Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.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 [0]:

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

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

Out[0]:

	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 [0]:

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

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

Out[0]:

	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 [0]:

```
# 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 [0]:

```
#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 : 211.52816454299833 seconds
```

In [0]:

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

Out[0]:

	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 [0]:

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

Out[0]:

	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 [0]:

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

In [0]:

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

Out[0]:

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

In [3]:

```
result= pd.read_csv('result.csv')
```

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 [4]:

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

# split the data into test and train by maintaining same distribution of output varaibl
e 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, t
est_size=0.2)
# split the train data into train and cross validation by maintaining same distribution
of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, t
est_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 [5]:

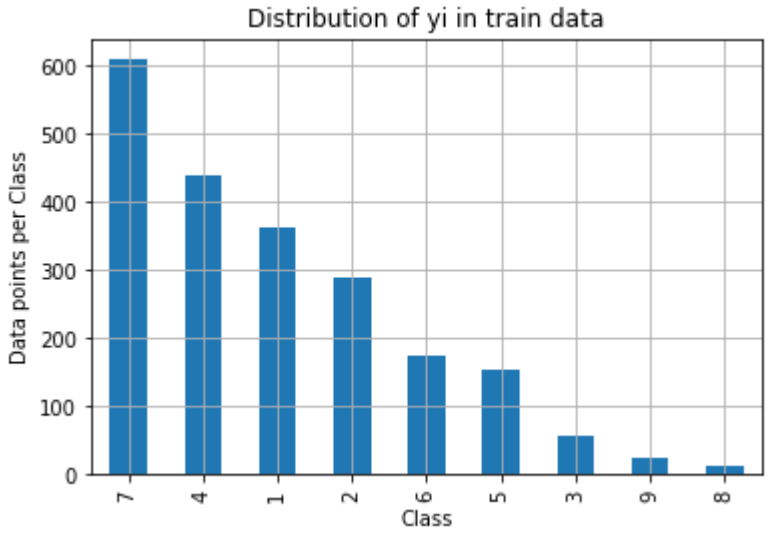
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

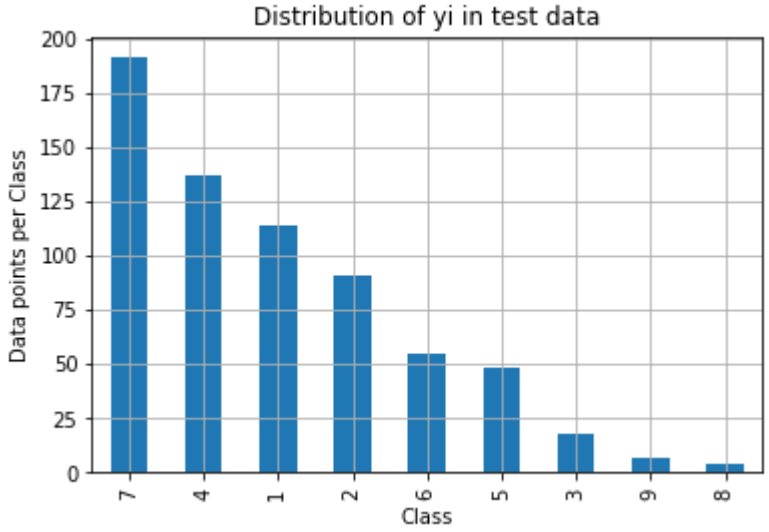
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [7]:

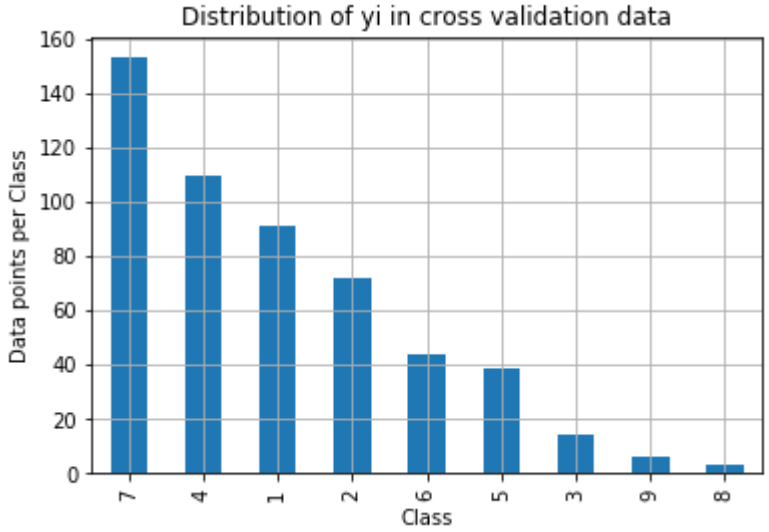
```
# it returns a dict, keys as class labels and values as the number of data points in th
at class
train_class_distribution = train_df['Class'].value_counts()
test_class_distribution = test_df['Class'].value_counts()
cv_class_distribution = cv_df['Class'].value_counts()

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

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

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

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




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



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




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

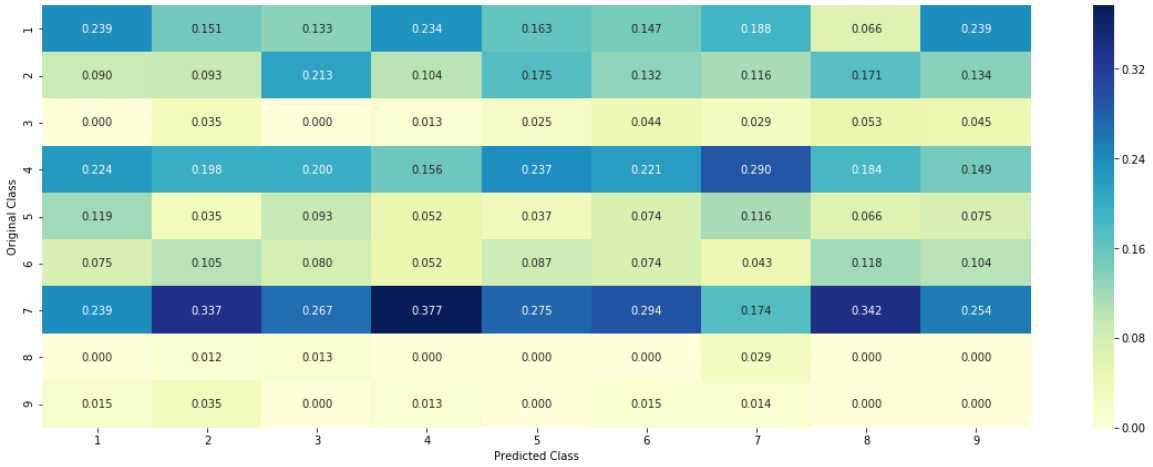
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predic
ted class j

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

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

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

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
wo 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, yticklabel
s=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, yticklabel
s=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, yticklabel
s=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [11]:

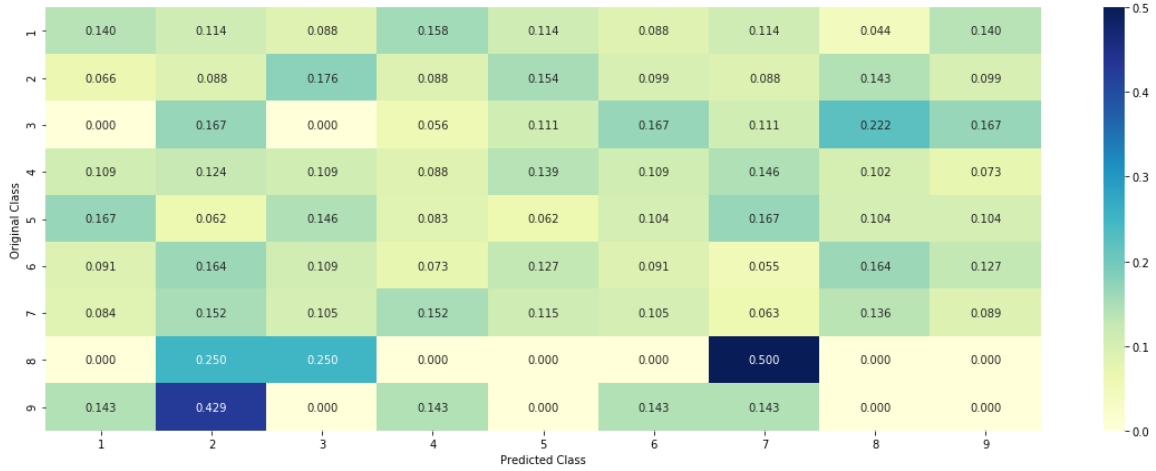
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.502814535579462
Log loss on Test Data using Random Model 2.5610408266323157
-------------------- Confusion matrix --------------------
```


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




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



Applying TFIDF vectorization with top 1000 words

3.3 Univariate Analysis

In [12]:

```
# 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 dat
a dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 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) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

    # denominator will contain the number of time that particular feature occured in wh
ole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pertic
ular 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 particul
ar 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, 0.068181818181818177, 0.
13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.
27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.
051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181
8177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917,
0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.0
72847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.0
66225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.
073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.
066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature valu
e 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 [13]:

```
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 : 236
BRCA1     170
TP53      102
EGFR       97
BRCA2      79
KIT        71
PTEN       64
BRAF       55
ERBB2      49
PDGFRA     40
ALK        40
Name: Gene, dtype: int64
```

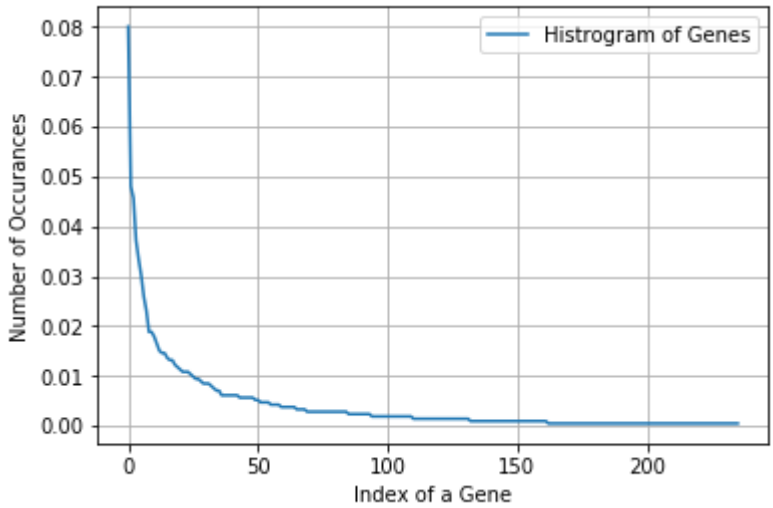
In [14]:

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

```
Ans: There are 236 different categories of genes in the train data, and th
ey are distibuted as follows
```

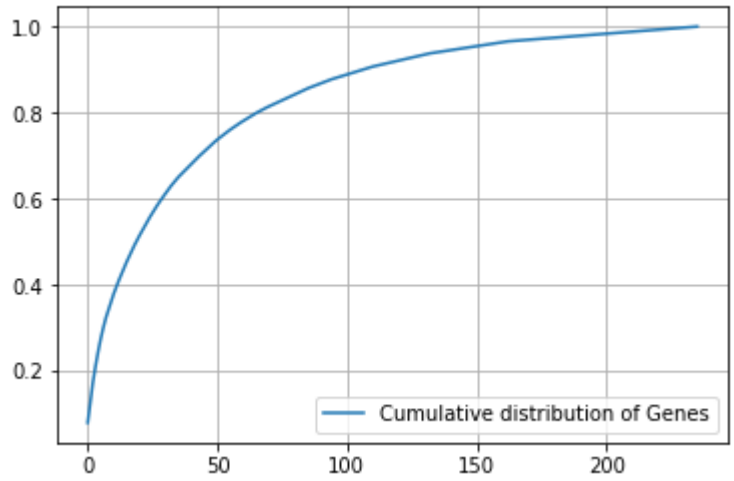
In [15]:

```
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 [16]:

```
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 [17]:

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

In [18]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding meth
od. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [19]:

```
# 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 [20]:

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

Out[20]:

```
1127      MET
570     SMAD3
807     ERCC2
204      EGFR
1569      ALK
Name: Gene, dtype: object
```

In [21]:

```
gene_vectorizer.get_feature_names()
```

Out[21]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'aurka',
 'aurkb',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
```

```
    'dnmt3a',
    'dusp4',
    'egfr',
    'eif1ax',
    'elf3',
    'ep300',
    'epas1',
    'erbb2',
    'erbb3',
    'erbb4',
    'ercc2',
    'ercc3',
    'ercc4',
    'erg',
    'esr1',
    'etv1',
    'etv6',
    'ewsr1',
    'ezh2',
    'fanca',
    'fancc',
    'fat1',
    'fbxw7',
    'fgf19',
    'fgf3',
    'fgf4',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'fgfr4',
    'flt1',
    'flt3',
    'foxa1',
    'foxl2',
    'foxp1',
    'gata3',
    'gli1',
    'gnas',
    'h3f3a',
    'hist1h1c',
    'hla',
    'hnf1a',
    'hras',
    'idh1',
    'idh2',
    'igf1r',
    'ikbke',
    'inpp4b',
    'jak1',
    'jak2',
    'jun',
    'kdm5a',
    'kdm5c',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'klf4',
    'kmt2a',
    'kmt2b',
    'kmt2c',
```

```
    'kmt2d',
    'knstrn',
    'kras',
    'lats2',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mapk1',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mga',
    'mlh1',
    'mpl',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myd88',
    'myod1',
    'nf1',
    'nf2',
    'nfe2l2',
    'nfkbia',
    'nkx2',
    'notch1',
    'npm1',
    'nras',
    'nsd1',
    'ntrk1',
    'ntrk3',
    'nup93',
    'pak1',
    'pbrm1',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
    'pim1',
    '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',
    'rictor',
    'rit1',
    'rnf43',
    'ros1',
    'rras2',
    'runx1',
    'rxra',
    'rybp',
    'sdhb',
    'sdhc',
    'setd2',
    'sf3b1',
    'shoc2',
    'smad2',
    'smad3',
    'smad4',
    'smarca4',
    'smarcb1',
    'smo',
    'sos1',
    'sox9',
    'spop',
    'src',
    'srsf2',
    'stag2',
    'stat3',
    'stk11',
    'tcf3',
    'tcf7l2',
    'tert',
    'tet1',
    'tet2',
    'tgfbr1',
    'tgfbr2',
    'tmprss2',
    'tp53',
    'tp53bp1',
    'tsc1',
    'tsc2',
    'u2af1',
    'vegfa',
    'vhl',
    'xpo1',
    'xrcc2',
    'yap1']
```

In [22]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encodin
g method. The shape of gene feature: (2124, 235)
```

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

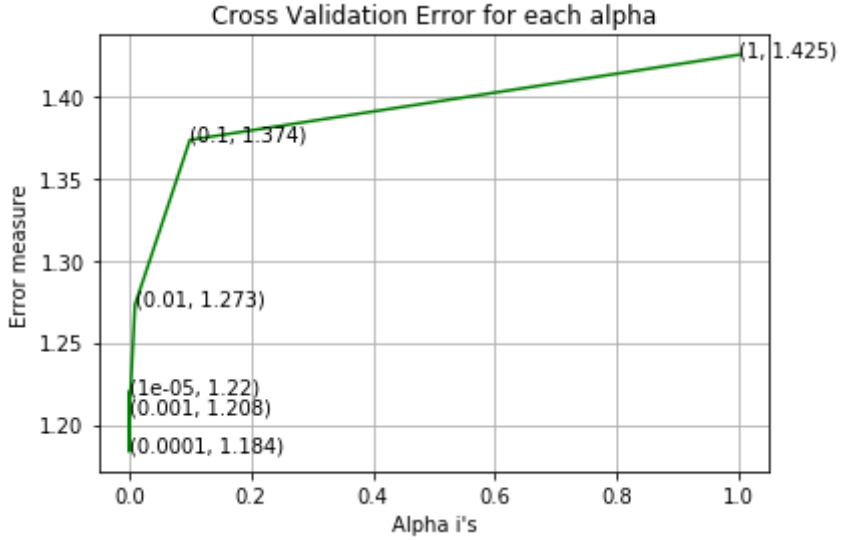
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [23]:

