0.3562 0.0173 0.0146 0.
0555 0.004  0.0019]]
Actual Class : 4
--------------------------------------------------
17 Text feature [hydrophobic] present in test data point [True]
74 Text feature [encoding] present in test data point [True]
76 Text feature [function] present in test data point [True]
90 Text feature [21] present in test data point [True]
132 Text feature [structure] present in test data point [True]
138 Text feature [indicated] present in test data point [True]
149 Text feature [signal] present in test data point [True]
151 Text feature [affecting] present in test data point [True]
152 Text feature [codon] present in test data point [True]
158 Text feature [reduced] present in test data point [True]
160 Text feature [subjected] present in test data point [True]
162 Text feature [therefore] present in test data point [True]
163 Text feature [05] present in test data point [True]
171 Text feature [loss] present in test data point [True]
173 Text feature [coding] present in test data point [True]
175 Text feature [signals] present in test data point [True]
182 Text feature [page] present in test data point [True]
186 Text feature [next] present in test data point [True]
190 Text feature [region] present in test data point [True]
193 Text feature [mean] present in test data point [True]
196 Text feature [less] present in test data point [True]
204 Text feature [2001] present in test data point [True]
206 Text feature [pa] present in test data point [True]
207 Text feature [showing] present in test data point [True]
209 Text feature [role] present in test data point [True]
210 Text feature [population] present in test data point [True]
218 Text feature [assessment] present in test data point [True]
219 Text feature [assess] present in test data point [True]
220 Text feature [2006] present in test data point [True]
226 Text feature [directed] present in test data point [True]
237 Text feature [constructs] present in test data point [True]
242 Text feature [even] present in test data point [True]
243 Text feature [screening] present in test data point [True]
248 Text feature [binding] present in test data point [True]
253 Text feature [another] present in test data point [True]
257 Text feature [value] present in test data point [True]
258 Text feature [one] present in test data point [True]
260 Text feature [previous] present in test data point [True]
262 Text feature [s1] present in test data point [True]
265 Text feature [www] present in test data point [True]
270 Text feature [change] present in test data point [True]
271 Text feature [analyzed] present in test data point [True]
274 Text feature [affect] present in test data point [True]
276 Text feature [essential] present in test data point [True]
279 Text feature [remaining] present in test data point [True]
281 Text feature [tested] present in test data point [True]
282 Text feature [nucleotide] present in test data point [True]
283 Text feature [antibodies] present in test data point [True]
285 Text feature [seven] present in test data point [True]
287 Text feature [construct] present in test data point [True]
295 Text feature [resulted] present in test data point [True]
296 Text feature [19] present in test data point [True]
```

```
300 Text feature [medium] present in test data point [True]
313 Text feature [ratio] present in test data point [True]
316 Text feature [2008] present in test data point [True]
319 Text feature [reporter] present in test data point [True]
320 Text feature [assessed] present in test data point [True]
325 Text feature [effect] present in test data point [True]
332 Text feature [provide] present in test data point [True]
333 Text feature [criteria] present in test data point [True]
337 Text feature [based] present in test data point [True]
344 Text feature [diagnosis] present in test data point [True]
345 Text feature [protein] present in test data point [True]
346 Text feature [sds] present in test data point [True]
347 Text feature [carrying] present in test data point [True]
362 Text feature [evidence] present in test data point [True]
370 Text feature [pathogenic] present in test data point [True]
372 Text feature [possible] present in test data point [True]
378 Text feature [within] present in test data point [True]
385 Text feature [small] present in test data point [True]
388 Text feature [vivo] present in test data point [True]
391 Text feature [http] present in test data point [True]
393 Text feature [gene] present in test data point [True]
394 Text feature [mutagenesis] present in test data point [True]
395 Text feature [strong] present in test data point [True]
397 Text feature [frame] present in test data point [True]
400 Text feature [furthermore] present in test data point [True]
402 Text feature [selected] present in test data point [True]
404 Text feature [mutation] present in test data point [True]
405 Text feature [relative] present in test data point [True]
407 Text feature [17] present in test data point [True]
409 Text feature [line] present in test data point [True]
412 Text feature [limited] present in test data point [True]
415 Text feature [wild] present in test data point [True]
416 Text feature [type] present in test data point [True]
420 Text feature [across] present in test data point [True]
421 Text feature [vector] present in test data point [True]
424 Text feature [individual] present in test data point [True]
426 Text feature [significantly] present in test data point [True]
427 Text feature [corresponding] present in test data point [True]
432 Text feature [added] present in test data point [True]
438 Text feature [whether] present in test data point [True]
439 Text feature [2002] present in test data point [True]
441 Text feature [cell] present in test data point [True]
442 Text feature [least] present in test data point [True]
443 Text feature [assays] present in test data point [True]
444 Text feature [sequence] present in test data point [True]
447 Text feature [1b] present in test data point [True]
449 Text feature [three] present in test data point [True]
451 Text feature [derived] present in test data point [True]
452 Text feature [splicing] present in test data point [True]
454 Text feature [incubated] present in test data point [True]
458 Text feature [cause] present in test data point [True]
461 Text feature [either] present in test data point [True]
462 Text feature [following] present in test data point [True]
464 Text feature [presence] present in test data point [True]
469 Text feature [genetic] present in test data point [True]
471 Text feature [would] present in test data point [True]
472 Text feature [total] present in test data point [True]
```

```
473 Text feature [37] present in test data point [True]
475 Text feature [localization] present in test data point [True]
479 Text feature [consistent] present in test data point [True]
481 Text feature [red] present in test data point [True]
483 Text feature [clones] present in test data point [True]
486 Text feature [39] present in test data point [True]
496 Text feature [co] present in test data point [True]
Out of the top  500  features  116 are present in query point
```

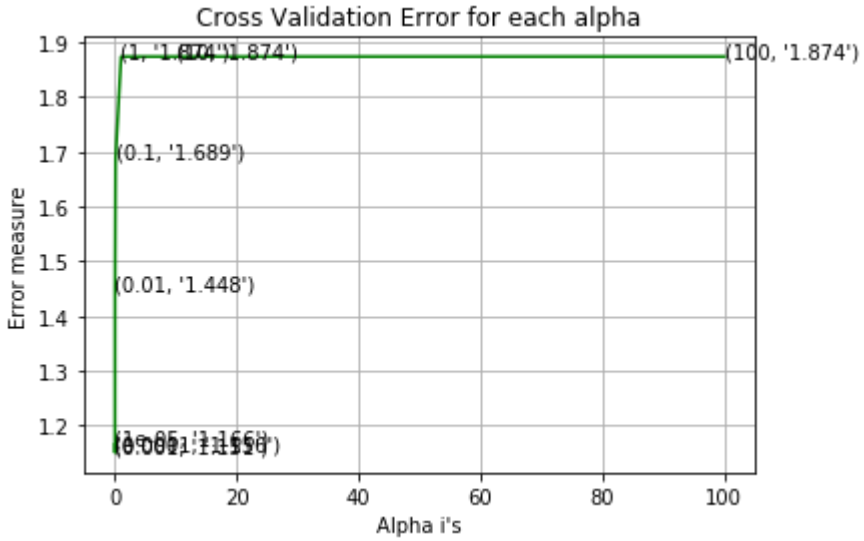
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [312]:
```
# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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



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

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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.165769673386477
for C = 0.0001
Log Loss : 1.1555618007878556
for C = 0.001
Log Loss : 1.1507249602637286
for C = 0.01
Log Loss : 1.44766894522108
for C = 0.1
Log Loss : 1.6885121492313717
for C = 1
Log Loss : 1.873766715022837
for C = 10
Log Loss : 1.873766534905684
for C = 100
Log Loss : 1.8737666748782762
```



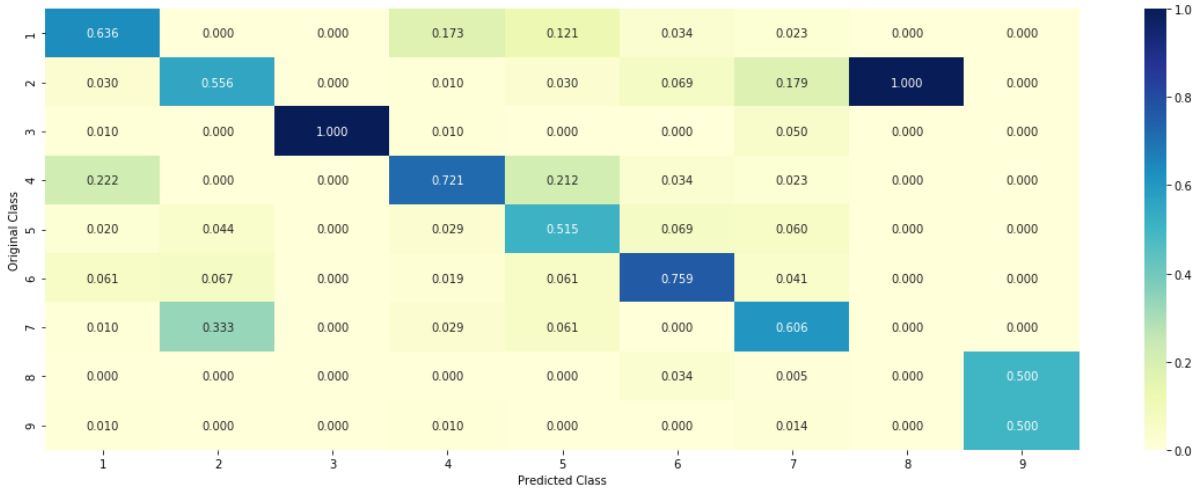