```
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/s
klearn.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='opt
imal', 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 Gradie
nt 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, lab
els=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_lo
ss(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 is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.21993040920516
For values of alpha =  0.0001 The log loss is: 1.1835948905909714
For values of alpha =  0.001 The log loss is: 1.2076145708563593
For values of alpha =  0.01 The log loss is: 1.2730489515122116
For values of alpha =  0.1 The log loss is: 1.3735167597951197
For values of alpha =  1 The log loss is: 1.4254453284289679
```




```
For values of best alpha =  0.0001 The train log loss is: 0.99678684678176
03
For values of best alpha =  0.0001 The cross validation log loss is: 1.183
5948905909714
For values of best alpha =  0.0001 The test log loss is: 1.166149201973616
8
```

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 [24]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  236
genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 519 out of  532 : 97.55639097744361
```

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 [25]:

```
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 : 1936
Truncating_Mutations    60
Deletion                44
Amplification           42
Fusions                 18
Overexpression           5
E17K                     3
G13C                     2
P130S                    2
A146V                    2
S308A                    2
Name: Variation, dtype: int64
```

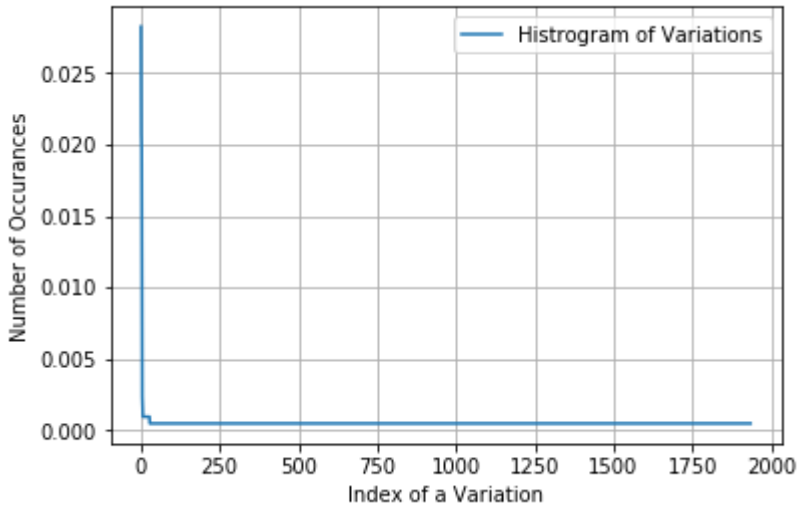
In [26]:

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

```
Ans: There are 1936 different categories of variations in the train data,
and they are distibuted as follows
```

In [27]:

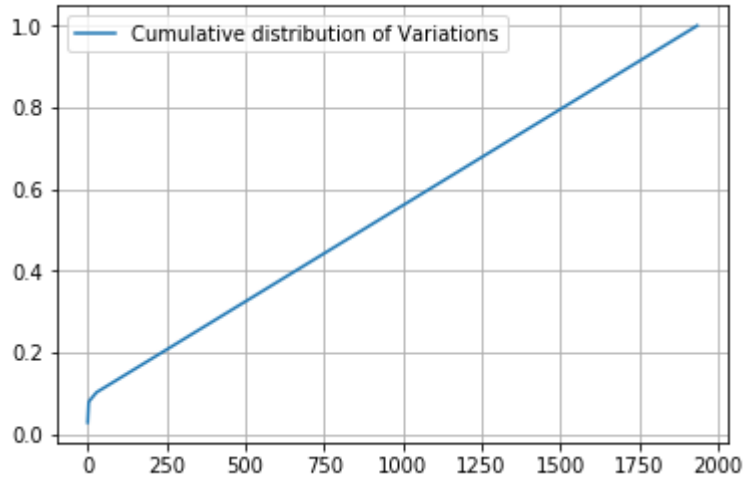
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [28]:

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

```
[0.02824859 0.04896422 0.06873823 ... 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 [29]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tr
ain_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tes
t_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df
))
```

In [30]:

```
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 re
sponse coding method. The shape of Variation feature: (2124, 9)
```

In [31]:

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

In [32]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot en
coding method. The shape of Variation feature:", train_variation_feature_onehotCoding.s
hape)
```

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

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

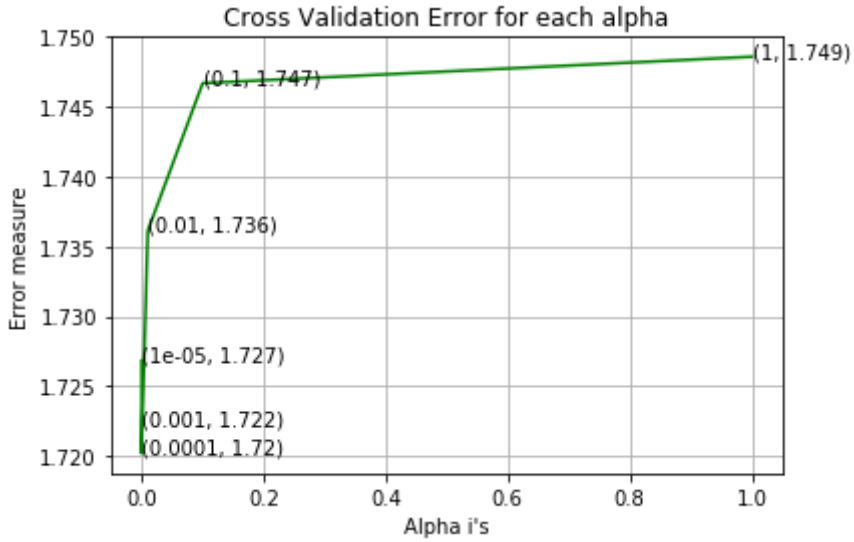
Let's build a model just like the earlier!

In [33]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
klearn.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='opt
imal', 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 Gradie
nt 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, lab
els=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 is:",log_lo
ss(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 is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7268239827763958
For values of alpha =  0.0001 The log loss is: 1.7201935620727473
For values of alpha =  0.001 The log loss is: 1.7223036635418707
For values of alpha =  0.01 The log loss is: 1.7361090386583584
For values of alpha =  0.1 The log loss is: 1.7466548034261693
For values of alpha =  1 The log loss is: 1.7485465340121547
```




```
For values of best alpha =  0.0001 The train log loss is: 0.73226250995896
66
For values of best alpha =  0.0001 The cross validation log loss is: 1.720
1935620727473
For values of best alpha =  0.0001 The test log loss is: 1.674432780214561
```

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 [34]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], "
 genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shap
e[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_covera
ge/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cove
rage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1936  genes in test and cr
oss validation data sets?
Ans
1. In test data 77 out of 665 : 11.578947368421053
2. In cross validation data 50 out of  532 : 9.398496240601503
```

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 [35]:

```
# 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 [36]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(wo
rd,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEX
T'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [37]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train d
ata
text_vectorizer = TfidfVectorizer(min_df=3, max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numb
er 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 [38]:

```
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 [39]:

```
#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 [40]:

```
# 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_fea
ture_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featur
e_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_resp
onseCoding.sum(axis=1)).T
```

In [41]:

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

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

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

In [42]:

```
#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 [43]:

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

```
Counter({9.266682484159043: 1, 8.725859173846212: 1, 34.712290672853825:
1, 17.562512473189138: 1, 5.752809047568143: 1, 6.9696793693638766: 1, 7.8
79979628115243: 1, 8.333219876885245: 1, 9.689306381867326: 1, 10.11014251
2244606: 1, 11.586531285635296: 1, 12.80641536045814: 1, 13.25036200630794
2: 1, 14.38404077647723: 1, 15.575977288828097: 1, 16.042494620829117: 1,
17.223472009882407: 1, 18.108463614246777: 1, 19.544757230938416: 1, 20.76
6537443159116: 1, 21.480283371881935: 1, 22.921805816837363: 1, 23.1744278
2634282: 1, 24.257171057696794: 1, 25.974281401679146: 1, 26.3427233958820
5: 1, 27.450461022721658: 1, 28.353431717033107: 1, 29.395056544349004: 1,
30.60232060381817: 1, 31.92031343102982: 1, 32.53642419641201: 1, 33.60134
838168781: 1, 34.73508782235693: 1, 35.46967152287078: 1, 36.5630722733932
7: 1, 6.877568905441875: 1, 38.04049487056047: 1, 39.07546488307855: 1, 4
0.515373086198565: 1, 41.53587579071896: 1, 42.528448129265556: 1, 7.24198
4138323475: 1, 44.42242527166129: 1, 45.26582791356535: 1, 46.896718586579
67: 1, 47.526209851091096: 1, 48.891535121534105: 1, 8.852641201696446: 1,
10.148359114825578: 1, 10.162432250135591: 1, 52.78509901806109: 1, 15.548
945417260418: 1, 54.95970368085532: 1, 9.370974034592901: 1, 56.1455169266
4318: 1, 7.84428238825516: 1, 59.16290424254553: 1, 60.62694046999782: 1,
10.069526148692528: 1, 62.9753497626508: 1, 63.64966707207451: 1, 64.53609
581185852: 1, 65.45123904611904: 1, 11.085517128333885: 1, 68.175586450994
5: 1, 71.1627272795254: 1, 15.088189385931452: 1, 12.019289241381903: 1, 1
7.04978406549129: 1, 75.79973502883901: 1, 76.35890989103724: 1, 77.104319
11057408: 1, 28.91268069562658: 1, 13.02165627516661: 1, 11.42857015481092
4: 1, 81.54841885825739: 1, 82.73337621545188: 1, 83.67251732248616: 1, 1
9.76527959059954: 1, 14.267536397810252: 1, 87.67504642165814: 1, 7.534787
489447266: 1, 10.36275482726696: 1, 90.50175116994794: 1, 15.4024401478877
54: 1, 9.581248191039773: 1, 20.284223170416844: 1, 16.92567395144341: 1,
17.139911602791507: 1, 105.69316733427148: 1, 106.84553049106638: 1, 14.17
4590802953514: 1, 29.2106262402033: 1, 18.83709402988365: 1, 12.0692406102
30079: 1, 19.341039409517396: 1, 117.56894580761268: 1, 8.003286877324483:
1, 8.1226220148986: 1, 20.978364984480542: 1, 38.58700333654953: 1, 21.305
140447101785: 1, 21.334239179530137: 1, 128.9406012914563: 1, 7.9265612920
464115: 1, 130.99705508640116: 1, 22.984347043580698: 1, 14.89170028283250
6: 1, 9.219825491380634: 1, 23.867374767882414: 1, 13.135359966381767: 1,
25.320285879217952: 1, 10.95719429795827: 1, 24.99559969866493: 1, 9.60321
6459725642: 1, 11.04621325801776: 1, 25.431852286282165: 1, 7.399037994875
896: 1, 26.859328469550853: 1, 27.898272058828276: 1, 9.830258883720363:
1, 16.046608880363028: 1, 28.655888180975655: 1, 14.304255035197652: 1, 2
9.962884694868425: 1, 8.711437069651257: 1, 10.874806197748397: 1, 179.541
9141255137: 1, 30.80802649963381: 1, 12.593774132328827: 1, 23.88601846112
2312: 1, 137.853620364891: 1, 31.450780890950345: 1, 18.39805818713892: 1,
32.35592996297467: 1, 9.028172094242947: 1, 33.40714747128258: 1, 46.63220
200591328: 1, 34.626606228699316: 1, 16.62428599940055: 1, 11.256222877299
079: 1, 35.07907135454809: 1, 41.78633661323572: 1, 7.6333997699138365: 1,
36.042830470469: 1, 14.333402223519844: 1, 9.613523983119247: 1, 37.049866
66790907: 1, 7.931650975580635: 1, 12.489900003230586: 1, 14.4389822244858
8: 1, 38.755558964900935: 1, 16.735798870629505: 1, 10.495220859567715: 1,
11.438915631521278: 1, 39.055363873968176: 1, 9.020156705566311: 1, 12.611
315315338969: 1, 7.648254922529065: 1, 40.16904905078833: 1, 8.85848689055
2967: 1, 41.51561632574906: 1, 8.173589970092223: 1, 249.753130044773: 1,
10.243928129404065: 1, 6.409804865283871: 1, 11.948165061335365: 1, 42.154
63318370715: 1, 13.46240161177579: 1, 43.74697172548792: 1, 17.23959796180
9547: 1, 7.758475909676121: 1, 13.39351446974956: 1, 9.395974944684287: 1,
31.76866643155447: 1, 26.961619775318262: 1, 9.09223593960077: 1, 9.887367
755747363: 1, 46.395153411960955: 1, 8.832450005981546: 1, 7.9530771599551
72: 1, 9.223238464451606: 1, 21.419846960226856: 1, 10.980929749444458: 1,
48.43108456918664: 1, 18.3741831359886: 1, 7.690901128395081: 1, 49.504028
04619316: 1, 14.306957552837785: 1, 12.082754394273197: 1, 13.957213161821
01: 1, 27.82908005619813: 1, 10.23381117657803: 1, 18.24059677028362: 1, 1
8.82219345862938: 1, 9.76322732628089: 1, 6.5721110888156336: 1, 9.0880289
60779962: 1, 8.74764441646294: 1, 9.120872472563219: 1, 13.97097155641046
```

1: 1, 8.150624440081796: 1, 15.796242059098658: 1, 55.758467517298385: 1,
28.56082601335904: 1, 7.446931463965217: 1, 7.661633498916455: 1, 17.20787
8313606567: 1, 10.050800359104048: 1, 7.5640488990893875: 1, 19.1793569317
9292: 1, 17.093295395898426: 1, 20.35255337592994: 1, 11.200991223540939:
1, 16.91125245014597: 1, 11.258953676606627: 1, 8.45763056077284: 1, 12.11
8989576878647: 1, 61.981942115215226: 1, 7.980298552612082: 1, 8.665308420
20231: 1, 8.319996708293235: 1, 11.684271431819287: 1, 15.493641008482033:
1, 63.746238149468965: 1, 21.851137168073077: 1, 21.726168862288215: 1, 7.
539672099029031: 1, 13.002066236126979: 1, 20.759926728043265: 1, 67.47080
533000131: 1, 9.576625628485234: 1, 18.541488010079394: 1, 8.9669159809272
72: 1, 7.457273473076643: 1, 68.90324190547159: 1, 22.750128069611716: 1,
7.055100894973308: 1, 13.509346298020077: 1, 12.939064674124097: 1, 14.869
097106025869: 1, 8.352152371646962: 1, 31.49133041282993: 1, 14.6714645357
71605: 1, 14.927639978064047: 1, 17.21018106030176: 1, 9.029282600742226:
1, 10.74858152735945: 1, 18.760410098012702: 1, 9.40658013832789: 1, 8.567
228144962804: 1, 10.971752481412095: 1, 10.49582044891318: 1, 9.8379525231
3426: 1, 75.56241124306196: 1, 49.78872764431857: 1, 10.182794798239717:
1, 15.29509097411892: 1, 11.362817221309367: 1, 8.655818088927786: 1, 17.4
06052666122005: 1, 8.01747942250209: 1, 21.561733450115632: 1, 11.54876979
290873: 1, 22.522530549816924: 1, 9.891796971143599: 1, 10.90408828546817
3: 1, 8.762564054204635: 1, 7.7402626635236835: 1, 9.745390650931308: 1, 7
9.10040909206144: 1, 21.703370953741732: 1, 10.636409604224058: 1, 10.6760
55410846772: 1, 12.51014358949183: 1, 16.126587541603953: 1, 9.74841686267
9962: 1, 8.883032973921628: 1, 8.338557865604765: 1, 12.665100689310721:
1, 8.13984263540091: 1, 15.588659491020284: 1, 8.74052381809793: 1, 9.2475
66959037178: 1, 14.037817324600518: 1, 33.727997582532055: 1, 10.735507266
144396: 1, 23.351569310021137: 1, 16.76430657726619: 1, 17.83909757775950
3: 1, 9.982267799352556: 1, 16.37496199390888: 1, 10.730946549289518: 1, 1
3.755754410601503: 1, 14.88492566236604: 1, 13.292808477740524: 1, 11.2498
98099187474: 1, 19.86588782312345: 1, 11.10983523867944: 1, 8.859929700433
3: 1, 9.487457404028849: 1, 8.953376402598304: 1, 7.1029689726972025: 1, 1
1.547510301288472: 1, 12.760197353570499: 1, 9.706311691033275: 1, 16.0168
63517175423: 1, 18.90866627759397: 1, 7.7825104502322695: 1, 18.0000738119
32424: 1, 15.225581769929894: 1, 13.617705063813606: 1, 7.764228002777295:
1, 16.660137599584193: 1, 13.308115445809827: 1, 14.848957020933485: 1, 8.
50077425915468: 1, 10.05931047381406: 1, 27.228345213621107: 1, 11.8275813
88529271: 1, 10.329585175213456: 1, 8.761182844410483: 1, 9.1390367296465
7: 1, 16.66248455155866: 1, 7.715679784641491: 1, 6.9054669999960465: 1,
6.833052879960439: 1, 10.646702851605308: 1, 11.27732005123382: 1, 36.3862
5287092861: 1, 19.509052712519434: 1, 20.355169265579867: 1, 11.5968741366
70886: 1, 15.642884823832192: 1, 13.894414959471197: 1, 18.77209417305171
3: 1, 11.656995833759337: 1, 10.023502504642055: 1, 11.889910050373889: 1,
10.514836862808504: 1, 10.637527535467013: 1, 7.809045562754539: 1, 9.3291
67133334119: 1, 15.902980442958432: 1, 20.98958245793341: 1, 16.7294093934
63: 1, 9.387677745895358: 1, 38.139951226368474: 1, 14.124810856847386: 1,
29.81839522661411: 1, 17.346964789983: 1, 12.121373494575185: 1, 12.712654
729233495: 1, 18.67081831135906: 1, 14.997734493151789: 1, 8.0639123206275
71: 1, 9.304624122410198: 1, 8.236018626499819: 1, 21.573731079067812: 1,
11.040609871783765: 1, 15.522704443741736: 1, 14.127326937276447: 1, 30.62
9251962655868: 1, 8.399806012444996: 1, 13.765589276165603: 1, 25.38932391
2314328: 1, 12.47711378748254: 1, 10.090702722742844: 1, 9.16048415332031
7: 1, 110.73650419403474: 1, 7.728205581900725: 1, 7.2679804754199955: 1,
13.455330386069518: 1, 22.030323325848542: 1, 15.561534622413793: 1, 8.395
580761660463: 1, 8.565020823538793: 1, 14.058810337440017: 1, 31.520937858
849887: 1, 9.666342898507851: 1, 14.117874962376622: 1, 12.53264811498968
5: 1, 12.744146735748759: 1, 9.636795592953636: 1, 8.621930423903976: 1, 1
0.719221749086364: 1, 115.60928071361998: 1, 21.14757393528649: 1, 40.7915
9171295253: 1, 23.011434207655896: 1, 11.97480307767582: 1, 6.773299614684
568: 1, 10.209955052213322: 1, 21.770353281271618: 1, 117.12008391830254:
1, 8.216491860147297: 1, 12.76343188073339: 1, 7.411912029343403: 1, 10.71
2905135672974: 1, 15.58889748055427: 1, 15.024188885643971: 1, 9.248500214

213482: 1, 7.740885381254505: 1, 11.779714836711184: 1, 10.12960259105760
1: 1, 17.18338640475403: 1, 24.985341677731014: 1, 28.143636648469545: 1,
7.020358045750875: 1, 11.889796400951045: 1, 8.045040156125486: 1, 14.7249
78515389797: 1, 7.803059493970428: 1, 16.392866153463277: 1, 16.9612228555
74352: 1, 12.219727967334142: 1, 42.15640696887524: 1, 25.22230029226276:
1, 8.656379663320628: 1, 11.624304063807745: 1, 8.36771278100908: 1, 13.20
2422667433353: 1, 19.153750806252155: 1, 8.018587736182198: 1, 17.77945611
887697: 1, 12.416194442003894: 1, 7.57503993038655: 1, 13.340546890027445:
1, 9.800547946983793: 1, 8.80096563719033: 1, 43.245121881413404: 1, 26.78
900950763304: 1, 9.111341842816934: 1, 17.08943423347148: 1, 16.6078743363
96275: 1, 7.3400875658602525: 1, 23.767403746503703: 1, 20.59013920187979
7: 1, 18.06546587446761: 1, 10.799707815087931: 1, 10.780760899176228: 1,
10.31454537145074: 1, 14.115119961803662: 1, 19.190672683714503: 1, 16.003
052606034963: 1, 9.853927969254462: 1, 10.776359726396556: 1, 10.391592636
4464: 1, 11.95317806319748: 1, 22.157089169261756: 1, 16.212166932595064:
1, 12.294222774181733: 1, 19.408332738074936: 1, 23.824462299966928: 1, 7.
6243722756895576: 1, 7.795959674944332: 1, 35.1310630005306: 1, 15.8638158
80571456: 1, 11.16303233293021: 1, 7.741676929931514: 1, 11.34870276887691
2: 1, 21.401171377912522: 1, 7.460281575290262: 1, 15.371175442425333: 1,
12.69428097928533: 1, 9.048067900770183: 1, 20.034832241192223: 1, 11.8661
80514994404: 1, 12.193624307632346: 1, 10.42793869741371: 1, 18.6949049848
05447: 1, 29.27004522844507: 1, 21.194234297436083: 1, 12.300136961603364:
1, 19.408183124147648: 1, 7.208608416481522: 1, 8.48379914595254: 1, 8.800
028765483395: 1, 6.841348476182655: 1, 14.197128481651152: 1, 21.652769297
11104: 1, 15.604501419907828: 1, 21.99113517753388: 1, 8.520544281876733:
1, 11.453926428249867: 1, 8.077406053500876: 1, 19.233696470431287: 1, 30.
266664890453853: 1, 8.934037253017008: 1, 13.490590951966063: 1, 14.605314
357548847: 1, 22.29616447207695: 1, 9.413601795684722: 1, 10.5935301042946
75: 1, 16.104934006165077: 1, 13.6233922868535: 1, 22.40328921071202: 1, 1
0.204142939975284: 1, 36.40874812567044: 1, 13.882373863420057: 1, 12.4069
37004528109: 1, 8.573673955960922: 1, 21.421509710847683: 1, 16.3291448051
0995: 1, 14.235857004971287: 1, 12.059431583430753: 1, 25.427489883987153:
1, 10.695940893043053: 1, 17.69118511082202: 1, 8.30318065694017: 1, 10.10
9996838500601: 1, 10.325502780885781: 1, 7.4614634435185945: 1, 23.8386464
61517417: 1, 14.445362518820124: 1, 6.839123826832417: 1, 7.12324935463220
5: 1, 22.256751461757403: 1, 18.796607420857036: 1, 7.943788143539935: 1,
11.41047657075759: 1, 32.530729406052444: 1, 12.059652724523426: 1, 10.760
405571028492: 1, 8.975458069597464: 1, 8.051666839309915: 1, 11.4157526205
13624: 1, 18.4044876622411: 1, 9.490315889852283: 1, 9.701704728151986: 1,
7.700125151760414: 1, 10.614902502981053: 1, 9.134029182440633: 1, 10.0188
98816497243: 1, 67.55445881063959: 1, 14.863585067912204: 1, 13.4428042580
79736: 1, 16.892342945714326: 1, 13.684682662081297: 1, 12.66682579651922
9: 1, 7.978431680739727: 1, 12.21039245645571: 1, 25.066971616163134: 1, 2
1.732422450238406: 1, 8.184577058437338: 1, 13.288170236863008: 1, 12.7285
41162342518: 1, 17.099827076272152: 1, 8.782048730137683: 1, 17.5434750784
79368: 1, 9.459194478363278: 1, 22.51537057504502: 1, 8.828640169934053:
1, 15.632689425930849: 1, 10.893590794404652: 1, 13.258861814314523: 1, 1
6.68583150556885: 1, 16.546427450984837: 1, 26.720370066747765: 1, 9.79462
9773869769: 1, 7.886705261060212: 1, 9.420153200398513: 1, 12.479425316133
753: 1, 18.084890673892637: 1, 10.284789386757742: 1, 9.635426344076707:
1, 14.63677739946514: 1, 8.803557541188395: 1, 25.578498776305924: 1, 12.4
63413140176275: 1, 27.106590250336186: 1, 19.12399805504662: 1, 10.1091625
98762065: 1, 13.108953543243963: 1, 6.9502655711025465: 1, 7.6176617900040
25: 1, 19.7384359846019: 1, 8.023120408632675: 1, 15.993814953784872: 1, 1
0.293476601119645: 1, 11.882260726758698: 1, 19.642798584626583: 1, 18.169
641411091547: 1, 17.17681401289454: 1, 7.53848095537718: 1, 19.64746859025
0593: 1, 10.059412696633714: 1, 8.13021461543553: 1, 7.498413372339387: 1,
20.224650942608044: 1, 10.264881716370825: 1, 9.711504076437336: 1, 10.690
885243657581: 1, 11.378686724670429: 1, 12.592491033694142: 1, 11.05358202
6288709: 1, 8.813797090622542: 1, 6.527872162661978: 1, 38.67018398903534:
1, 21.08929329811469: 1, 12.433071053284738: 1, 18.95316160194339: 1, 17.0

71130283628513: 1, 30.34456661772956: 1, 14.185573857156808: 1, 13.8436640
76231573: 1, 10.278653526155958: 1, 9.626807872624758: 1, 9.16720155508764
6: 1, 9.663296360695439: 1, 9.598233203429482: 1, 9.25895923703032: 1, 22.
20380398304642: 1, 26.75255859358154: 1, 15.710925138056195: 1, 9.55711523
1031279: 1, 8.718090433495702: 1, 31.31643480548465: 1, 11.42685161966882:
1, 14.66223145362731: 1, 12.225718683154243: 1, 16.32975057331387: 1, 16.1
52328602761962: 1, 9.243075545911756: 1, 10.297031136658445: 1, 7.86727091
39778165: 1, 40.03587535108312: 1, 23.62224292231017: 1, 14.3546885337104
2: 1, 7.531754560269834: 1, 30.459108267051995: 1, 11.893730449903808: 1,
15.123566721636632: 1, 22.967452003950118: 1, 11.953685841134002: 1, 19.57
749573145606: 1, 18.446937816444137: 1, 24.72020272514164: 1, 7.3373287350
65782: 1, 16.1995245496777: 1, 31.507174705974148: 1, 15.031176436726865:
1, 16.739789308963434: 1, 12.289894972329792: 1, 12.482661599415463: 1, 4
2.883283725510154: 1, 8.423702879763297: 1, 8.096544276960135: 1, 19.19947
7828270865: 1, 10.566363194556194: 1, 10.507704756662388: 1, 14.3911553613
64177: 1, 17.49325611999158: 1, 15.716052424474407: 1, 22.987510317943453:
1, 8.887438815300213: 1, 13.926679628337167: 1, 9.064613338772253: 1, 9.22
2970870129595: 1, 9.017627760611518: 1, 16.812647836944574: 1, 7.971144009
390302: 1, 17.67057599223324: 1, 28.47866520116413: 1, 18.24004181200858:
1, 8.054981858423597: 1, 14.504678708715074: 1, 8.151843013482774: 1, 10.0
10511376569426: 1, 9.181907768947646: 1, 10.825864510162234: 1, 19.9560244
5916013: 1, 31.74555340903316: 1, 15.123828497518799: 1, 15.97871391139209
3: 1, 28.031053841157906: 1, 11.806238197965044: 1, 12.199006912971099: 1,
11.342118160899243: 1, 8.397343042611967: 1, 17.134532075612157: 1, 16.172
11942303457: 1, 7.047235896406066: 1, 7.400984401822498: 1, 18.79427178760
9055: 1, 20.819030582643432: 1, 17.55397281990781: 1, 15.559756634830311:
1, 8.618846781017092: 1, 13.83230965266701: 1, 16.282845060912553: 1, 12.3
1048208854331: 1, 7.028770813353309: 1, 14.081624500015224: 1, 11.39010209
2204915: 1, 8.109962516949922: 1, 8.238534220977233: 1, 6.806234512805984:
1, 9.862193260876259: 1, 21.085452214509484: 1, 10.394638631847217: 1, 13.
39571111075057: 1, 8.978263008902127: 1, 21.70245877467307: 1, 30.53201612
4970035: 1, 13.015944415893117: 1, 9.889141787966668: 1, 12.58394706729768
4: 1, 9.100349723403484: 1, 7.891461796564516: 1, 9.533109435254085: 1, 2
2.451827273943348: 1, 7.585600989711753: 1, 10.05010885843679: 1, 16.74636
1780708316: 1, 31.517629261389352: 1, 31.645428861139337: 1, 14.8711735582
86287: 1, 18.002937050861306: 1, 16.44020594538623: 1, 18.702484261947397:
1, 10.761423515338537: 1, 21.651431616114255: 1, 12.356372804980829: 1, 1
1.81283439003052: 1, 6.943271492150917: 1, 7.173805678355054: 1, 8.5140097
38694192: 1, 10.39299042929483: 1, 46.12102212750501: 1, 32.0326813631003
3: 1, 15.032015659331861: 1, 10.744289142669611: 1, 8.507472019615703: 1,
12.90204989341392: 1, 11.901124839133773: 1, 9.812711377671517: 1, 10.0704
45284946896: 1, 10.126931553052943: 1, 9.492124699922265: 1, 7.05749726946
7252: 1, 8.304524291754458: 1, 8.895755386529865: 1, 16.249952497680866:
1, 24.418158952674812: 1, 8.639120182023637: 1, 8.545445797410094: 1, 21.5
86815022578037: 1, 12.583706645616774: 1, 24.438366909703262: 1, 21.753687
00652589: 1, 8.659387327419143: 1, 14.630509032681022: 1, 8.44989308854328
4: 1, 31.043664947952575: 1, 8.311850621423298: 1, 17.225632130926456: 1,
9.03897024211308: 1, 10.37098340551717: 1, 36.52844943174737: 1, 8.0038470
33376893: 1, 9.594867590783725: 1, 20.89666408545014: 1, 14.79872991088684
1: 1, 10.856255909569724: 1, 7.5536628689726415: 1, 10.766932798985312: 1,
26.619674724832695: 1, 16.52724291420607: 1, 35.392350399179584: 1, 18.342
840558498764: 1, 10.820482803670433: 1, 17.687153819214416: 1, 19.95798559
917274: 1, 16.742803347424402: 1, 14.473440173436204: 1, 17.98562392176394
2: 1, 12.680706929620332: 1, 27.557243320848375: 1, 9.276345938113229: 1,
10.081776225737247: 1, 9.001113278259352: 1, 8.740614357730683: 1, 10.9389
46580174816: 1, 19.016133147213324: 1, 10.711452402632327: 1, 32.089566228
199985: 1, 14.99917371092181: 1, 12.51709415987115: 1, 15.883691521584824:
1, 11.857230081965636: 1, 11.901302758911042: 1, 13.505421280966885: 1, 1
6.21884088882903: 1, 11.098311353031267: 1, 18.853029934668786: 1, 9.13856
7752378883: 1, 38.93500505979795: 1, 13.00462434664554: 1, 37.661398371308
63: 1, 20.88886453648466: 1, 8.596231268430214: 1, 8.343912195523112: 1, 1

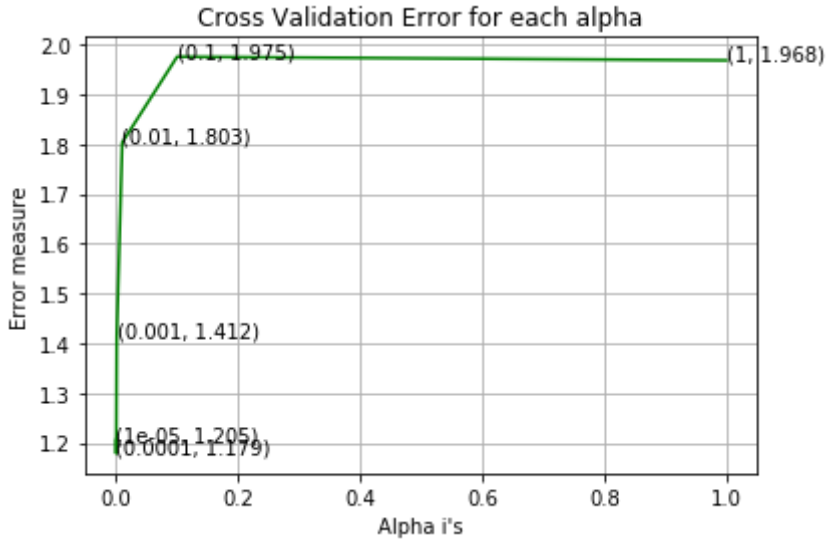
7.477790802693825: 1, 16.298015161327683: 1, 18.155866984935955: 1, 29.431
358020633024: 1, 12.252266447820165: 1, 16.188525169439078: 1, 18.02137880
6261826: 1, 8.026274345046025: 1, 8.442068368080147: 1, 9.548095512940174:
1, 6.801395159791233: 1, 10.676287611630926: 1, 21.311859458332137: 1, 11.
185177311567307: 1, 14.207939964844497: 1, 10.206482069666665: 1, 27.03092
2054246414: 1, 7.615245644872038: 1, 14.808170938247452: 1, 30.77590436688
4003: 1, 12.567145956924024: 1, 13.451551110590305: 1, 21.433500350396358:
1, 20.05751415684878: 1, 19.229873037114125: 1, 9.0604145971415O8: 1, 7.68
9863188865266: 1, 39.43914525492003: 1, 22.87330854695533: 1, 8.5027884033
86127: 1, 14.29187230707388: 1, 17.3559787215466: 1, 16.873177716235222:
1, 31.021402062499572: 1, 14.32396143407691: 1, 11.371573000968807: 1, 10.
2315824253614: 1, 15.443379418953361: 1, 7.535517633551091: 1, 23.50294484
6620906: 1, 6.944191790577285: 1, 7.485449308316847: 1, 20.92209645987644
6: 1, 8.034922545735009: 1, 10.403994642683177: 1, 15.822350391231897: 1,
10.47043757597343: 1, 11.647832220791448: 1, 22.267889719360006: 1, 9.0125
14513983142: 1, 10.752599105689042: 1, 19.229461208375664: 1, 9.3073247834
89547: 1, 8.494562119905547: 1, 15.072304891410623: 1, 16.146338657672704:
1, 11.963462580318417: 1, 21.366379874332154: 1, 12.987867024050237: 1, 1
0.86477085605207: 1, 8.742844752592099: 1, 29.03848413673173: 1, 30.422205
405853063: 1, 8.555690898035913: 1, 9.37543028454696: 1, 6.28308951130898:
1, 16.483974636880923: 1, 8.978067321705401: 1, 11.645202363570766: 1, 7.9
04358833666922: 1, 26.3424558800851: 1, 17.74881420720489: 1, 9.5020777185
00644: 1, 14.53389930406728: 1, 21.105440105222225: 1, 12.769289249694673:
1, 13.497736100904026: 1, 26.40218729634249: 1, 9.44221844911052: 1, 10.92
4716964201556: 1, 22.888413149515728: 1, 18.55915375809837: 1, 11.94718957
300175: 1, 10.342003512746972: 1, 10.99711754100333: 1, 10.19250680007623
4: 1, 18.3905775822402: 1, 11.73244022437413: 1, 46.76379014641248: 1, 13.
900006150008712: 1, 14.54161171645372: 1, 11.659095448691813: 1, 10.606669
574227533: 1, 9.952465396086415: 1, 10.499905675008101: 1, 10.489073222668
392: 1, 7.996678525218392: 1, 14.368781756485323: 1, 15.207866004449196:
1, 19.870081494470828: 1, 11.69280836203719: 1, 15.210997525100103: 1, 9.6
55087507505948744: 1, 15.416755352417477: 1, 9.554850375865472: 1, 8.75782930
9049441: 1, 11.680664112063248: 1, 14.116340832256094: 1, 13.9276453605335
28: 1, 7.622127634398646: 1, 12.136692637582167: 1, 20.180331183473346: 1,
16.550717987700622: 1, 9.184867759681365: 1, 8.898649650369155: 1, 8.06450
7357656199: 1, 63.66486408212424: 1, 18.453071412221128: 1, 14.90957823089
867: 1, 19.076591809014964: 1, 17.160941373920462: 1, 12.309820274402334:
1, 9.212361630548292: 1, 21.213342669463376: 1, 8.771247727906848: 1, 9.88
0305158136887: 1, 6.813451065424341: 1, 21.16396492511301: 1, 9.7276010274
16664: 1, 14.666266367096304: 1, 9.52419434695436: 1, 10.51995745032787:
1, 21.942680723935847: 1, 8.948774926453238: 1, 13.306769299960045: 1, 15.
339721715957255: 1, 9.505875440985468: 1, 16.501837562251456: 1, 9.1845888
61271886: 1, 21.337056950887263: 1, 20.183514370171824: 1, 8.5148050103341
33: 1, 7.321884028650568: 1, 7.524706592986988: 1, 22.590367159670183: 1,
14.23585970091634: 1, 9.761391957459265: 1, 22.488757024505347: 1, 9.06366
4072943794: 1, 31.82774531257145: 1, 9.74465762880388: 1, 14.4826229082550
74: 1, 12.180026715910346: 1, 14.486257193686166: 1, 10.307081456677645:
1, 10.79852509016703: 1, 23.597827066243912: 1, 6.2627347821631325: 1, 10.
423664301860498: 1, 10.564091248922807: 1, 9.427747811743227: 1, 31.005830
180311303: 1, 8.938350397662367: 1, 11.170207941027082: 1, 32.651884109605
46: 1, 15.3830384353572: 1, 23.839913163317327: 1, 10.776827430669961: 1,
11.519446830696547: 1, 7.593642798186533: 1, 8.88836515962072: 1, 24.29129
9054056328: 1, 10.2067466641693: 1, 7.574961072396089: 1, 21.9302907175105
02: 1, 12.081201982000517: 1, 21.765255241774575: 1, 11.610247769787103:
1, 11.056689346441198: 1, 8.242423836011605: 1, 25.39452265010837: 1, 17.4
22477944610797: 1, 16.57609575788802: 1, 12.084139193738643: 1, 13.5292447
98941612: 1, 12.063258394337504: 1, 16.13894444185376: 1, 11.5775101850681
89: 1, 7.871396522387013: 1, 25.059148508130637: 1, 8.257844681668129: 1,
16.561880602401683: 1, 12.9640777349183: 1, 6.885422016288712: 1, 10.74582
4557181777: 1, 17.129925416764955: 1})

In [44]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
klearn.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='opt
imal', 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 Gradie
nt 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_, eps=1e-15
))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, lab
els=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 is:",log_lo
ss(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 is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2053685397104188
For values of alpha =  0.0001 The log loss is: 1.1787487453756782
For values of alpha =  0.001 The log loss is: 1.4121857684520862
For values of alpha =  0.01 The log loss is: 1.8028229289937225
For values of alpha =  0.1 The log loss is: 1.9749570690669598
For values of alpha =  1 The log loss is: 1.9676993370012519
```




```
For values of best alpha =  0.0001 The train log loss is: 0.80755083027715
21
For values of best alpha =  0.0001 The cross validation log loss is: 1.178
7487453756782
For values of best alpha =  0.0001 The test log loss is: 1.135543045903289
8
```