```
For values of best alpha =  0.001 The train log loss is: 0.7741107260129452
For values of best alpha =  0.001 The cross validation log loss is: 1.1507249
602637286
For values of best alpha =  0.001 The test log loss is: 1.1541738960684906
```

4.4.2. Testing model with best hyper parameters

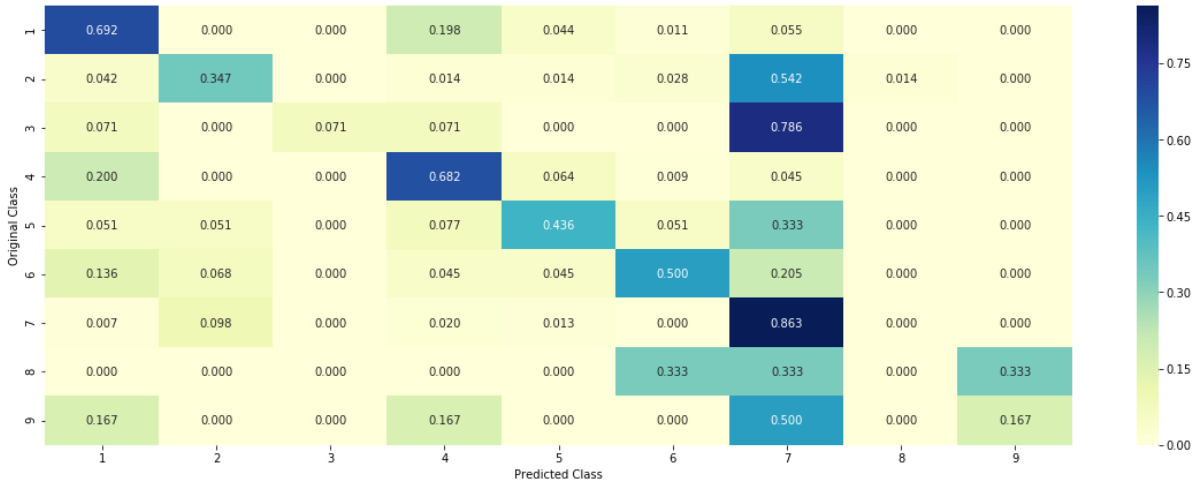
In [313]:
```
# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 1.1507249602637286
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```


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




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



4.3.3. Feature Importance

4.3.3.1. For InCorrectly classified point

In [314]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1787 0.0311 0.013  0.6536 0.0166 0.0741 0.
023  0.0035 0.0064]]
Actual Class : 4
--------------------------------------------------
30 Text feature [show] present in test data point [True]
137 Text feature [recognition] present in test data point [True]
138 Text feature [unable] present in test data point [True]
167 Text feature [see] present in test data point [True]
175 Text feature [inactivation] present in test data point [True]
183 Text feature [catalytic] present in test data point [True]
190 Text feature [nature] present in test data point [True]
202 Text feature [purified] present in test data point [True]
206 Text feature [substrate] present in test data point [True]
212 Text feature [high] present in test data point [True]
217 Text feature [rate] present in test data point [True]
221 Text feature [4a] present in test data point [True]
226 Text feature [differences] present in test data point [True]
229 Text feature [reaction] present in test data point [True]
235 Text feature [yeast] present in test data point [True]
239 Text feature [2a] present in test data point [True]
245 Text feature [indicate] present in test data point [True]
246 Text feature [particular] present in test data point [True]
247 Text feature [comparison] present in test data point [True]
248 Text feature [shows] present in test data point [True]
249 Text feature [deletion] present in test data point [True]
251 Text feature [3b] present in test data point [True]
262 Text feature [mm] present in test data point [True]
266 Text feature [1b] present in test data point [True]
277 Text feature [require] present in test data point [True]
286 Text feature [complex] present in test data point [True]
287 Text feature [representative] present in test data point [True]
288 Text feature [direct] present in test data point [True]
290 Text feature [bound] present in test data point [True]
298 Text feature [strand] present in test data point [True]
302 Text feature [co] present in test data point [True]
307 Text feature [29] present in test data point [True]
308 Text feature [200] present in test data point [True]
309 Text feature [led] present in test data point [True]
312 Text feature [cannot] present in test data point [True]
313 Text feature [motif] present in test data point [True]
315 Text feature [described] present in test data point [True]
318 Text feature [fact] present in test data point [True]
324 Text feature [mutants] present in test data point [True]
329 Text feature [regions] present in test data point [True]
335 Text feature [recent] present in test data point [True]
336 Text feature [view] present in test data point [True]
342 Text feature [contribute] present in test data point [True]
356 Text feature [several] present in test data point [True]
360 Text feature [partial] present in test data point [True]
366 Text feature [specifically] present in test data point [True]
367 Text feature [mutated] present in test data point [True]
369 Text feature [protein] present in test data point [True]
371 Text feature [experiments] present in test data point [True]
374 Text feature [standard] present in test data point [True]
375 Text feature [consequences] present in test data point [True]
376 Text feature [larger] present in test data point [True]
```

```
380 Text feature [despite] present in test data point [True]
381 Text feature [phase] present in test data point [True]
382 Text feature [inhibitory] present in test data point [True]
385 Text feature [ref] present in test data point [True]
395 Text feature [characterized] present in test data point [True]
396 Text feature [15] present in test data point [True]
398 Text feature [negative] present in test data point [True]
399 Text feature [absence] present in test data point [True]
401 Text feature [family] present in test data point [True]
403 Text feature [1a] present in test data point [True]
405 Text feature [observed] present in test data point [True]
406 Text feature [cancers] present in test data point [True]
410 Text feature [respectively] present in test data point [True]
415 Text feature [figure] present in test data point [True]
416 Text feature [performed] present in test data point [True]
418 Text feature [multiple] present in test data point [True]
419 Text feature [key] present in test data point [True]
430 Text feature [blue] present in test data point [True]
432 Text feature [likely] present in test data point [True]
439 Text feature [residue] present in test data point [True]
440 Text feature [primary] present in test data point [True]
454 Text feature [presence] present in test data point [True]
455 Text feature [considered] present in test data point [True]
459 Text feature [overall] present in test data point [True]
461 Text feature [involved] present in test data point [True]
468 Text feature [directly] present in test data point [True]
469 Text feature [disease] present in test data point [True]
471 Text feature [23] present in test data point [True]
475 Text feature [product] present in test data point [True]
476 Text feature [complete] present in test data point [True]
478 Text feature [ph] present in test data point [True]
482 Text feature [suggested] present in test data point [True]
488 Text feature [12] present in test data point [True]
490 Text feature [core] present in test data point [True]
494 Text feature [indeed] present in test data point [True]
495 Text feature [whereas] present in test data point [True]
499 Text feature [bind] present in test data point [True]
Out of the top  500  features  89 are present in query point
```

4.3.3.2. For correctly classified point

In [315]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2987 0.0977 0.0363 0.2477 0.0628 0.0527 0.
19   0.0084 0.0057]]
Actual Class : 4
--------------------------------------------------
15 Text feature [hydrophobic] present in test data point [True]
23 Text feature [2006] present in test data point [True]
164 Text feature [encoding] present in test data point [True]
165 Text feature [signals] present in test data point [True]
171 Text feature [signal] present in test data point [True]
172 Text feature [affecting] present in test data point [True]
177 Text feature [next] present in test data point [True]
178 Text feature [subjected] present in test data point [True]
180 Text feature [showing] present in test data point [True]
181 Text feature [reduced] present in test data point [True]
184 Text feature [2001] present in test data point [True]
187 Text feature [function] present in test data point [True]
194 Text feature [therefore] present in test data point [True]
197 Text feature [role] present in test data point [True]
200 Text feature [binding] present in test data point [True]
203 Text feature [2008] present in test data point [True]
204 Text feature [essential] present in test data point [True]
205 Text feature [21] present in test data point [True]
212 Text feature [structure] present in test data point [True]
213 Text feature [directed] present in test data point [True]
214 Text feature [indicated] present in test data point [True]
215 Text feature [www] present in test data point [True]
217 Text feature [derived] present in test data point [True]
222 Text feature [even] present in test data point [True]
224 Text feature [less] present in test data point [True]
226 Text feature [value] present in test data point [True]
231 Text feature [mean] present in test data point [True]
232 Text feature [one] present in test data point [True]
233 Text feature [loss] present in test data point [True]
242 Text feature [19] present in test data point [True]
244 Text feature [gene] present in test data point [True]
247 Text feature [cell] present in test data point [True]
248 Text feature [screening] present in test data point [True]
249 Text feature [s1] present in test data point [True]
250 Text feature [constructs] present in test data point [True]
254 Text feature [human] present in test data point [True]
255 Text feature [mutagenesis] present in test data point [True]
264 Text feature [across] present in test data point [True]
266 Text feature [construct] present in test data point [True]
268 Text feature [nucleotide] present in test data point [True]
269 Text feature [sds] present in test data point [True]
277 Text feature [individual] present in test data point [True]
278 Text feature [33] present in test data point [True]
279 Text feature [05] present in test data point [True]
281 Text feature [identified] present in test data point [True]
285 Text feature [2002] present in test data point [True]
286 Text feature [population] present in test data point [True]
287 Text feature [indicates] present in test data point [True]
288 Text feature [vivo] present in test data point [True]
290 Text feature [coding] present in test data point [True]
291 Text feature [2005] present in test data point [True]
296 Text feature [added] present in test data point [True]
```

```
300 Text feature [significantly] present in test data point [True]
302 Text feature [whether] present in test data point [True]
303 Text feature [codon] present in test data point [True]
304 Text feature [al] present in test data point [True]
306 Text feature [protein] present in test data point [True]
308 Text feature [another] present in test data point [True]
309 Text feature [assessed] present in test data point [True]
314 Text feature [assess] present in test data point [True]
316 Text feature [previous] present in test data point [True]
319 Text feature [et] present in test data point [True]
323 Text feature [17] present in test data point [True]
325 Text feature [page] present in test data point [True]
326 Text feature [tested] present in test data point [True]
329 Text feature [1999] present in test data point [True]
330 Text feature [required] present in test data point [True]
331 Text feature [antibodies] present in test data point [True]
333 Text feature [splicing] present in test data point [True]
334 Text feature [evidence] present in test data point [True]
336 Text feature [region] present in test data point [True]
341 Text feature [reporter] present in test data point [True]
344 Text feature [incubated] present in test data point [True]
345 Text feature [pathogenic] present in test data point [True]
346 Text feature [49] present in test data point [True]
347 Text feature [affect] present in test data point [True]
351 Text feature [criteria] present in test data point [True]
356 Text feature [inhibit] present in test data point [True]
358 Text feature [analyzed] present in test data point [True]
368 Text feature [assessment] present in test data point [True]
370 Text feature [expressed] present in test data point [True]
371 Text feature [2004] present in test data point [True]
376 Text feature [diagnosis] present in test data point [True]
377 Text feature [total] present in test data point [True]
380 Text feature [red] present in test data point [True]
382 Text feature [1997] present in test data point [True]
386 Text feature [sequence] present in test data point [True]
389 Text feature [independent] present in test data point [True]
390 Text feature [within] present in test data point [True]
391 Text feature [mutation] present in test data point [True]
396 Text feature [remaining] present in test data point [True]
404 Text feature [small] present in test data point [True]
408 Text feature [well] present in test data point [True]
410 Text feature [three] present in test data point [True]
413 Text feature [exon] present in test data point [True]
415 Text feature [line] present in test data point [True]
429 Text feature [fig] present in test data point [True]
432 Text feature [effects] present in test data point [True]
435 Text feature [acids] present in test data point [True]
436 Text feature [furthermore] present in test data point [True]
442 Text feature [http] present in test data point [True]
443 Text feature [specific] present in test data point [True]
445 Text feature [2012] present in test data point [True]
453 Text feature [27] present in test data point [True]
456 Text feature [involving] present in test data point [True]
460 Text feature [splice] present in test data point [True]
461 Text feature [localization] present in test data point [True]
462 Text feature [medium] present in test data point [True]
463 Text feature [24] present in test data point [True]
```

```
464 Text feature [values] present in test data point [True]
465 Text feature [carrying] present in test data point [True]
467 Text feature [provide] present in test data point [True]
468 Text feature [28] present in test data point [True]
473 Text feature [48] present in test data point [True]
474 Text feature [average] present in test data point [True]
477 Text feature [possible] present in test data point [True]
478 Text feature [different] present in test data point [True]
480 Text feature [addition] present in test data point [True]
481 Text feature [tsc2] present in test data point [True]
486 Text feature [based] present in test data point [True]
487 Text feature [overall] present in test data point [True]
488 Text feature [determined] present in test data point [True]
Out of the top  500  features  122 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [316]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

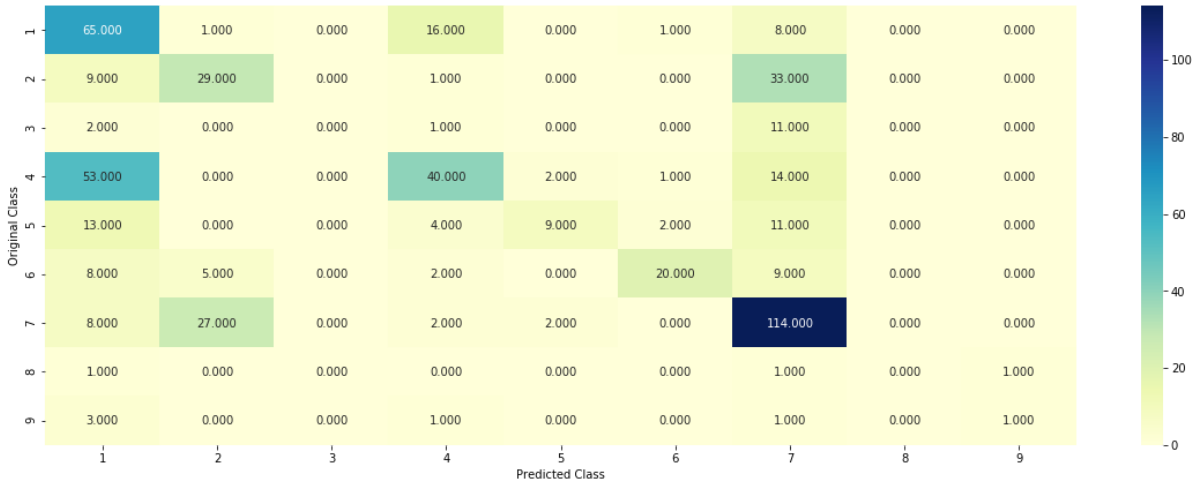
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],c
v_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2736833060564123
for n_estimators = 100 and max depth =  10
Log Loss : 1.2866238127242315
for n_estimators = 200 and max depth =  5
Log Loss : 1.2592700584628636
for n_estimators = 200 and max depth =  10
Log Loss : 1.2821426493553543
for n_estimators = 500 and max depth =  5
Log Loss : 1.254971842261894
for n_estimators = 500 and max depth =  10
Log Loss : 1.2749355432584017
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2541087998919056
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2708825529942867
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2530043674412312
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2713692573094837
For values of best estimator =  2000 The train log loss is: 0.837379315326220
1
For values of best estimator =  2000 The cross validation log loss is: 1.2530
043674412312
For values of best estimator =  2000 The test log loss is: 1.2287422809963742
```

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

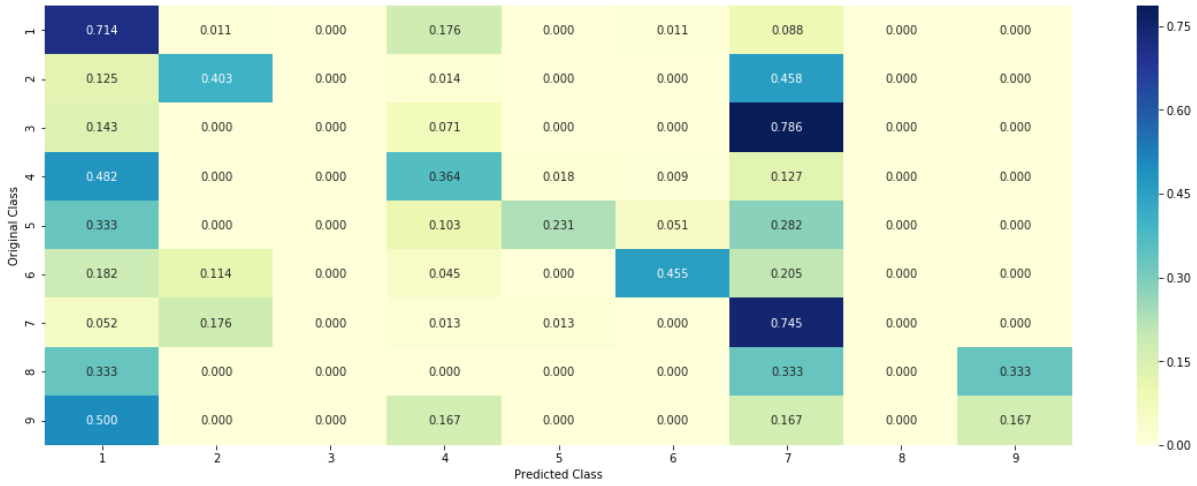
In [317]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.2530043674412312
Number of mis-classified points : 0.4774436090225564
-------------------- 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 [318]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion=
'gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 2
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2666 0.0639 0.0185 0.4108 0.0728 0.0855 0.
0551 0.0098 0.017 ]]
Actual Class : 4
--------------------------------------------------
4 Text feature [function] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
19 Text feature [treated] present in test data point [True]
22 Text feature [expression] present in test data point [True]
23 Text feature [protein] present in test data point [True]
28 Text feature [neutral] present in test data point [True]
40 Text feature [therapeutic] present in test data point [True]
42 Text feature [yeast] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
62 Text feature [resistance] present in test data point [True]
66 Text feature [drug] present in test data point [True]
70 Text feature [predicted] present in test data point [True]
72 Text feature [downstream] present in test data point [True]
74 Text feature [inactivation] present in test data point [True]
84 Text feature [sensitive] present in test data point [True]
86 Text feature [assays] present in test data point [True]
94 Text feature [information] present in test data point [True]
99 Text feature [sequence] present in test data point [True]
Out of the top  100  features  19 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [319]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4615 0.009  0.0121 0.0884 0.2802 0.1297 0.
0144 0.0035 0.0012]]
Actuall Class : 1
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
4 Text feature [function] present in test data point [True]
6 Text feature [activation] present in test data point [True]
7 Text feature [suppressor] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
10 Text feature [loss] present in test data point [True]
11 Text feature [missense] present in test data point [True]
13 Text feature [brca1] present in test data point [True]
14 Text feature [constitutive] present in test data point [True]
16 Text feature [deleterious] present in test data point [True]
17 Text feature [variants] present in test data point [True]
22 Text feature [expression] present in test data point [True]
23 Text feature [protein] present in test data point [True]
25 Text feature [pathogenic] present in test data point [True]
26 Text feature [stability] present in test data point [True]
27 Text feature [functional] present in test data point [True]
28 Text feature [neutral] present in test data point [True]
30 Text feature [cells] present in test data point [True]
34 Text feature [57] present in test data point [True]
38 Text feature [cell] present in test data point [True]
39 Text feature [classified] present in test data point [True]
42 Text feature [yeast] present in test data point [True]
43 Text feature [patients] present in test data point [True]
44 Text feature [signaling] present in test data point [True]
45 Text feature [functions] present in test data point [True]
49 Text feature [brca2] present in test data point [True]
51 Text feature [defective] present in test data point [True]
53 Text feature [clinical] present in test data point [True]
58 Text feature [repair] present in test data point [True]
59 Text feature [ring] present in test data point [True]
60 Text feature [activate] present in test data point [True]
61 Text feature [ligand] present in test data point [True]
63 Text feature [proteins] present in test data point [True]
65 Text feature [brct] present in test data point [True]
68 Text feature [ovarian] present in test data point [True]
70 Text feature [predicted] present in test data point [True]
71 Text feature [use] present in test data point [True]
72 Text feature [downstream] present in test data point [True]
76 Text feature [nuclear] present in test data point [True]
78 Text feature [expected] present in test data point [True]
82 Text feature [variant] present in test data point [True]
83 Text feature [affected] present in test data point [True]
84 Text feature [sensitive] present in test data point [True]
85 Text feature [breast] present in test data point [True]
86 Text feature [assays] present in test data point [True]
88 Text feature [response] present in test data point [True]
90 Text feature [history] present in test data point [True]
93 Text feature [mammalian] present in test data point [True]
94 Text feature [information] present in test data point [True]
95 Text feature [genes] present in test data point [True]
96 Text feature [dna] present in test data point [True]
```

```
97 Text feature [bard1] present in test data point [True]
99 Text feature [sequence] present in test data point [True]
Out of the top  100  features  54 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [320]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