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

Ans. Yes, it seems like!

In [45]:

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

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

In [46]:

```
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 da
ta")
```

```
3.497 % of word of test data appeared in train data
3.861 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [47]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to e
ach 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 [48]:

```
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 [49]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

Stacking the three types of features

In [50]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_f
eature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feat
ure_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_on
ehotCoding))

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).to
csr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_vari
ation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variati
on_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fea
ture_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_r
esponseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_resp
onseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCo
ding))
```

In [51]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCo
ding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodi
ng.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_o
nehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3203)
(number of data points * number of features) in test data =  (665, 3203)
(number of data points * number of features) in cross validation data = (5
32, 3203)
```

In [52]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_response
Coding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCo
ding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_r
esponseCoding.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 = (5
32, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

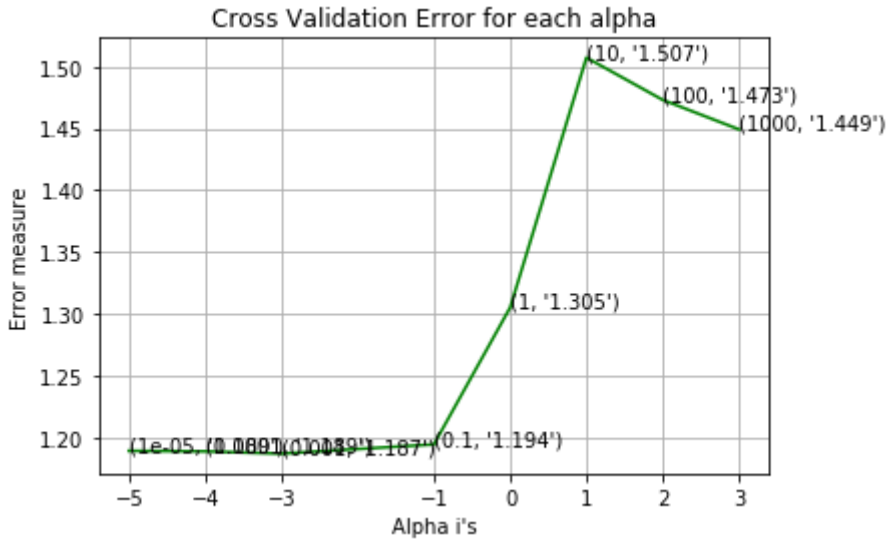
4.1.1.1. Hyper parameter tuning

In [53]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
odules/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/n
aive-bayes-algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
s/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/n
aive-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 est
imates
    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 is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.1892423135321055
for alpha = 0.0001
Log Loss : 1.1888455138224032
for alpha = 0.001
Log Loss : 1.1869792879456582
for alpha = 0.1
Log Loss : 1.194459118429091
for alpha = 1
Log Loss : 1.3052001373839295
for alpha = 10
Log Loss : 1.5072152003238395
for alpha = 100
Log Loss : 1.473049600463203
for alpha = 1000
Log Loss : 1.449181733255898
```




```
For values of best alpha =  0.001 The train log loss is: 0.440125231462607
5
For values of best alpha =  0.001 The cross validation log loss is: 1.1869
792879456582
For values of best alpha =  0.001 The test log loss is: 1.1868537699849748
```

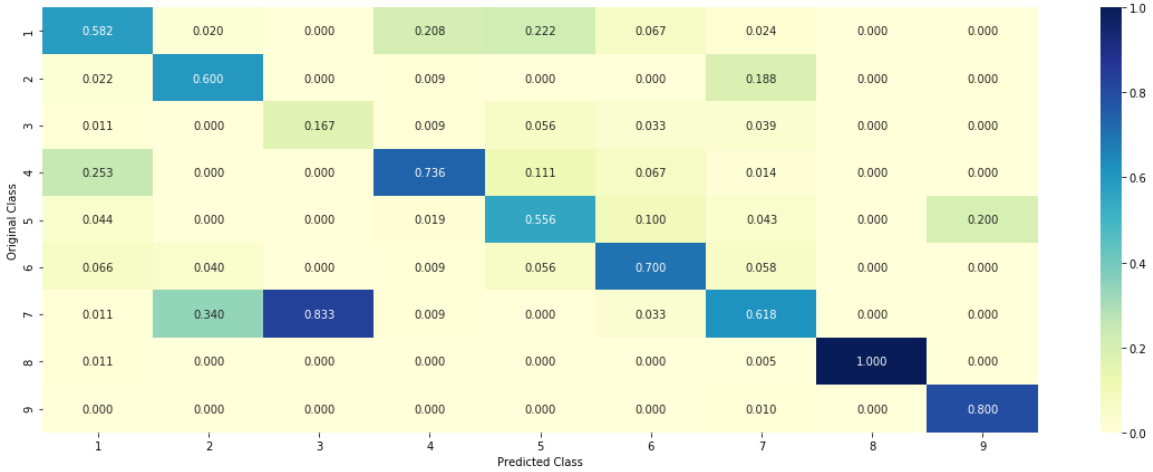
4.1.1.2. Testing the model with best hyper paramters

In [54]:

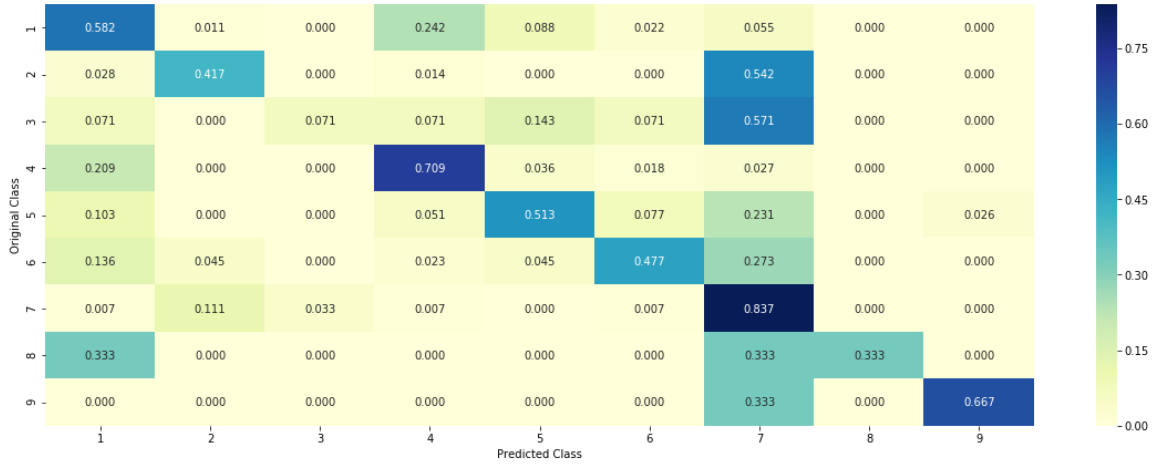
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
odules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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/n
aive-bayes-algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
s/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 estimat
es
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehot
Coding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.1869792879456582
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```


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




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



4.1.1.3. Sample test point-1

In [55]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0979 0.0618 0.0142 0.1592 0.5048 0.0469
0.1035 0.0064 0.0052]]
Actual Class : 5
--------------------------------------------------
7 Text feature [006] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.1.1.4. Sample test point-2

In [60]:

```
test_point_index = 200
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0562 0.0419 0.0095 0.7512 0.0308 0.0317
0.0707 0.0044 0.0036]]
Actual Class : 4
--------------------------------------------------
54 Text feature [105k] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.2. K Nearest Neighbour Classification