# 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))
```

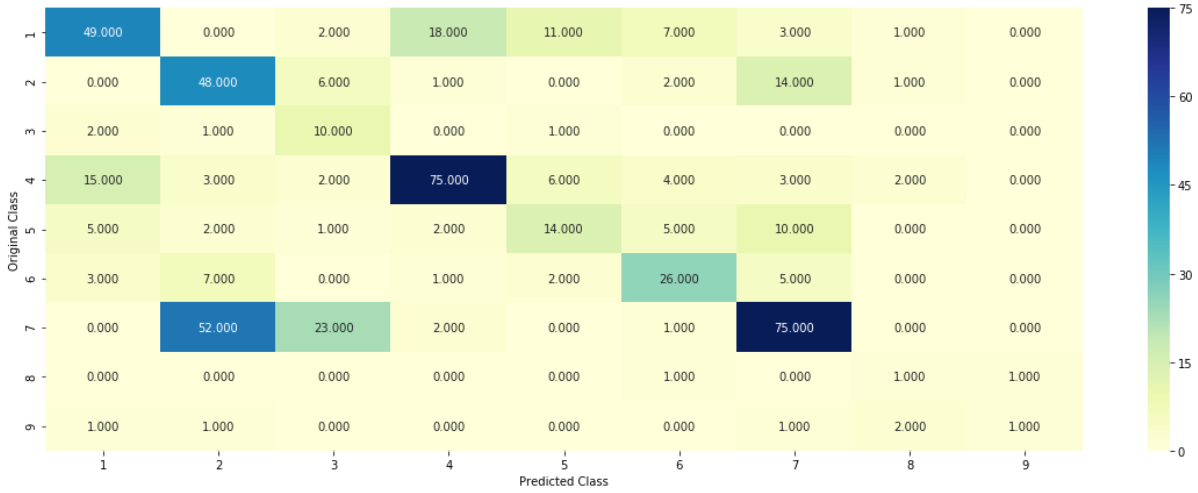
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log
 loss is:",log_loss(train_y, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross vali
dation log loss is:",log_loss(cv_y, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log l
oss is:",log_loss(test_y, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.079365058178695
for n_estimators = 10 and max depth =  3
Log Loss : 1.8462568629847556
for n_estimators = 10 and max depth =  5
Log Loss : 1.6096718798690968
for n_estimators = 10 and max depth =  10
Log Loss : 1.7633623284047186
for n_estimators = 50 and max depth =  2
Log Loss : 1.7622098557578474
for n_estimators = 50 and max depth =  3
Log Loss : 1.4779538115449786
for n_estimators = 50 and max depth =  5
Log Loss : 1.4371067464371061
for n_estimators = 50 and max depth =  10
Log Loss : 1.602248688059107
for n_estimators = 100 and max depth =  2
Log Loss : 1.651866558136504
for n_estimators = 100 and max depth =  3
Log Loss : 1.4815840784491068
for n_estimators = 100 and max depth =  5
Log Loss : 1.3487282040932687
for n_estimators = 100 and max depth =  10
Log Loss : 1.6375837824221184
for n_estimators = 200 and max depth =  2
Log Loss : 1.677717997094418
for n_estimators = 200 and max depth =  3
Log Loss : 1.512448759049706
for n_estimators = 200 and max depth =  5
Log Loss : 1.3675126372326867
for n_estimators = 200 and max depth =  10
Log Loss : 1.630680588040141
for n_estimators = 500 and max depth =  2
Log Loss : 1.6747384716800862
for n_estimators = 500 and max depth =  3
Log Loss : 1.5379303963159887
for n_estimators = 500 and max depth =  5
Log Loss : 1.3884965770944975
for n_estimators = 500 and max depth =  10
Log Loss : 1.661659826257498
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6825821016824942
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5668780450451543
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3567924320714277
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6604394899738006
For values of best alpha =  100 The train log loss is: 0.058311217812717565
For values of best alpha =  100 The cross validation log loss is: 1.348728204
0932687
For values of best alpha =  100 The test log loss is: 1.3145360187535424
```

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

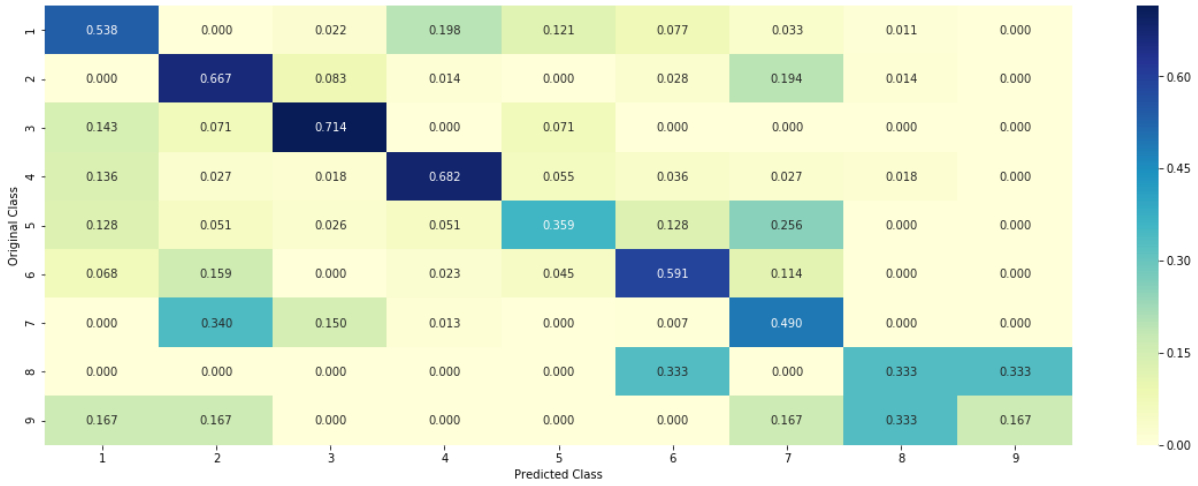
In [321]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.348728204093269
Number of mis-classified points : 0.43796992481203006
-------------------- 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 [322]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion=
'gini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.036  0.0049 0.0735 0.0357 0.6898 0.1486 0.
0033 0.0042 0.004 ]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text 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
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [323]:
```
test_point_index = 2
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1246 0.0395 0.146  0.5226 0.0199 0.0673 0.
0176 0.0428 0.0195]]
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
Gene is important feature
Text is important feature
Text 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
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [324]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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


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

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

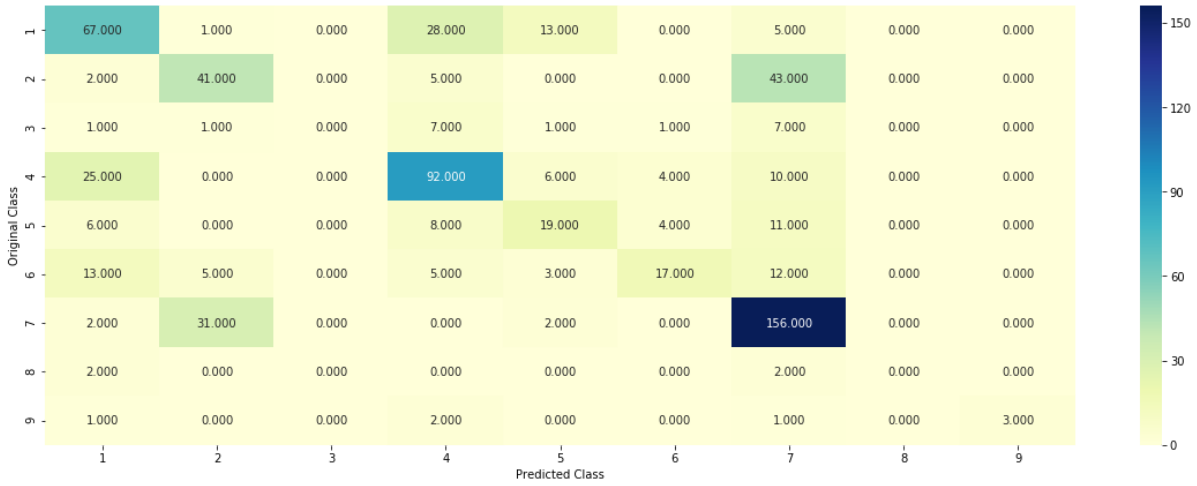
```
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


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

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


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

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

```
Logistic Regression :  Log Loss: 1.10
Support vector machines : Log Loss: 1.87
Naive Bayes : Log Loss: 1.28
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.539
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.223
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.290
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.432
```

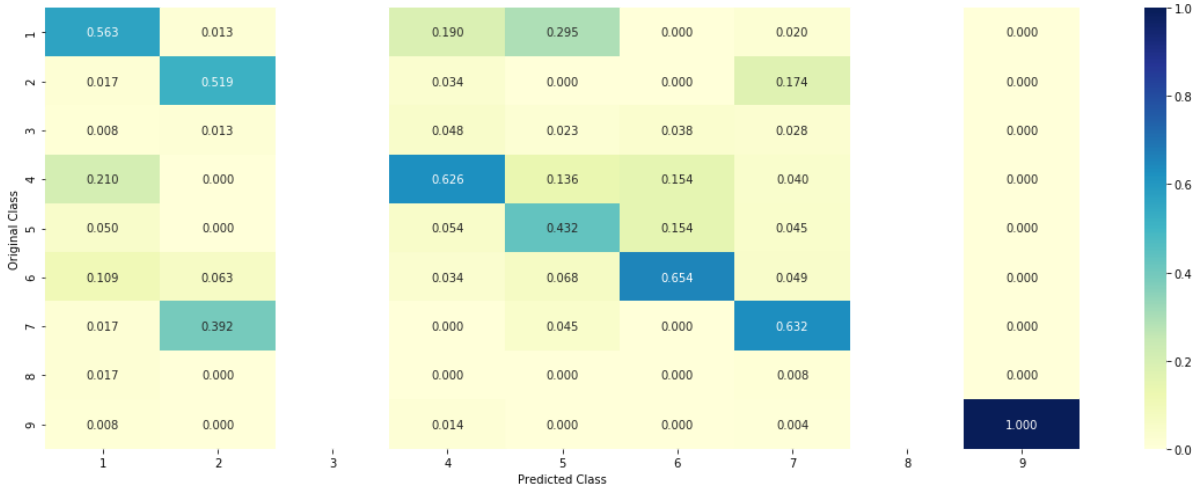
4.7.2 testing the model with the best hyper parameters

In [325]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

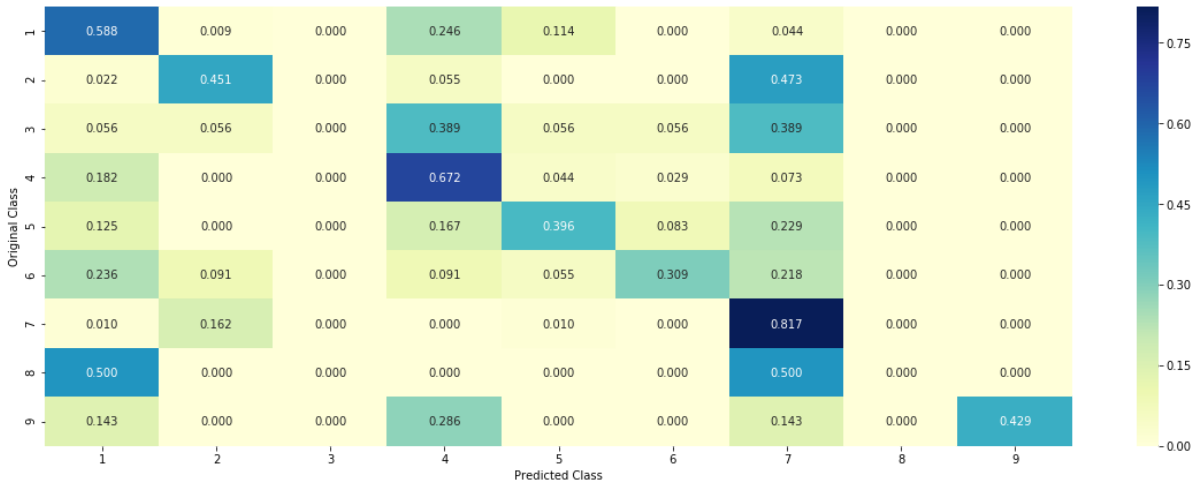
```
Log loss (train) on the stacking classifier : 0.788983916559872
Log loss (CV) on the stacking classifier : 1.2233851026771143
Log loss (test) on the stacking classifier : 1.1894744759046902
Number of missclassified point : 0.40601503759398494
-------------------- Confusion matrix --------------------
```




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



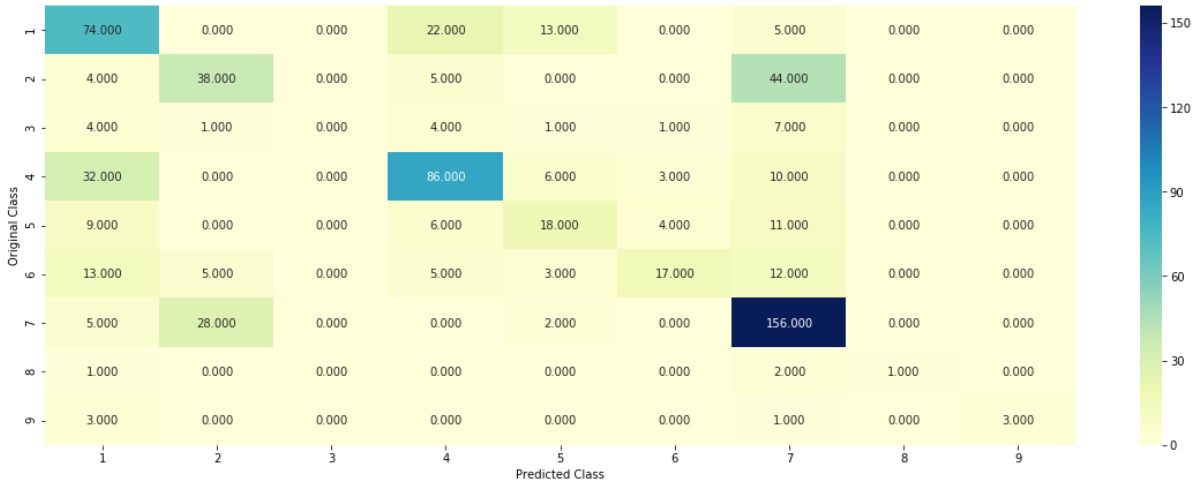

```
-------------------- Recall matrix (Row sum=1) --------------------
```



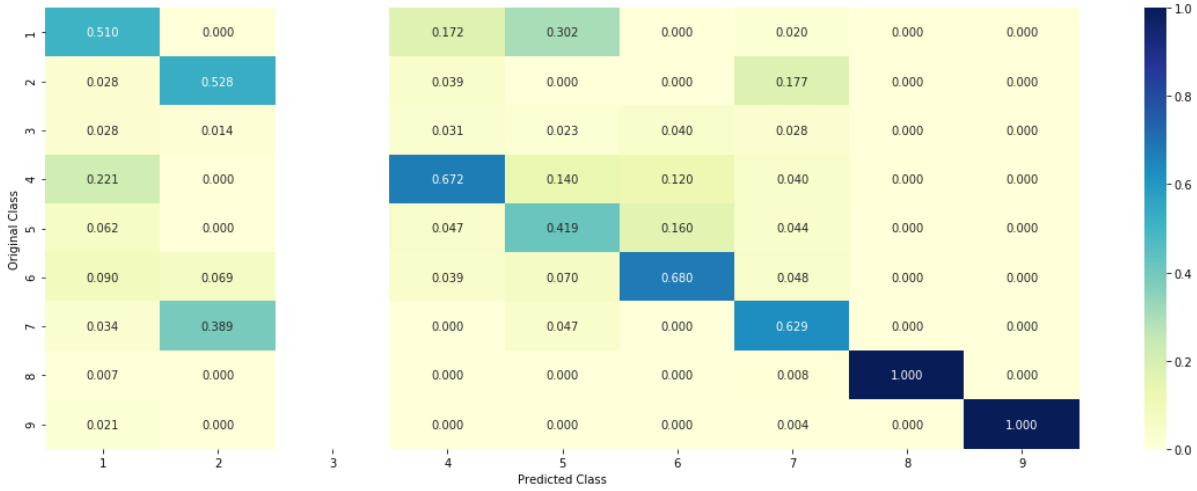
4.7.3 Maximum Voting classifier

In [326]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

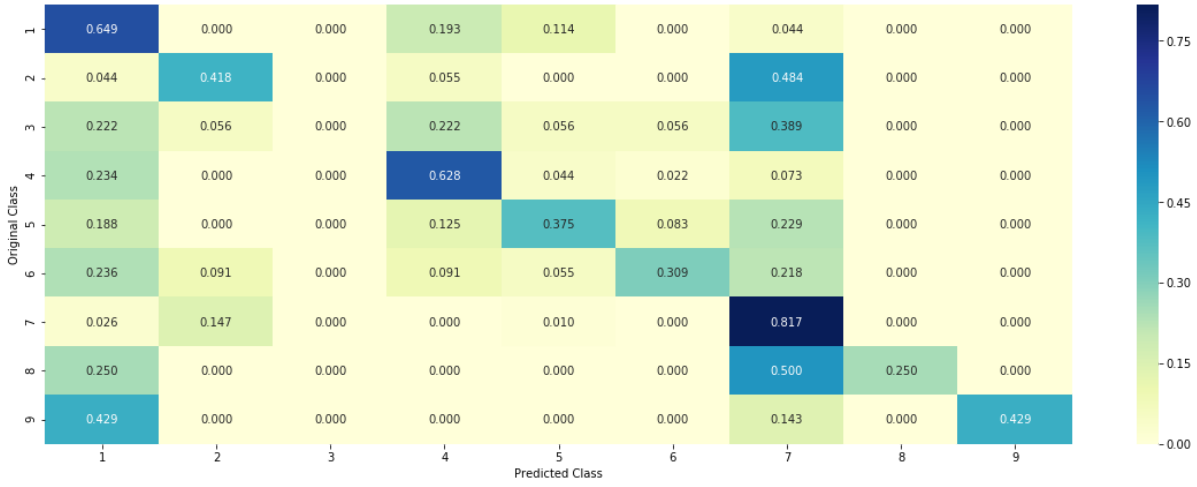
```
Log loss (train) on the VotingClassifier : 0.928872836710069
Log loss (CV) on the VotingClassifier : 1.26266597039127
Log loss (test) on the VotingClassifier : 1.2180427822758082
Number of missclassified point : 0.40902255639097745
-------------------- Confusion matrix --------------------
```




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




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



5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [327]:
```
print(train_df.shape, y_train.shape)
print(cv_df.shape, y_cv.shape)
print(test_df.shape, y_test.shape)
```

```
(2124, 5) (2124,)
(532, 5) (532,)
(665, 5) (665,)
```

In [328]:
```
train_df.head()
```

Out[328]:

	ID	Gene	Variation	Class	TEXT
1076	1076	FOXA1	F400I	1	characterization prostate cancer transcriptome...
283	283	NKX2-1	TRA-NKX2-1_Fusion	2	40 pediatric cases underlying oncogenic rearra...
2125	2125	CCND1	T286I	7	activities cyclin dependent kinases serve inte...
962	962	ESR1	Fusions	2	crucial role recurrent gene fusions developmen...
1278	1278	HRAS	G12V	7	three dimensional structure complex human h ra...

In [537]:
```
# Feature : Gene

#Apply Count Vectorizer that includes both unigram and bigram
count_vect = CountVectorizer(ngram_range=(1, 2))
train_gene_feature_onehotCoding = count_vect.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = count_vect.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = count_vect.transform(cv_df['Gene'])

# Normalize the gene feature vectors
# train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCoding,
axis = 0)
# test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCoding, a
xis = 0)
# cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding, axis
 = 0)

# Feature : Variation
train_variation_feature_onehotCoding = count_vect.fit_transform(train_df['Vari
ation'])
test_variation_feature_onehotCoding = count_vect.transform(test_df['Variation'
])
cv_variation_feature_onehotCoding = count_vect.transform(cv_df['Variation'])

# normalize the Variation feature vectors
# train_variation_feature_onehotCoding = normalize(train_variation_feature_one
hotCoding, axis = 0)
# test_variation_feature_onehotCoding = normalize(test_variation_feature_oneho
tCoding, axis = 0)
# cv_variation_feature_onehotCoding = normalize(cv_variation_feature_onehotCod
ing, axis = 0)
```

In [538]:
```
# Feature : TEXT
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data

count_vect = CountVectorizer(ngram_range=(1, 2), min_df=3)
train_text_feature_onehotCoding = count_vect.fit_transform(train_df['TEXT'])
# train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding,
axis=0)
test_text_feature_onehotCoding = count_vect.transform(test_df['TEXT'])
# test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, a
xis=0)
cv_text_feature_onehotCoding = count_vect.transform(cv_df['TEXT'])
# cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=
0)
```

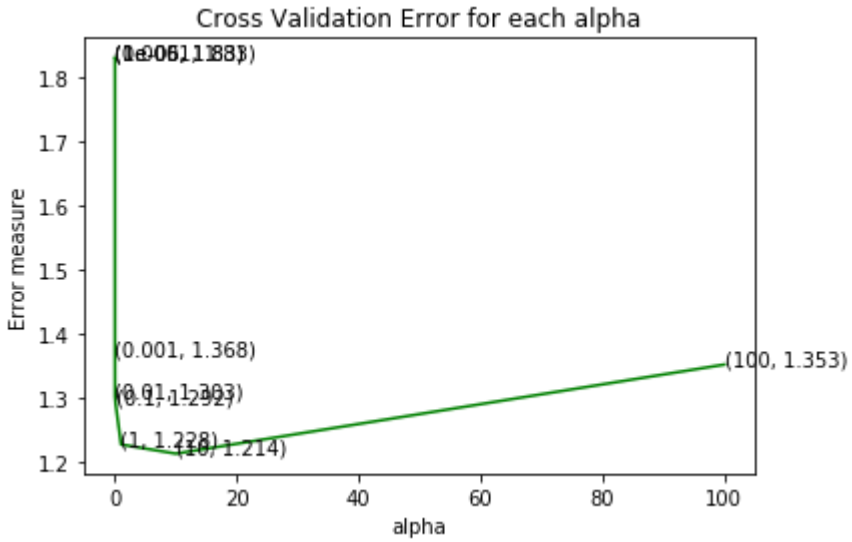
In [540]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
y_train = train_df['Class']