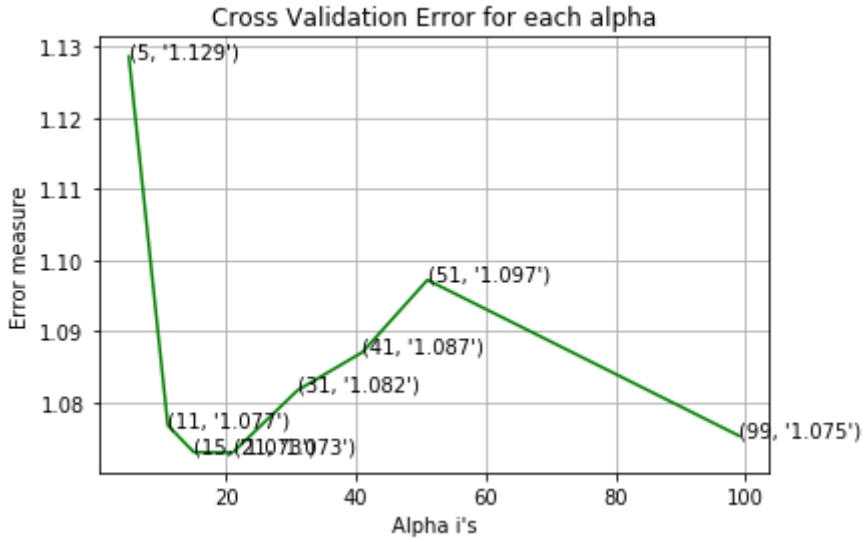
4.2.1. Hyper parameter tuning

In [61]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
nerated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=3
0, 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/module
s/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 est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for alpha = 5
Log Loss : 1.128641012167746
for alpha = 11
Log Loss : 1.0767743983588318
for alpha = 15
Log Loss : 1.0729951210381534
for alpha = 21
Log Loss : 1.072929872222681
for alpha = 31
Log Loss : 1.0817092785714417
for alpha = 41
Log Loss : 1.0870645610044765
for alpha = 51
Log Loss : 1.0972233241200589
for alpha = 99
Log Loss : 1.075301964242919
```



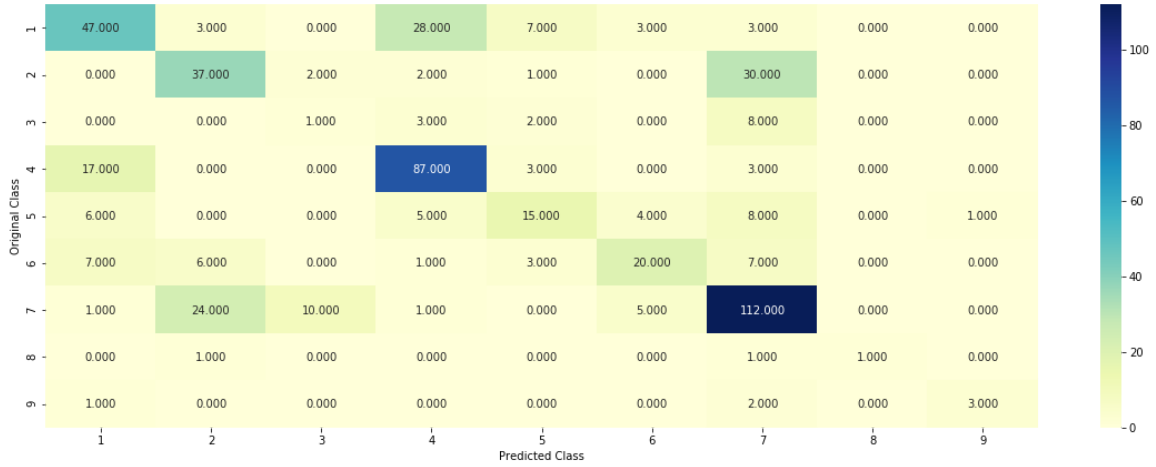

```
For values of best alpha =  21 The train log loss is: 0.7288740861448965
For values of best alpha =  21 The cross validation log loss is: 1.0729298
72222681
For values of best alpha =  21 The test log loss is: 1.0775671067460144
```

4.2.2. Testing the model with best hyper paramters

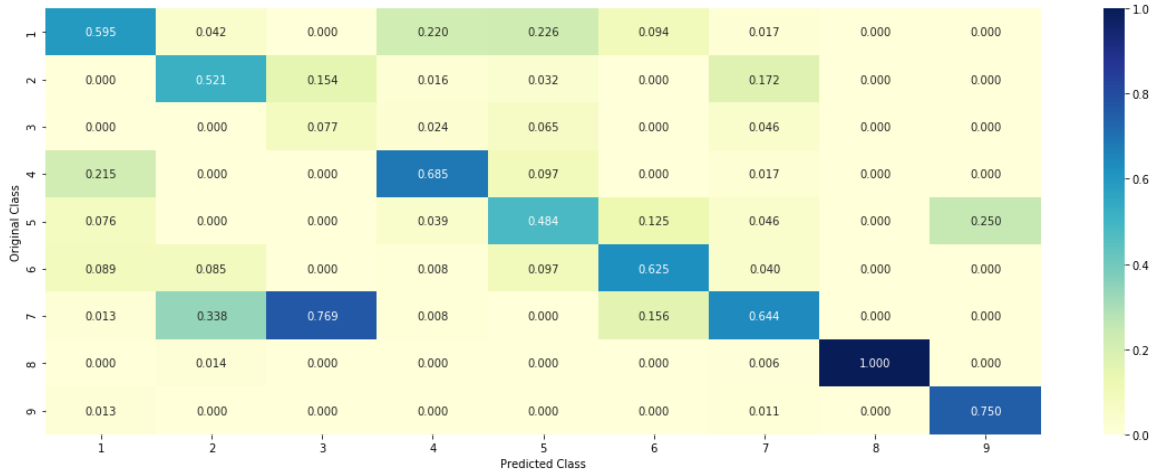
In [62]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
nerated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=3
0, 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 : 1.072929872222681
Number of mis-classified points : 0.39285714285714285
-------------------- Confusion matrix --------------------
```




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




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

4.2.3.Sample test point-1

In [63]:

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

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

```
Predicted Class : 1
Actual Class : 5
The  21  nearest neighbours of the test points belongs to classes [3 5 5 1
4 5 4 4 4 4 4 4 4 4 5 5 1 5 5 6 4]
Fequency of nearest points : Counter({4: 10, 5: 7, 1: 2, 3: 1, 6: 1})
```

4.2.4. Sample test point-2

In [64]:

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

test_point_index = 100

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

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 21 and the nearest neighbours of the test points be
longs to classes [4 4 4 4 3 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 4]
Fequency of nearest points : Counter({4: 19, 3: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

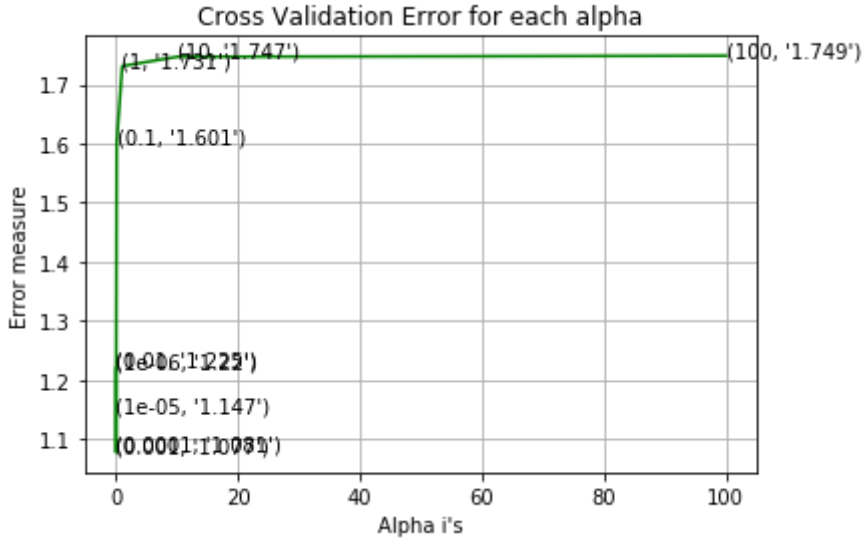
4.3.1.1. Hyper paramter tuning

In [65]:

```
# 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', los
s='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.2197709841653004
for alpha = 1e-05
Log Loss : 1.1469313299767185
for alpha = 0.0001
Log Loss : 1.081370256919392
for alpha = 0.001
Log Loss : 1.0773308429020143
for alpha = 0.01
Log Loss : 1.2247482885916166
for alpha = 0.1
Log Loss : 1.601221437687427
for alpha = 1
Log Loss : 1.731114207984259
for alpha = 10
Log Loss : 1.746847383344125
for alpha = 100
Log Loss : 1.748683704177787
```



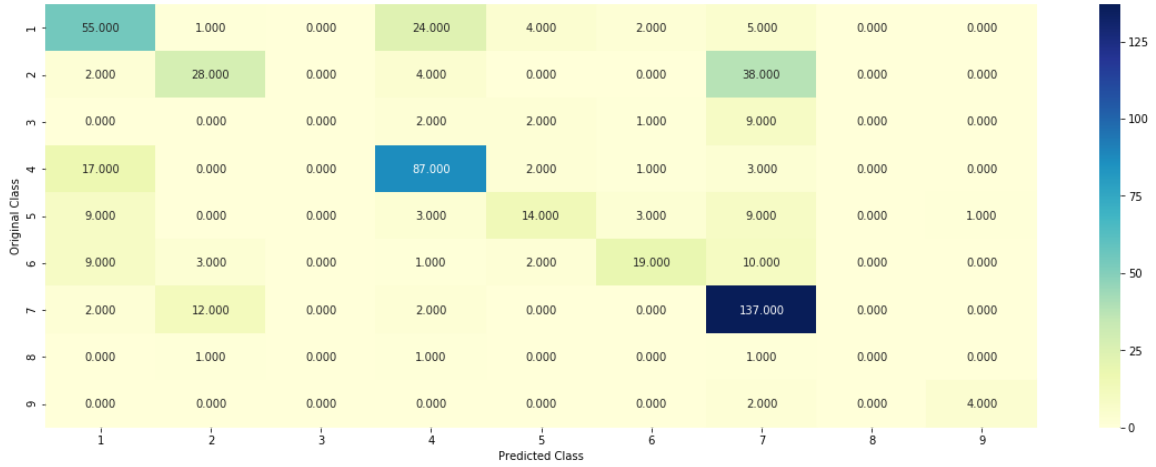

```
For values of best alpha =  0.001 The train log loss is: 0.722518358438219
4
For values of best alpha =  0.001 The cross validation log loss is: 1.0773
308429020143
For values of best alpha =  0.001 The test log loss is: 1.0235813310366604
```

4.3.1.2. Testing the model with best hyper paramters

In [66]:

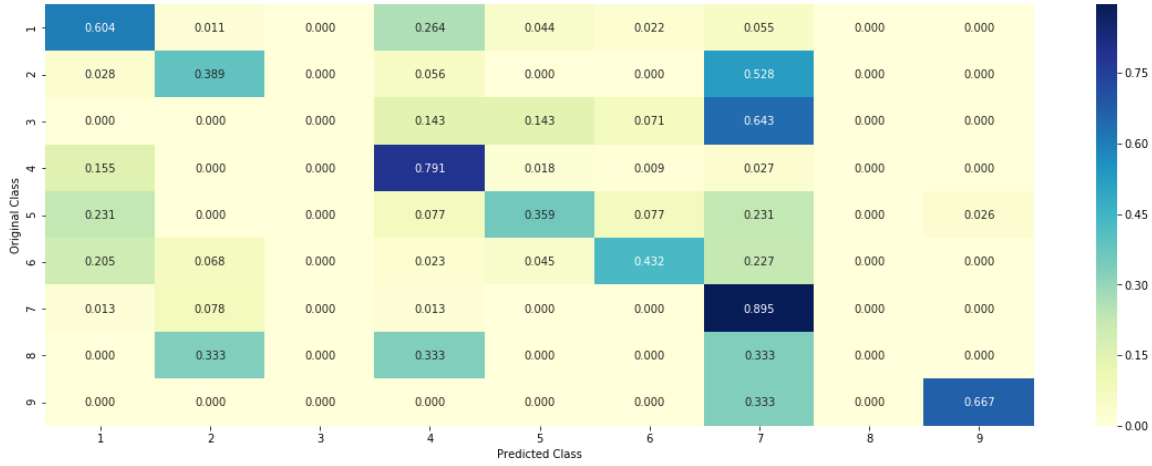
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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 : 1.0773308429020143
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




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


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



4.3.1.3. Feature Importance

In [67]:

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

4.3.1.3.1. Sample test point-1

In [69]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0173 0.0071 0.0211 0.9261 0.0096 0.0071
0.0057 0.0048 0.0012]]
Actual Class : 4
--------------------------------------------------
438 Text feature [131] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.1.3.2. Sample test point-2

In [68]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1295 0.0247 0.0335 0.4001 0.3683 0.025
0.0097 0.0069 0.0023]]
Actual Class : 5
--------------------------------------------------
32 Text feature [128] present in test data point [True]
443 Text feature [117] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.2. Without Class balancing

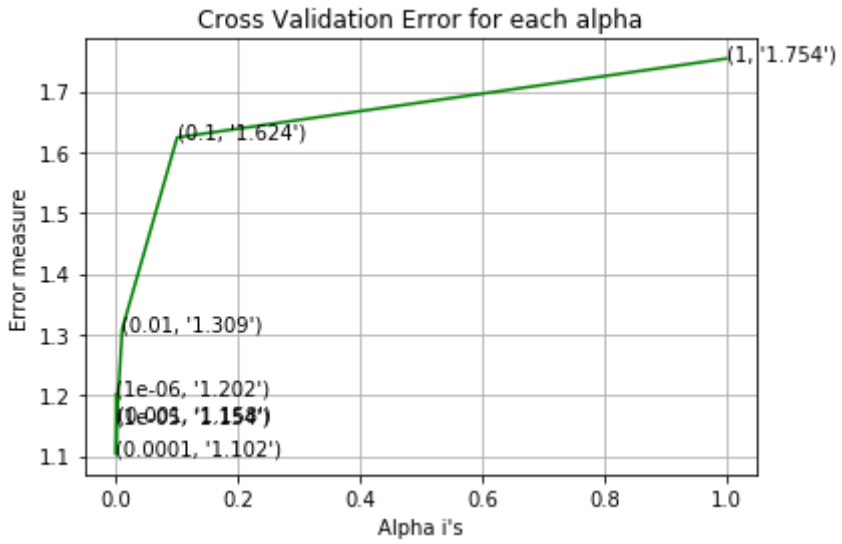
4.3.2.1. Hyper paramter tuning

In [70]:

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

```
for alpha = 1e-06
Log Loss : 1.201679121975529
for alpha = 1e-05
Log Loss : 1.1541799032895455
for alpha = 0.0001
Log Loss : 1.1020536706067718
for alpha = 0.001
Log Loss : 1.1577999840006932
for alpha = 0.01
Log Loss : 1.3088707829563073
for alpha = 0.1
Log Loss : 1.6241581404580299
for alpha = 1
Log Loss : 1.75393232456892
```



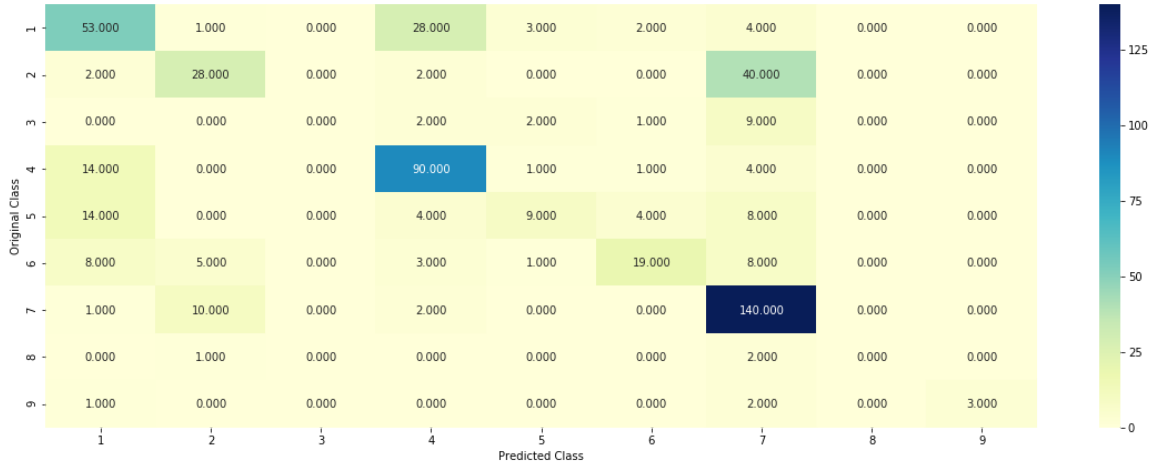

```
For values of best alpha =  0.0001 The train log loss is: 0.38895488897287
89
For values of best alpha =  0.0001 The cross validation log loss is: 1.102
0536706067718
For values of best alpha =  0.0001 The test log loss is: 0.977895765385327
7
```