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
y_test = test_df['Class']

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
y_cv = cv_df['Class']
```

In [541]:
```
print(train_x_onehotCoding.shape, y_train.shape)
print(cv_x_onehotCoding.shape, y_cv.shape)
print(test_x_onehotCoding.shape, y_test.shape)
```

```
(2124, 773730) (2124,)
(532, 773730) (532,)
(665, 773730) (665,)
```

In [542]:
```
# Train the model using Logistics Regression and Calibration model

alpha = [10**x for x in range(-6,3)]

cv_log_loss_values = []
for i in alpha:
    clf = SGDClassifier(class_weight = "balanced",alpha = i, loss = 'log', pen
alty = 'l2', random_state = 42)
    clf.fit(train_x_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf,method = 'sigmoid')
    sig_clf.fit(train_x_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
#     log_loss = log_loss(y_cv, predict_y,labels=clf.classes_, eps=1e-15)
    cv_log_loss_values.append(log_loss(y_cv, predict_y,labels=clf.classes_, ep
s=1e-15))
    print("for alpha = ", i, "The log loss is ", log_loss(y_cv, predict_y,labe
ls=clf.classes_, eps=1e-15))

plt.plot(alpha, cv_log_loss_values, c = 'g')
for i, txt in enumerate(np.round(cv_log_loss_values,3)):
    plt.annotate((alpha[i],txt),(alpha[i],cv_log_loss_values[i]))

plt.title('Cross Validation Error for each alpha')
plt.xlabel('alpha')
plt.ylabel('Error measure')
plt.show()

optimal_alpha = alpha[np.argmin(cv_log_loss_values)]

clf = SGDClassifier(class_weight = "balanced",alpha = optimal_alpha, loss = 'l
og', penalty = 'l2', random_state = 42)
clf.fit(train_x_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(method = 'sigmoid')
sig_clf.fit(train_x_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The train log loss is ", log_l
oss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The CV log loss is ",log_loss(
y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The test log loss is ", log_lo
ss(y_test, predict_y))
```

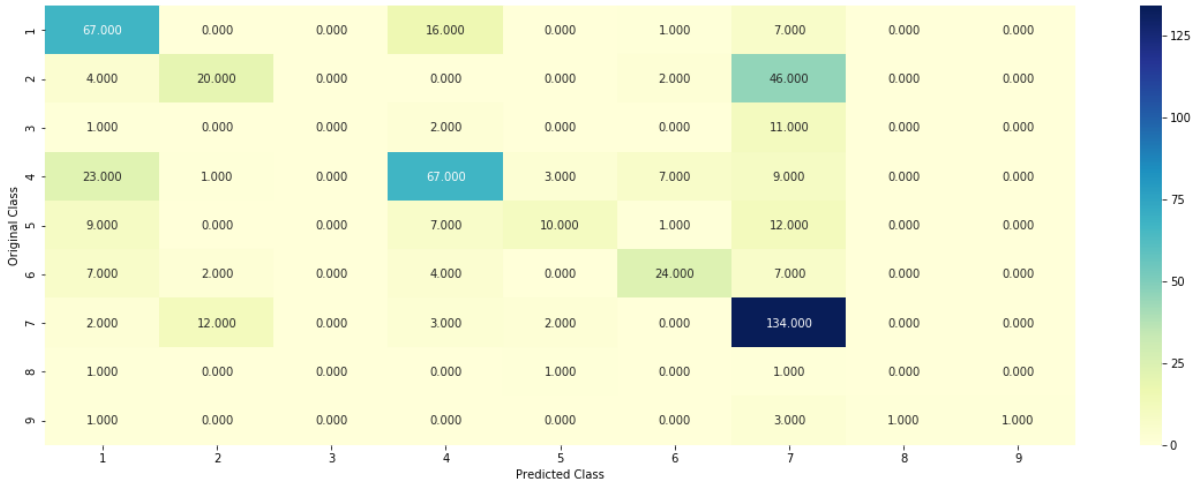
```
for alpha =  1e-06 The log loss is  1.8304997567764278
for alpha =  1e-05 The log loss is  1.8304997567764278
for alpha =  0.0001 The log loss is  1.8304997567764278
for alpha =  0.001 The log loss is  1.3675245788047465
for alpha =  0.01 The log loss is  1.3029895662623745
for alpha =  0.1 The log loss is  1.2921713185245127
for alpha =  1 The log loss is  1.2282032632077804
for alpha =  10 The log loss is  1.2139672769072973
for alpha =  100 The log loss is  1.3526205023298254
```



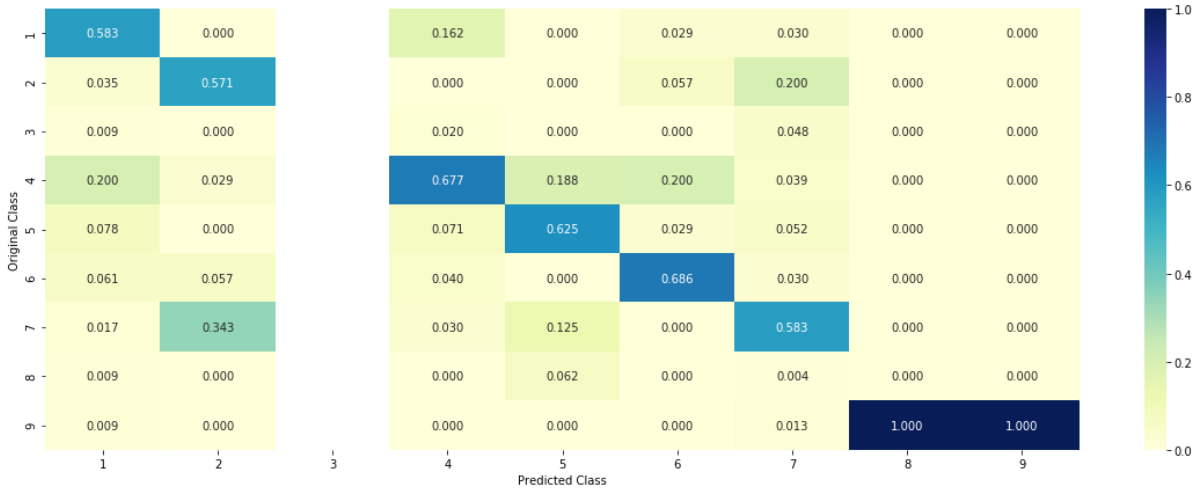

```
For the value alpha :  10 The train log loss is  1.0818267506696408
For the value alpha :  10 The CV log loss is  1.380650468959619
For the value alpha :  10 The test log loss is  1.3463612388642923
```

In [543]:
```
clf = SGDClassifier(alpha = optimal_alpha, penalty='l2', loss='log', random_st
ate=42)
clf.fit(train_x_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, y_train)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
print("Log Loss :",log_loss(y_cv, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv
_x_onehotCoding)- y_cv))/y_cv.shape[0])
plot_confusion_matrix(y_cv, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.2102792173052233
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```




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




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

Feature Engineering

In [515]:
```
tfidf = TfidfVectorizer()
train_gene_feature_onehotCoding = tfidf.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = tfidf.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = tfidf.transform(cv_df['Gene'])


train_variation_feature_onehotCoding = tfidf.fit_transform(train_df['Variatio
n'])
test_variation_feature_onehotCoding = tfidf.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = tfidf.transform(cv_df['Variation'])
```

In [516]:
```
tfidf = TfidfVectorizer(ngram_range = (1,4),max_features = 50000, min_df=3)
train_text_feature_onehotCoding = tfidf.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = tfidf.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = tfidf.transform(cv_df['TEXT'])

# Normalize the text feature vectors
# train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding,
axis=0)
# test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, a
xis=0)
# cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=
0)
```

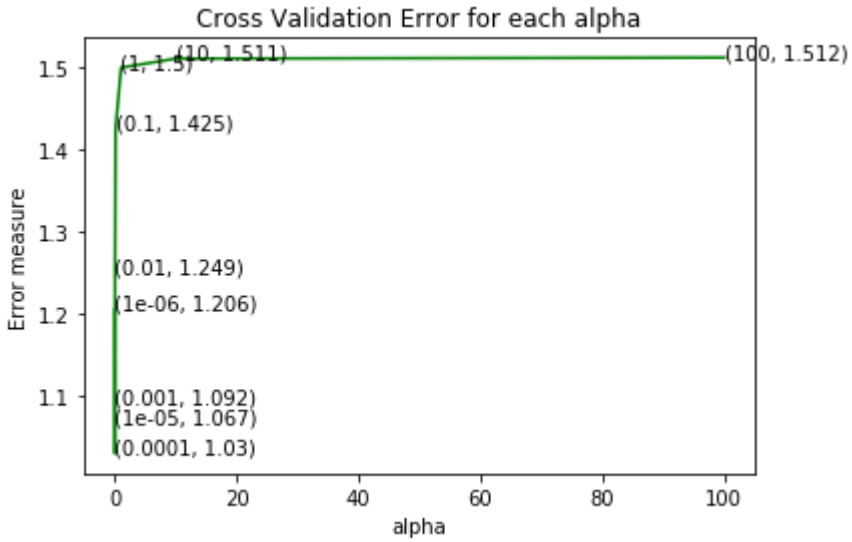
In [517]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

# Final Train data
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature
_onehotCoding)).tocsr()
y_train = train_df['Class']

# Final Test data
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_on
ehotCoding)).tocsr()
y_test = test_df['Class']

# Final CV data
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCo
ding)).tocsr()
y_cv = cv_df['Class']
```

In [518]:
```
print(train_x_onehotCoding.shape, y_train.shape)
print(cv_x_onehotCoding.shape, y_cv.shape)
print(test_x_onehotCoding.shape, y_test.shape)
```

```
(2124, 52198) (2124,)
(532, 52198) (532,)
(665, 52198) (665,)
```

In [519]:
```
# Train the model using Logistics Regression and Calibration model

alpha = [10**x for x in range(-6,3)]

cv_log_loss_values = []
for i in alpha:
    clf = SGDClassifier(class_weight = "balanced",alpha = i, loss = 'log', pen
alty = 'l2', random_state = 42)
    clf.fit(train_x_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method = 'sigmoid')
    sig_clf.fit(train_x_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_loss_values.append(log_loss(y_cv, predict_y,labels=clf.classes_, ep
s=1e-15))
    print("for alpha = ", i, "The log loss is ", log_loss(y_cv, predict_y,labe
ls=clf.classes_, eps=1e-15))

plt.plot(alpha, cv_log_loss_values, c = 'g')
for i, txt in enumerate(np.round(cv_log_loss_values,3)):
    plt.annotate((alpha[i],txt),(alpha[i],cv_log_loss_values[i]))

plt.title('Cross Validation Error for each alpha')
plt.xlabel('alpha')
plt.ylabel('Error measure')
plt.show()

optimal_alpha = alpha[np.argmin(cv_log_loss_values)]

clf = SGDClassifier(class_weight = "balanced",alpha = optimal_alpha, loss = 'l
og', penalty = 'l2', random_state = 42)
clf.fit(train_x_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method = 'sigmoid')
sig_clf.fit(train_x_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The train log loss is ", log_l
oss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The CV log loss is ",log_loss(
y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The test log loss is ", log_lo
ss(y_test, predict_y))
```

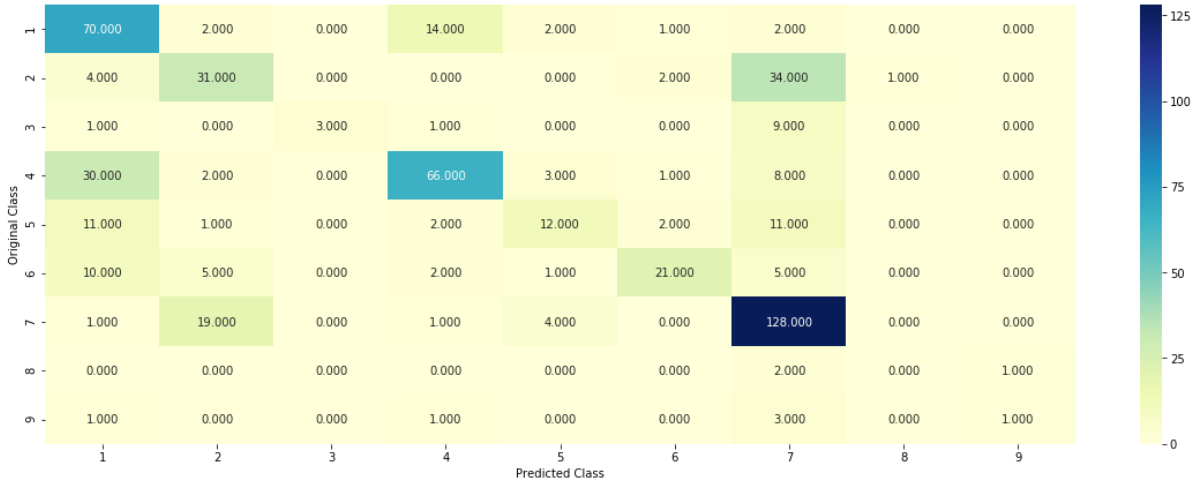
```
for alpha =  1e-06 The log loss is  1.205723093332427
for alpha =  1e-05 The log loss is  1.0671781492799948
for alpha =  0.0001 The log loss is  1.0304068761292855
for alpha =  0.001 The log loss is  1.0916224092080749
for alpha =  0.01 The log loss is  1.2491578124455045
for alpha =  0.1 The log loss is  1.424828973736361
for alpha =  1 The log loss is  1.499645609243859
for alpha =  10 The log loss is  1.5105588692500813
for alpha =  100 The log loss is  1.5118057888253564
```




```
For the value alpha :  0.0001 The train log loss is  0.4018758954607178
For the value alpha :  0.0001 The CV log loss is  1.0304068761292855
For the value alpha :  0.0001 The test log loss is  0.9711204762016626
```

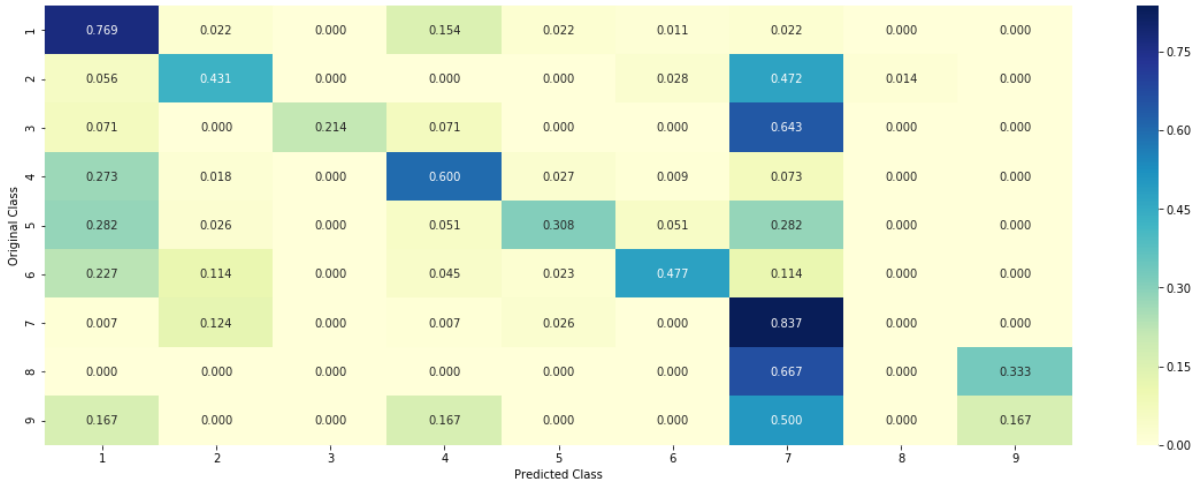
In [442]:
```
# Test the model on test data

clf = SGDClassifier(alpha = optimal_alpha, loss = 'log', penalty = 'l2', rando
m_state = 42)
clf.fit(train_x_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method = 'sigmoid')
sig_clf.fit(train_x_onehotCoding, y_train)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv
_x_onehotCoding)- y_cv))/y_cv.shape[0])
plot_confusion_matrix(y_cv, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

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




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


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



2nd Hack

In [544]:
```
# Ref : https://medium.com/@tulasiram11729/personalized-cancer-diagnosis-3d6f09a6b8c9
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [545]:
```
# one-hot encoding of Variation feature.
var_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = var_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = var_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = var_vectorizer.transform(cv_df['Variation'])
```

In [546]:
```
tfidf = TfidfVectorizer()
train_text_feature_onehotCoding = tfidf.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = tfidf.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = tfidf.transform(cv_df['TEXT'])
```

In [547]:
```
list = []
for word in result['Gene'].values:
    list.append(word)
for word in result['Variation'].values:
    list.append(word)
```

In [548]:
```
tfidf = TfidfVectorizer(ngram_range = (1,4),max_features = 25000, min_df=3)
list_vect = tfidf.fit_transform(list)
gene_variation_features = tfidf.get_feature_names()

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

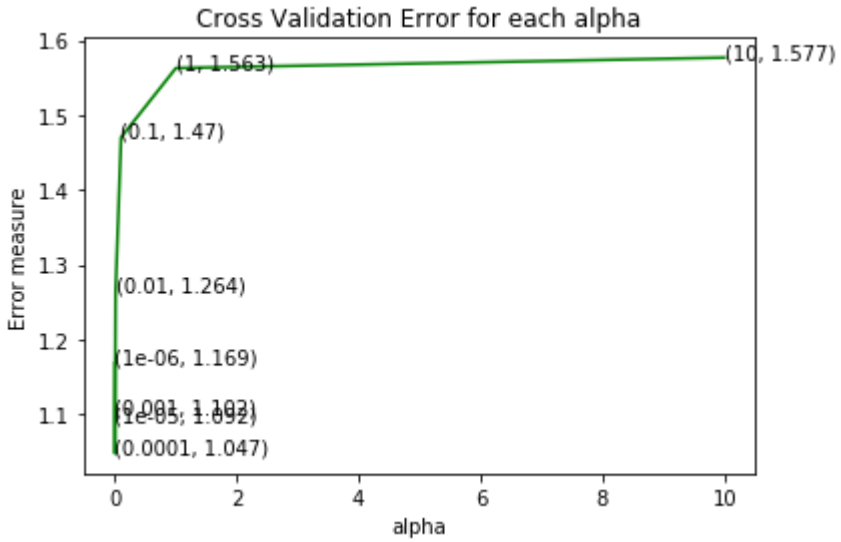
In [549]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

# Final Train data
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text, train_
text_feature_onehotCoding)).tocsr()
y_train = train_df['Class']

# Final Test data
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text, test_text
_feature_onehotCoding)).tocsr()
y_test = test_df['Class']

# Final CV data
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text, cv_text_feature
_onehotCoding)).tocsr()
y_cv = cv_df['Class']
```

In [550]:
```
print(train_x_onehotCoding.shape, y_train.shape)
print(cv_x_onehotCoding.shape, y_cv.shape)
print(test_x_onehotCoding.shape, y_test.shape)
```

```
(2124, 124376) (2124,)
(532, 124376) (532,)
(665, 124376) (665,)
```

In [551]:
```
# Train the model using Logistics Regression and Calibration model

alpha = [10**x for x in range(-6,2)]

cv_log_loss_values = []
for i in alpha:
    clf = SGDClassifier(class_weight = "balanced",alpha = i, loss = 'log', pen
alty = 'l2', random_state = 42)
    clf.fit(train_x_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method = 'sigmoid')
    sig_clf.fit(train_x_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_loss_values.append(log_loss(y_cv, predict_y,labels=clf.classes_, ep
s=1e-15))
    print("for alpha = ", i, "The log loss is ", log_loss(y_cv, predict_y,labe
ls=clf.classes_, eps=1e-15))

plt.plot(alpha, cv_log_loss_values, c = 'g')
for i, txt in enumerate(np.round(cv_log_loss_values,3)):
    plt.annotate((alpha[i],txt),(alpha[i],cv_log_loss_values[i]))

plt.title('Cross Validation Error for each alpha')
plt.xlabel('alpha')
plt.ylabel('Error measure')
plt.show()

optimal_alpha = alpha[np.argmin(cv_log_loss_values)]