4.3.2.2. Testing model with best hyper parameters

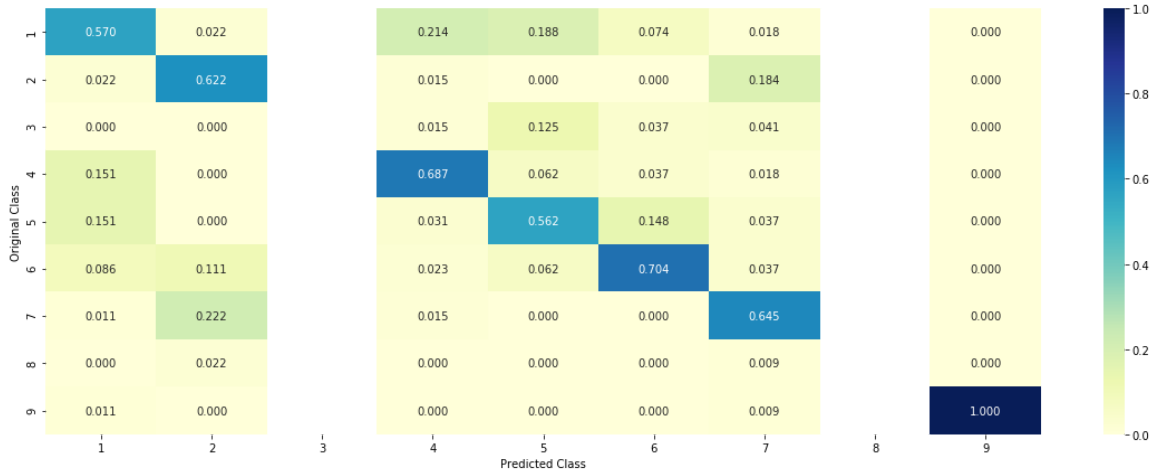
In [71]:

```
# 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)
```

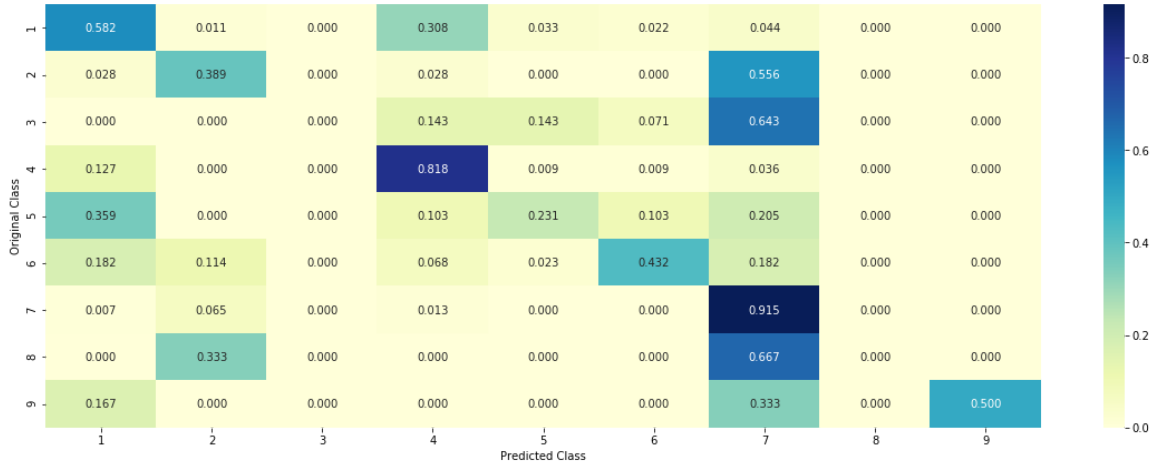
```
Log loss : 1.1020536706067718
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Sample test point-1

In [72]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1263 0.0151 0.029  0.2977 0.5095 0.0111
0.0051 0.0052 0.0011]]
Actual Class : 5
--------------------------------------------------
177 Text feature [006] present in test data point [True]
329 Text feature [111] present in test data point [True]
402 Text feature [13] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2.4. Sample test point-2

In [75]:

```
test_point_index = 250
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[2.170e-02 3.300e-03 9.600e-03 8.470e-02
1.980e-02 8.535e-01 1.900e-03
  4.800e-03 7.000e-04]]
Actual Class : 6
--------------------------------------------------
232 Text feature [100] present in test data point [True]
278 Text feature [119] present in test data point [True]
363 Text feature [13] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.4. Linear Support Vector Machines

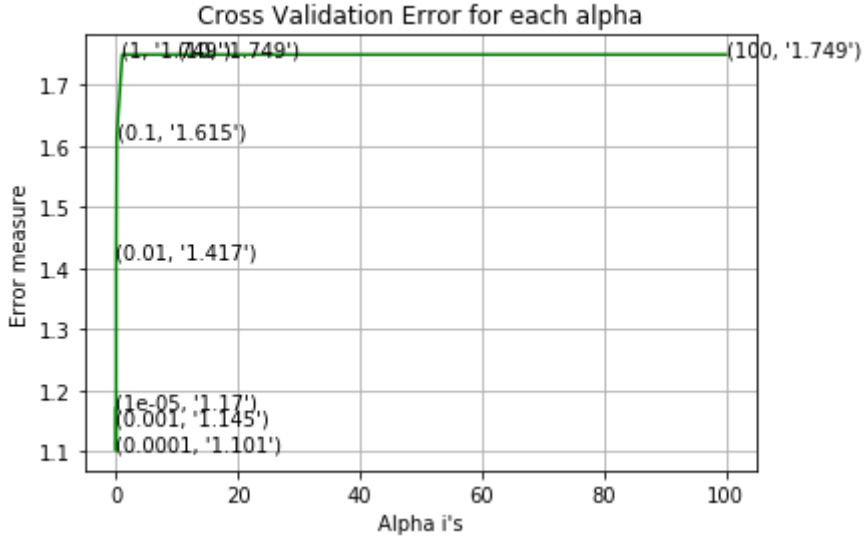
4.4.1. Hyper paramter tuning

In [76]:

```
# 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, probabili
ty=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap
e='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/m
athematical-derivation-copy-8/
# --------------------------------



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge',
random_state=42)
    clf.fit(train_x_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', los
s='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 is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
 is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.1701244987863386
for C = 0.0001
Log Loss : 1.1014642836387338
for C = 0.001
Log Loss : 1.1453017985868903
for C = 0.01
Log Loss : 1.4172082849011838
for C = 0.1
Log Loss : 1.6150773465487553
for C = 1
Log Loss : 1.7491520446791677
for C = 10
Log Loss : 1.7491614856574755
for C = 100
Log Loss : 1.7491577374818599
```




```
For values of best alpha =  0.0001 The train log loss is: 0.31514633304460
62
For values of best alpha =  0.0001 The cross validation log loss is: 1.101
4642836387338
For values of best alpha =  0.0001 The test log loss is: 0.986018361141826
```

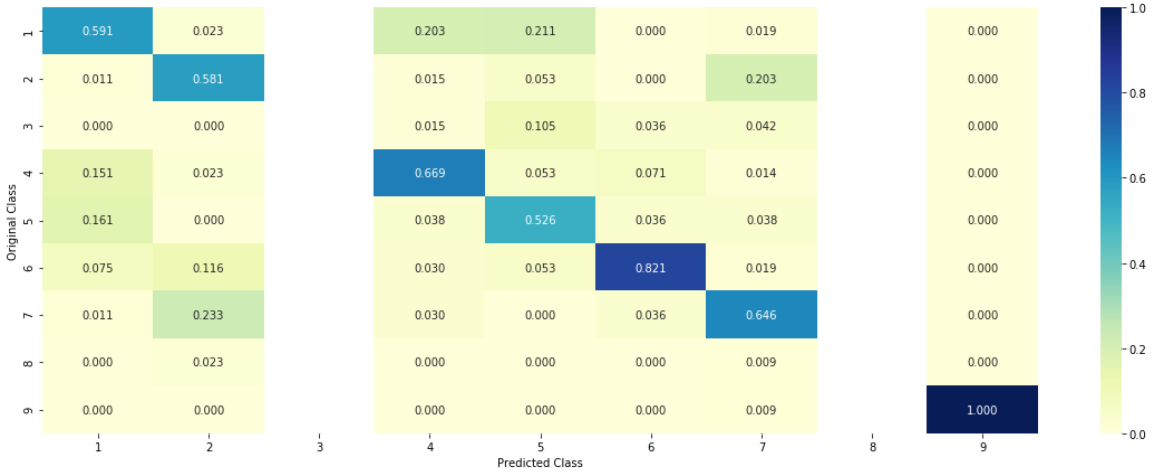
4.4.2. Testing model with best hyper parameters

In [77]:

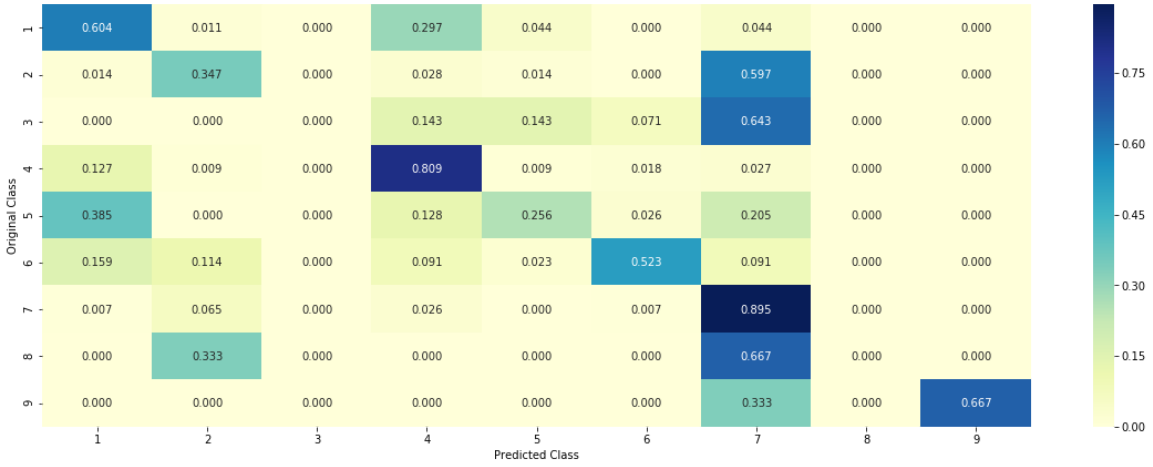
```
# read more about support vector machines with linear kernals here http://scikit-learn.
org/stable/modules/generated/sklearn.svm.SVC.html

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
ty=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap
e='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/m
athematical-derivation-copy-8/
# ---------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balance
d')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
2,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y,
clf)
```

```
Log loss : 1.1014642836387338
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```


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




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



4.3.3. Feature Importance

4.3.3.1. Sample test point-1

In [78]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1321 0.0396 0.0373 0.2564 0.4893 0.012
0.0257 0.0055 0.002 ]]
Actual Class : 5
--------------------------------------------------
178 Text feature [006] present in test data point [True]
244 Text feature [111] present in test data point [True]
405 Text feature [12] present in test data point [True]
423 Text feature [13] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.3.2. Sample test point-2

In [79]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0748 0.024  0.0309 0.78   0.041  0.0175
0.0242 0.0044 0.0032]]
Actual Class : 4
--------------------------------------------------
356 Text feature [131] present in test data point [True]
396 Text feature [123] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [80]:

```
# ---------------------------------
# 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=Non
e, 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/r
andom-forest-and-their-construction-2/
# ---------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
s/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, ran
dom_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_, ep
s=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
```

```
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_err
or_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', m
ax_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

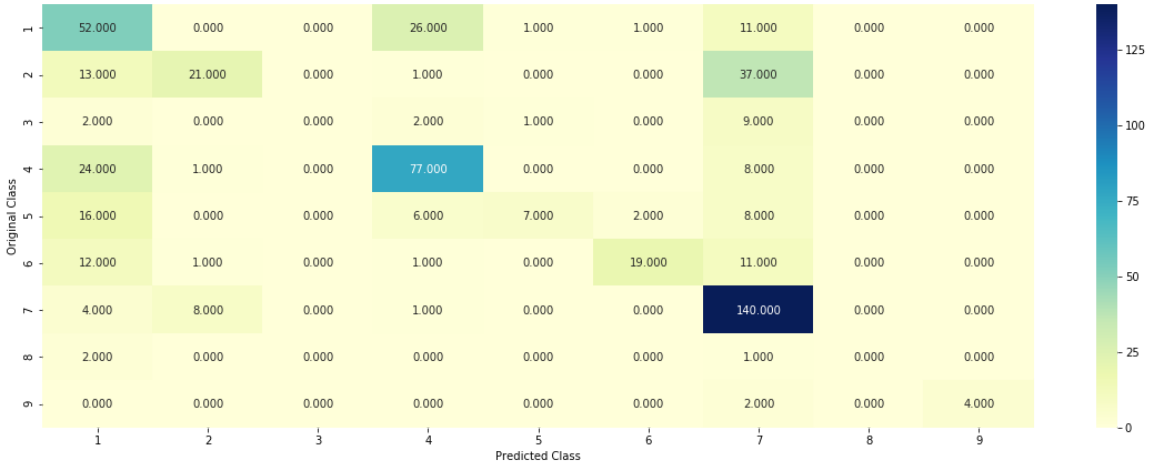
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1896322242651554
for n_estimators = 100 and max depth =  10
Log Loss : 1.2331864818450227
for n_estimators = 200 and max depth =  5
Log Loss : 1.1817589232735966
for n_estimators = 200 and max depth =  10
Log Loss : 1.2231018441947756
for n_estimators = 500 and max depth =  5
Log Loss : 1.1790152277173065
for n_estimators = 500 and max depth =  10
Log Loss : 1.2141027878816948
for n_estimators = 1000 and max depth =  5
Log Loss : 1.176279710482046
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2114806913875407
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1743230933220627
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2102482061532853
For values of best estimator =  2000 The train log loss is: 0.855496587620
7795
For values of best estimator =  2000 The cross validation log loss is: 1.1
743230933220627
For values of best estimator =  2000 The test log loss is: 1.1915722519162
113
```

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

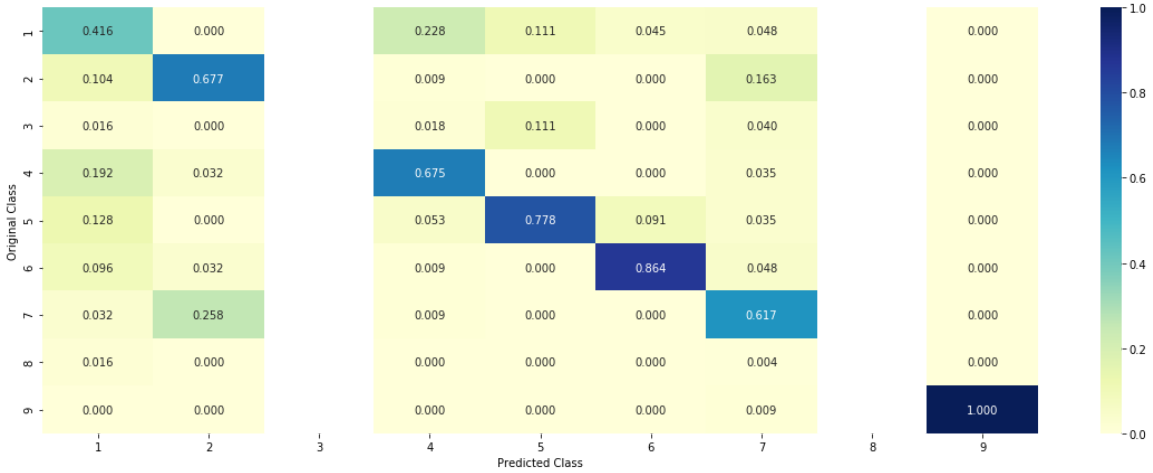
In [81]:

```
# ---------------------------------
# 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=Non
e, 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/r
andom-forest-and-their-construction-2/
# ---------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', m
ax_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)
```

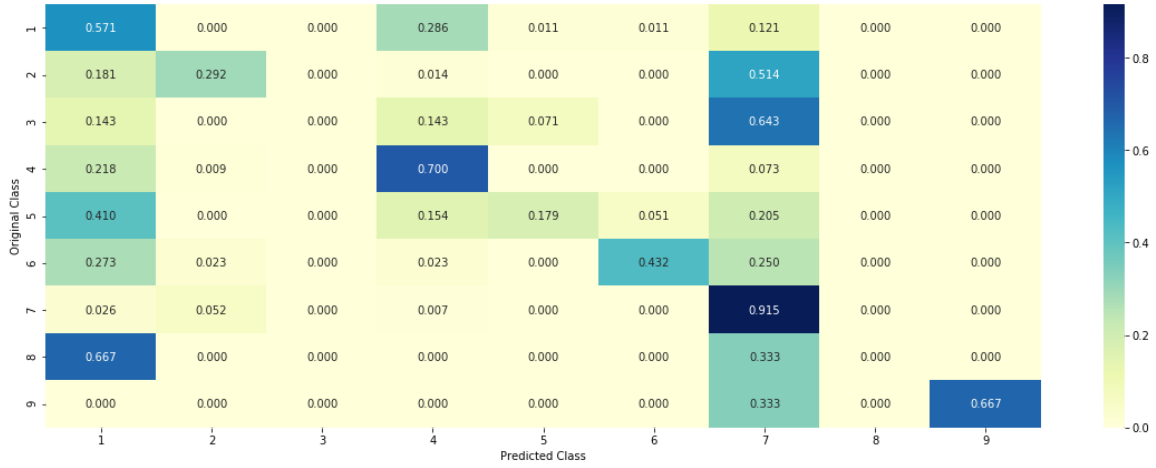
```
Log loss : 1.1743230933220627
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```




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




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



4.5.3. Feature Importance

4.5.3.1. Sample test point-1

In [84]:

```
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_onehotCod
ing[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_index],test_
df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_featu
re)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0874 0.0124 0.0323 0.7671 0.036  0.0299
0.0272 0.0036 0.0042]]
Actual Class : 4
--------------------------------------------------
12 Text feature [12er] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3.2. Sample test point-2

In [82]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[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_index],test_
df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_featu
re)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3226 0.0114 0.0255 0.3828 0.1388 0.0922
0.0175 0.0041 0.0051]]
Actual Class : 5
--------------------------------------------------
11 Text feature [111] present in test data point [True]
53 Text feature [110] present in test data point [True]
98 Text feature [006] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [85]:

```
# ---------------------------------
# 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=Non
e, 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/r
andom-forest-and-their-construction-2/
# ---------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
s/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, ran
dom_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_, ep
s=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
```

```
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_err
or_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', m
ax_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.135719941414764
for n_estimators = 10 and max depth =  3
Log Loss : 1.6952090436997127
for n_estimators = 10 and max depth =  5
Log Loss : 1.2912647350061037
for n_estimators = 10 and max depth =  10
Log Loss : 1.8441383577519639
for n_estimators = 50 and max depth =  2
Log Loss : 1.6309521165210845
for n_estimators = 50 and max depth =  3
Log Loss : 1.3985378271377833
for n_estimators = 50 and max depth =  5
Log Loss : 1.3234222745151694
for n_estimators = 50 and max depth =  10
Log Loss : 1.735982924562293
for n_estimators = 100 and max depth =  2
Log Loss : 1.5504058942913166
for n_estimators = 100 and max depth =  3
Log Loss : 1.3809954502419888
for n_estimators = 100 and max depth =  5
Log Loss : 1.2875391102083167
for n_estimators = 100 and max depth =  10
Log Loss : 1.6959955349374796
for n_estimators = 200 and max depth =  2
Log Loss : 1.587377950927457
for n_estimators = 200 and max depth =  3
Log Loss : 1.4225345207843514
for n_estimators = 200 and max depth =  5
Log Loss : 1.3619498294651056
for n_estimators = 200 and max depth =  10
Log Loss : 1.6895718253482075
for n_estimators = 500 and max depth =  2
Log Loss : 1.6446868480116386
for n_estimators = 500 and max depth =  3
Log Loss : 1.473784906480509
for n_estimators = 500 and max depth =  5
Log Loss : 1.3573753517015095
for n_estimators = 500 and max depth =  10
Log Loss : 1.7773759488835714
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6333482356631093
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4985573118447633
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3619191821541292
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7832816517917296
For values of best alpha =  100 The train log loss is: 0.06240244921619527
6
For values of best alpha =  100 The cross validation log loss is: 1.287539
110208317
For values of best alpha =  100 The test log loss is: 1.230829755337582
```

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

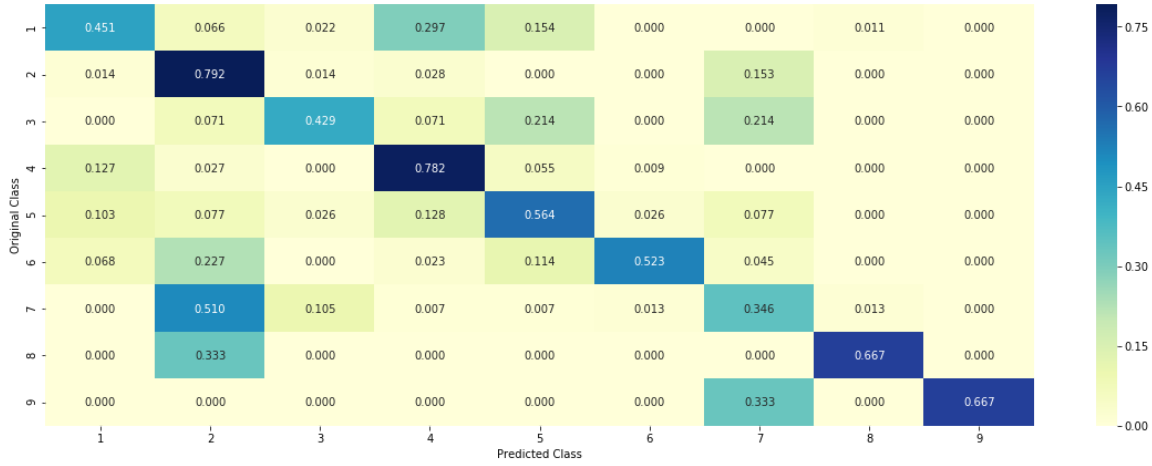
In [86]:

```
# ---------------------------------
# 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=Non
e, 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/r
andom-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,c
v_y, clf)
```

```
Log loss : 1.287539110208317
Number of mis-classified points : 0.4473684210526316
-------------------- Confusion matrix --------------------
```


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


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



4.5.5. Feature Importance

4.5.5.1. Sample test point-1

In [87]:

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


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseC
oding[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: [[0.0949 0.0157 0.1203 0.1824 0.4448 0.1011
0.0076 0.0169 0.0163]]
Actual Class : 5
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Sample test point-2

In [88]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseC
oding[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: [[0.0725 0.0305 0.2972 0.4554 0.0314 0.0415
0.0116 0.0341 0.0258]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 Testing with hyper parameter tuning

In [89]:

```
# 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
```

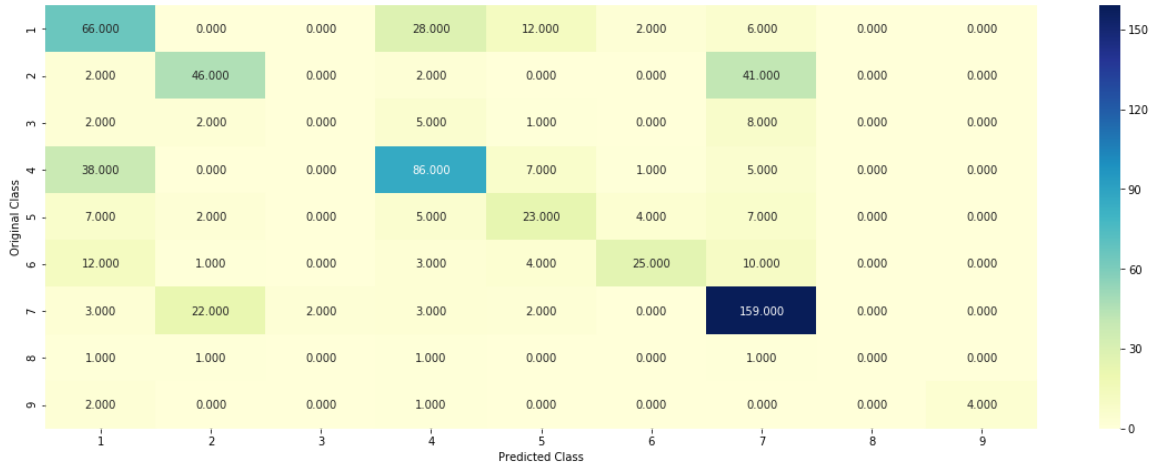
```
Logistic Regression :  Log Loss: 1.08
Support vector machines : Log Loss: 1.75
Naive Bayes : Log Loss: 1.19
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.816
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.707
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.307
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.241
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.588
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.906
```

4.7.2 Testing the model with the best hyper parameters

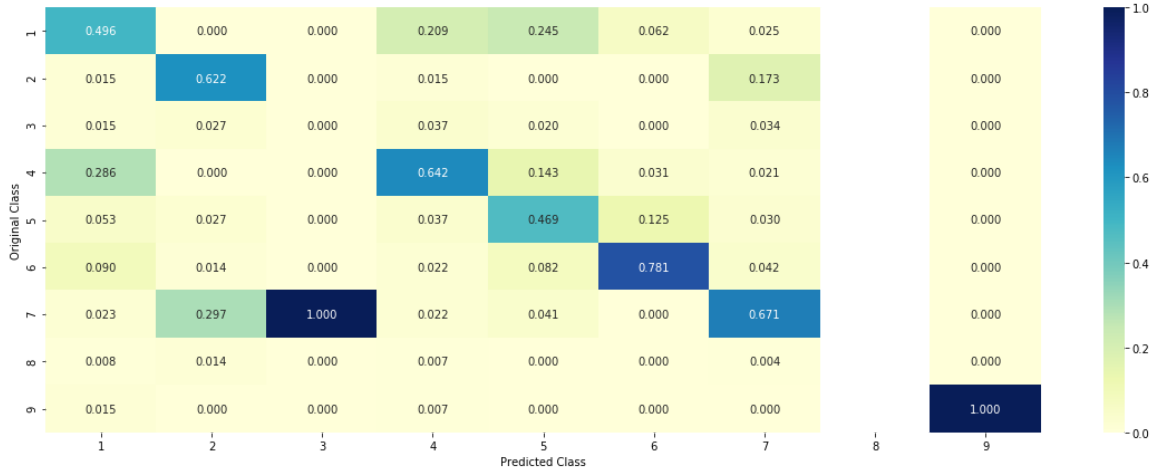
In [90]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
r, 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_onehotC
oding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.33631170574631286
Log loss (CV) on the stacking classifier : 1.2407099874013656
Log loss (test) on the stacking classifier : 1.2339319317296902
Number of missclassified point : 0.3849624060150376
-------------------- Confusion matrix --------------------
```




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



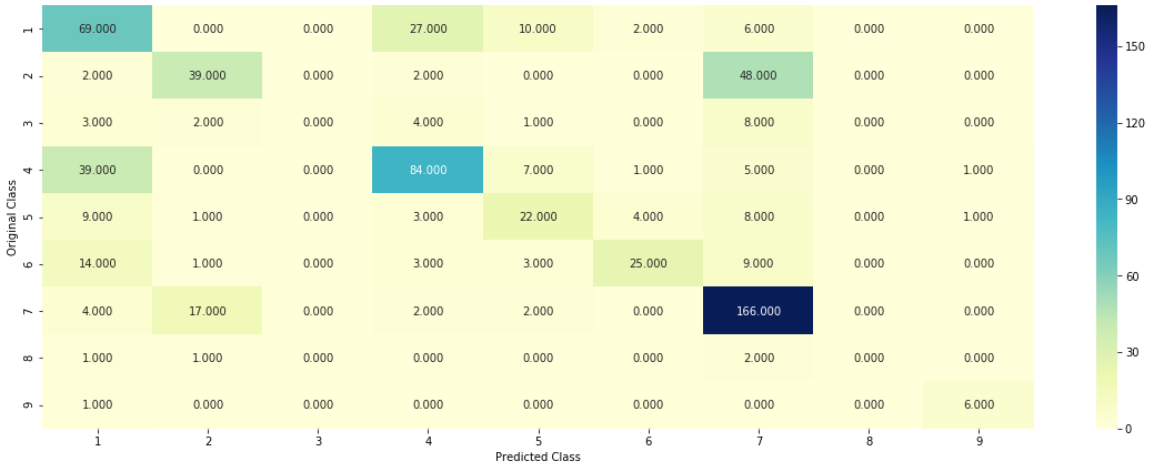

```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.7.3 Maximum Voting classifier

In [91]:

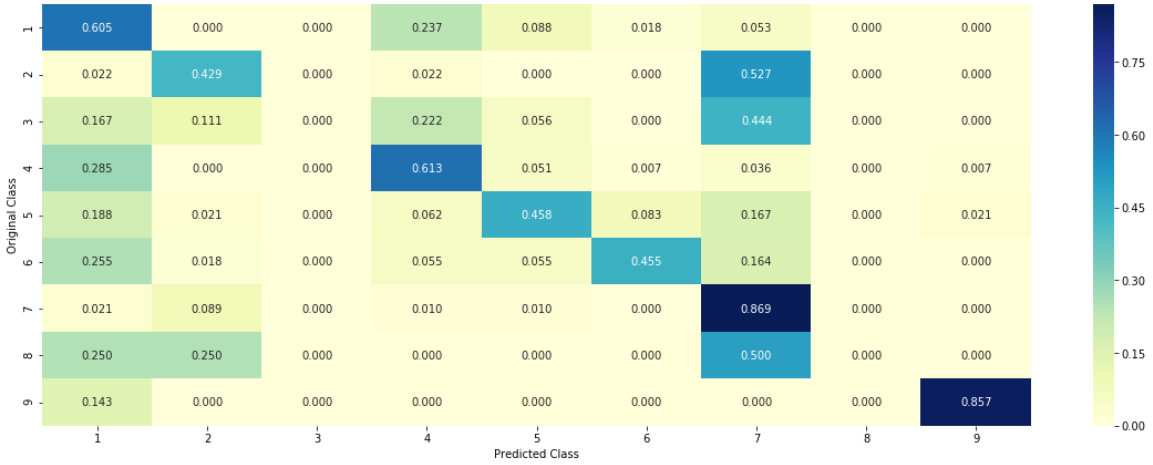
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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 : 0.7906274207600275
Log loss (CV) on the VotingClassifier : 1.1963991261956455
Log loss (test) on the VotingClassifier : 1.1833961223580824
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```




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


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



Logistic regression with unigrams and bigrams in CountVectorizer

1.0 Gene

In [101]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
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 [102]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encodin
g method. The shape of gene feature: (2124, 238)
```