clf = SGDClassifier(class_weight = "balanced",alpha = optimal_alpha, loss = 'l
og', penalty = 'l2', random_state = 42)
clf.fit(train_x_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method = 'sigmoid')
sig_clf.fit(train_x_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The train log loss is ", log_l
oss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The CV log loss is ",log_loss(
y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print("For the value alpha : ", optimal_alpha, "The test log loss is ", log_lo
ss(y_test, predict_y))
```

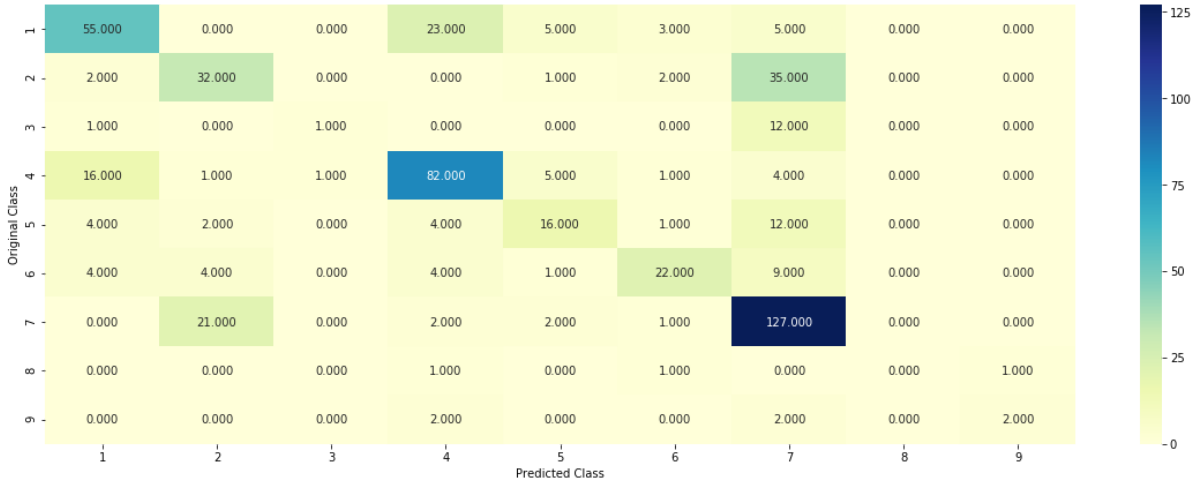
```
for alpha =  1e-06 The log loss is  1.1686493426592406
for alpha =  1e-05 The log loss is  1.0920477872710876
for alpha =  0.0001 The log loss is  1.0474041234306193
for alpha =  0.001 The log loss is  1.1024583966400152
for alpha =  0.01 The log loss is  1.2639668458454714
for alpha =  0.1 The log loss is  1.470227252242387
for alpha =  1 The log loss is  1.562774566582957
for alpha =  10 The log loss is  1.5767741582525021
```



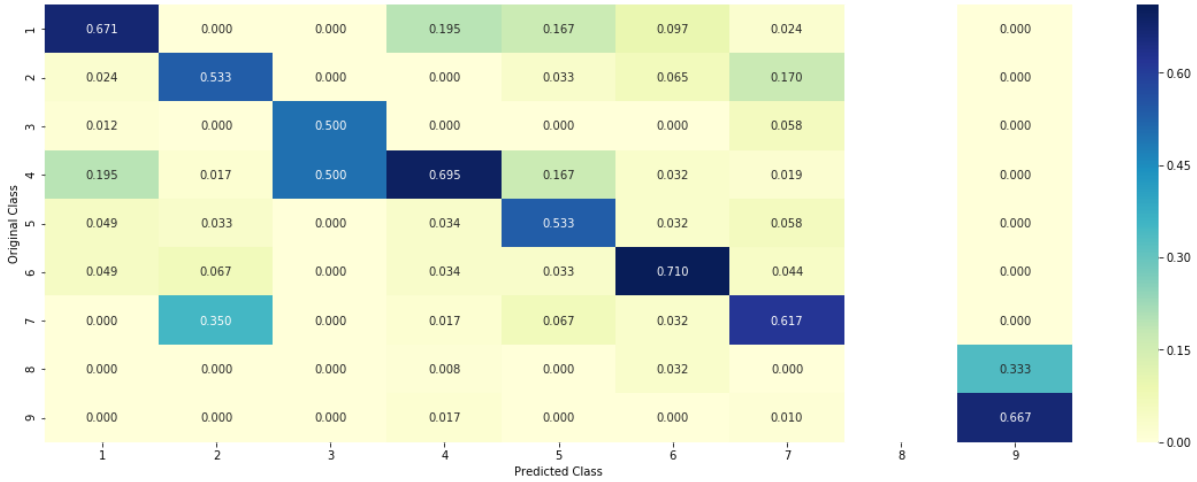

```
For the value alpha :  0.0001 The train log loss is  0.42118857766786044
For the value alpha :  0.0001 The CV log loss is  1.0474041234306193
For the value alpha :  0.0001 The test log loss is  0.9976629257025263
```

In [552]:
```
# Test the model on test data

clf = SGDClassifier(alpha = optimal_alpha, loss = 'log', penalty = 'l2', rando
m_state = 42)
clf.fit(train_x_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method = 'sigmoid')
sig_clf.fit(train_x_onehotCoding, y_train)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv
_x_onehotCoding)- y_cv))/y_cv.shape[0])
plot_confusion_matrix(y_cv, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

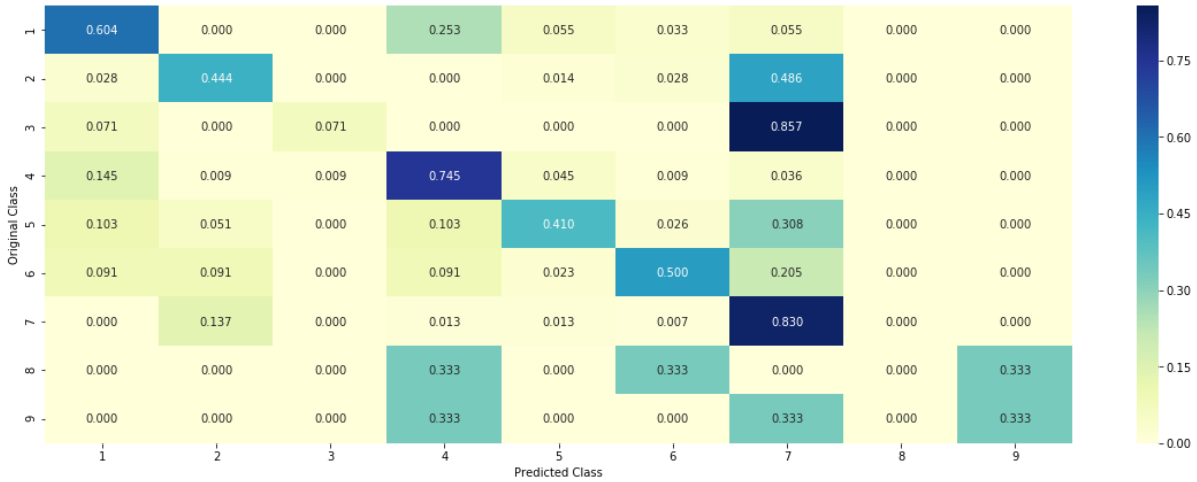
```
Number of missclassified point : 0.36654135338345867
-------------------- Confusion matrix --------------------
```




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




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



Complete process

1. Initially, we have 2 csv files, one having ID, Gene, Variation and Class. Other having ID and TEXT.
2. After converting into dataframes, the 2 dataframes are joined on ID.
3. The features are Gene, Variation and TEXT. Label is Class which is a multi-class classification(1-9)
4. Next, Text feature is preprocessed and Null values in Text are filled with Gene and Variation values.
5. Then the Data is split into train, test and CV
6. The features of string are converted into numerical vectors using two ways. One hot Encoding(BOW or tfidf) and Response coding(Mean Vlaue Replacement). These are selected based on the model
7. Univariate Analysis is done using any model to find the stability of each feature.
8. Then the three features are stacked horizontally to form the final features.
9. Applied the above models on the given features and found the log loss for each model.

Results

In [556]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No", "Model", "Train", "CV", "Test", "% of Misclassificati
on points"]

x.add_row([1, "Naive Bayes",0.724, 1.275, 1.216, 41.9])
x.add_row([2, "KNN",0.731, 1.11, 1.117,38.9])
x.add_row([3, "Logistic Regression with class Balancing",0.551, 1.037, 1.047,3
5.52])
x.add_row([4, "Logistic Regression without class Balancing",0.540, 1.083, 1.06
8, 35.15])
x.add_row([5, "Linear SVM",0.774, 1.150, 1.154, 36.84])
x.add_row([6, "Random Forest Classifier with One hot Encoding",0.837, 1.253,
1.228, 47.74])
x.add_row([7, "Random Forest Classifier with Response Coding",0.058, 1.348, 1.
314, 43.79])
x.add_row([8, "Stacking Model: LR+NB+SVM",0.789, 1.223, 1.189, 40.6])
x.add_row([8, "Majority Voting Classifier",0.928, 1.262, 1.218, 40.9])
x.add_row([9, "Logistic Regression with Count Vectorizer including Unigrams an
d Bigrams",1.081, 1.380, 1.346,39.28])
x.add_row([10, "Feature Engineering 1",0.401, 1.030, 0.971,37.59])
x.add_row([11, "Feature Engineering 2",0.421, 1.047, 0.997,36.65])

print(x)
```

S.No	Model			
Train	CV	Test	% of Misclassification points	
1	Naive Bayes			
0.724	1.275	1.216	41.9	
2	KNN			
0.731	1.11	1.117	38.9	
3	Logistic Regression with class Balancing			
0.551	1.037	1.047	35.52	
4	Logistic Regression without class Balancing			
0.54	1.083	1.068	35.15	
5	Linear SVM			
0.774	1.15	1.154	36.84	
6	Random Forest Classifier with One hot Encoding			
0.837	1.253	1.228	47.74	
7	Random Forest Classifier with Response Coding			
0.058	1.348	1.314	43.79	
8	Stacking Model: LR+NB+SVM			
0.789	1.223	1.189	40.6	
8	Majority Voting Classifier			
0.928	1.262	1.218	40.9	
9	Logistic Regression with Count Vectorizer including Unigrams and Bigrams			
1.081	1.38	1.346	39.28	
10	Feature Engineering 1			
0.401	1.03	0.971	37.59	
11	Feature Engineering 2			
0.421	1.047	0.997	36.65	

Final Conclusion

1. Logistic Regression has performed well compared to other models and Random Forest has performed very poor.
2. Logistic Regression with Count Vectorizer including unigrams and bigrams has given log loss of 1.346. So we can say that tfidf Vectorizer is better than Count Vectorizer for this model.
3. After applying Feature Engineering with Tfidf including 4grams, the log loss is decreased to 0.971 which is very good value.
4. When I consider the Gene and Variants Feature values to transform the Text feature values using Tfidf, the log loss is decreased to 0.997 which is also a good one.
5. We can still reduce the log loss by changing the tfidf Vectorizer parameters but we can't see much difference from the present value.
6. So, to reduce the log loss further domain knowledge is needed where we can add more features.