2.0 Variation

In [105]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Var
iation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
n'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [106]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot en
coding method. The shape of Variation feature:", train_variation_feature_onehotCoding.s
hape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-
hot encoding method. The shape of Variation feature: (2124, 2047)
```

3.0 Text

In [107]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train d
ata
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1, 2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numb
er 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 : 781937
```

In [108]:

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

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

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

In [112]:

```
train_text_feature_onehotCoding.shape
```

Out[112]:

```
(2124, 781937)
```

4.0 Data preparation

In [109]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to e
ach 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 [110]:

```
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 [111]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

5.0 Stacking the three types of features

In [113]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_f
eature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feat
ure_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_on
ehotCoding))

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).to
csr()
cv_y = np.array(list(cv_df['Class']))
```

In [114]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCo
ding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodi
ng.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_o
nehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 78422
2)
(number of data points * number of features) in test data =  (665, 784222)
(number of data points * number of features) in cross validation data = (5
32, 784222)
```

6.0 LR Model

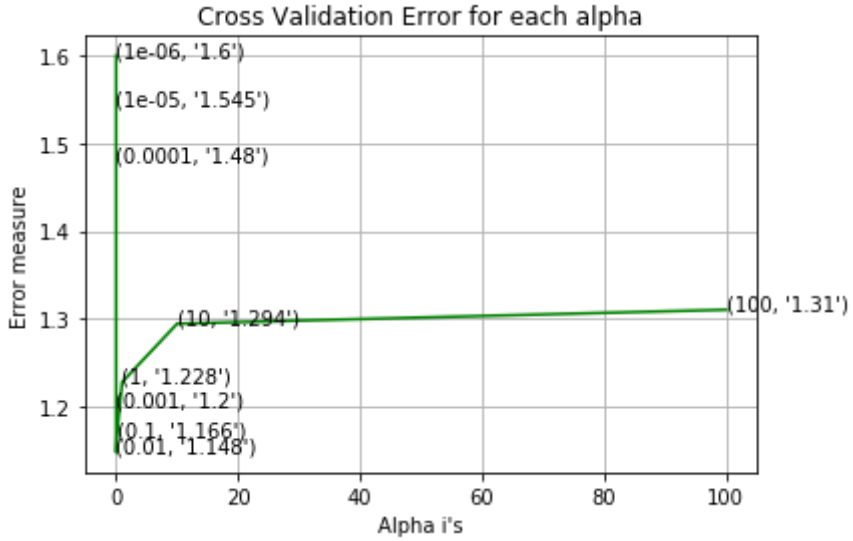
6.1 With Class balancing

6.1.1 Hyper paramter tuning

In [115]:

```
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', ran
dom_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    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 est
imates
    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', los
s='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.59978046258566
for alpha = 1e-05
Log Loss : 1.5447879750284637
for alpha = 0.0001
Log Loss : 1.4801426053801607
for alpha = 0.001
Log Loss : 1.2004429222450606
for alpha = 0.01
Log Loss : 1.1476165499437228
for alpha = 0.1
Log Loss : 1.165743882319161
for alpha = 1
Log Loss : 1.2279530896471464
for alpha = 10
Log Loss : 1.2943417540618667
for alpha = 100
Log Loss : 1.3102782398334656
```



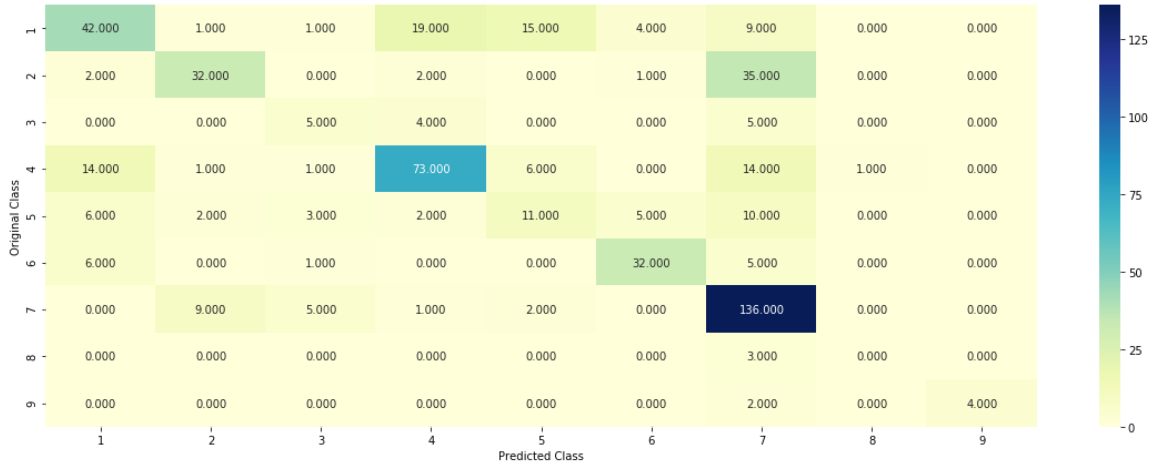

```
For values of best alpha =  0.01 The train log loss is: 0.7263982022306168
For values of best alpha =  0.01 The cross validation log loss is: 1.14761
65499437228
For values of best alpha =  0.01 The test log loss is: 1.1582218291350088
```

6.1.2 Testing the model with best hyper paramters

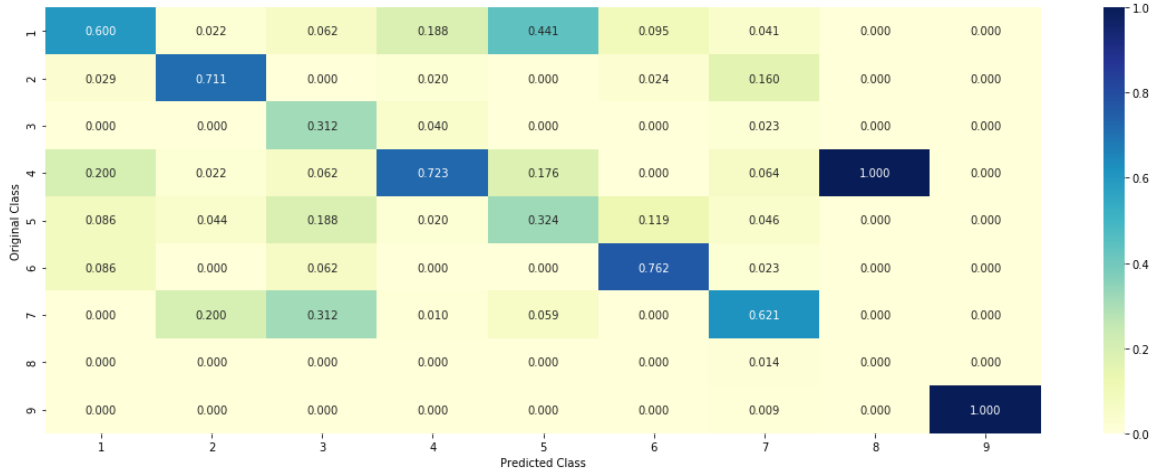
In [116]:

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

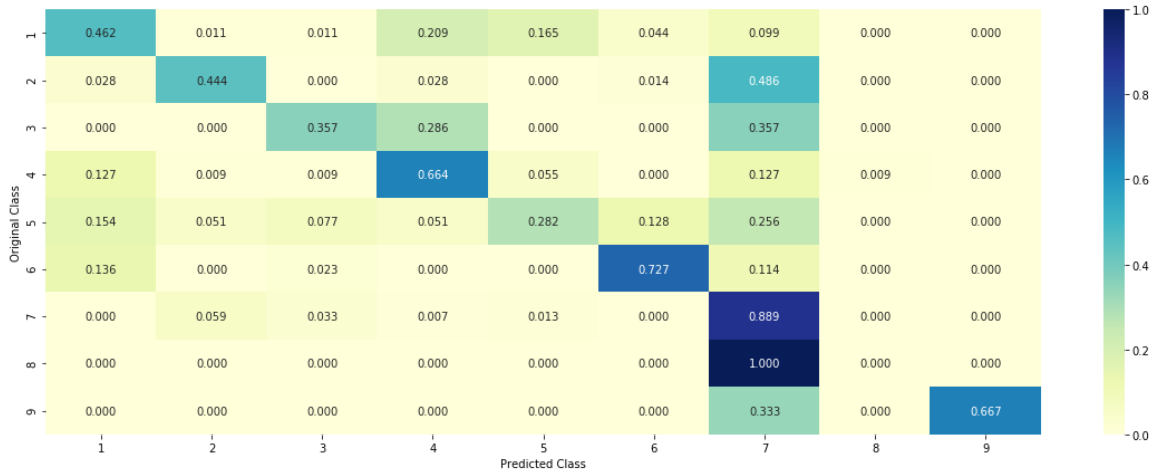
```
Log loss : 1.1476165499437228
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




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




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



6.1.3 Feature Importance

In [117]:

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

6.1.4 Sample test point-1

In [95]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[2.800e-03 8.766e-01 7.000e-04 7.000e-04
3.200e-03 9.000e-04 1.075e-01
  7.500e-03 0.000e+00]]
Actual Class : 2
--------------------------------------------------
59 Text feature [able] present in test data point [True]
115 Text feature [79] present in test data point [True]
120 Text feature [adjust] present in test data point [True]
124 Text feature [active] present in test data point [True]
137 Text feature [9b] present in test data point [True]
200 Text feature [85] present in test data point [True]
325 Text feature [072] present in test data point [True]
396 Text feature [32] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

6.1.5 Sample test point-2

In [96]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.2859 0.1587 0.0086 0.0195 0.3296 0.1437
0.0507 0.0029 0.0004]]
Actual Class : 6
--------------------------------------------------
58 Text feature [90] present in test data point [True]
243 Text feature [aberrations] present in test data point [True]
387 Text feature [41] present in test data point [True]
458 Text feature [101] present in test data point [True]
468 Text feature [17q25] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

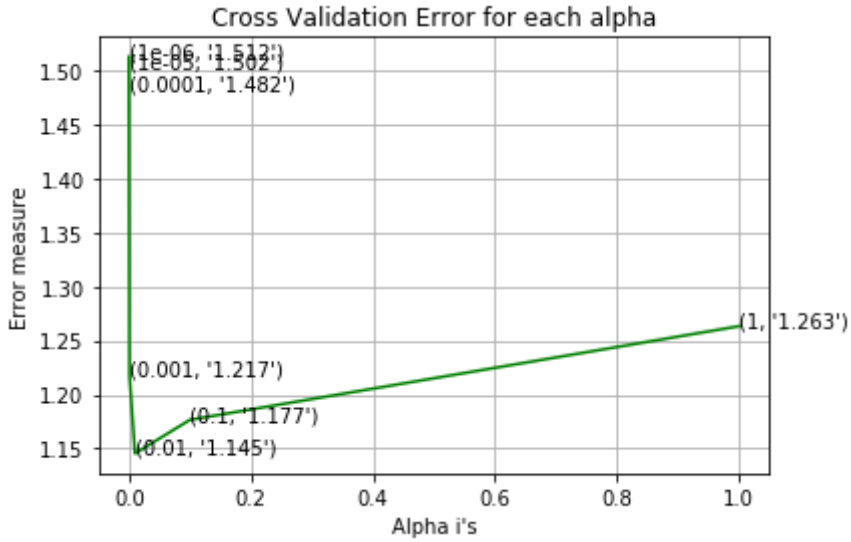
6.2 Without Class balancing

6.2.1 Hyper paramter tuning

In [121]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.5124463492005464
for alpha = 1e-05
Log Loss : 1.502278336902223
for alpha = 0.0001
Log Loss : 1.4822747655604693
for alpha = 0.001
Log Loss : 1.2174743568679418
for alpha = 0.01
Log Loss : 1.145449045638047
for alpha = 0.1
Log Loss : 1.1765556200164762
for alpha = 1
Log Loss : 1.2631973476797616
```



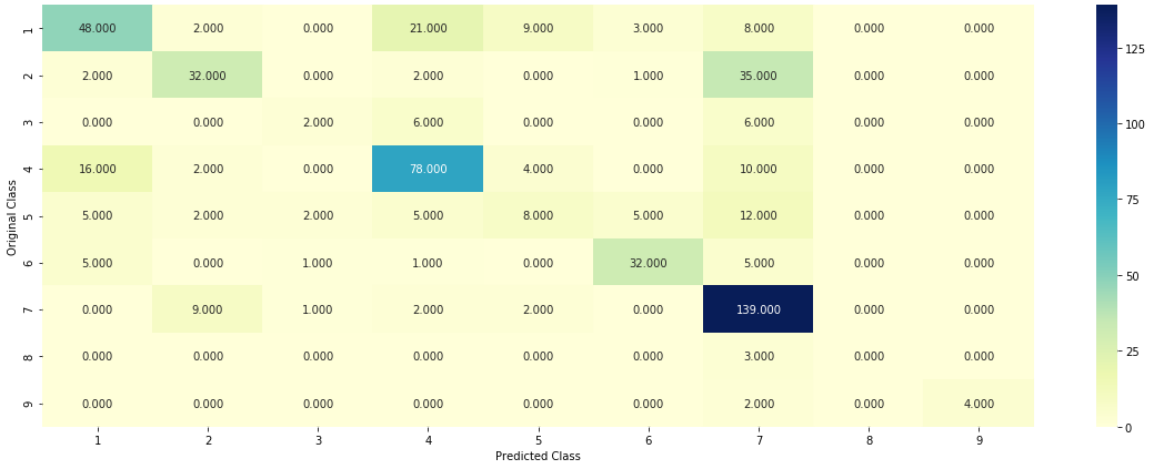

```
For values of best alpha =  0.01 The train log loss is: 0.7259842278904182
For values of best alpha =  0.01 The cross validation log loss is: 1.14544
9045638047
For values of best alpha =  0.01 The test log loss is: 1.1729505870159953
```

6.2.2 Testing model with best hyper parameters

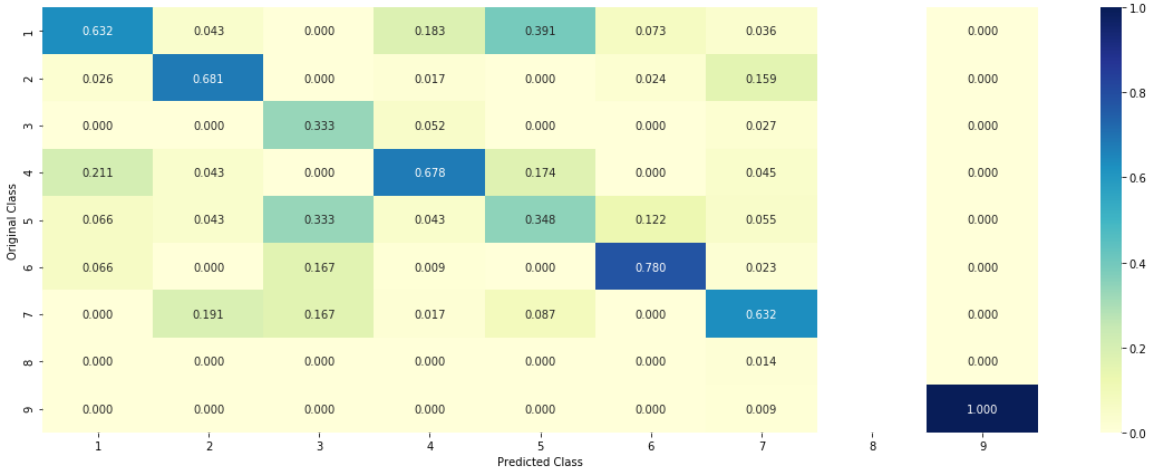
In [122]:

```
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)
```

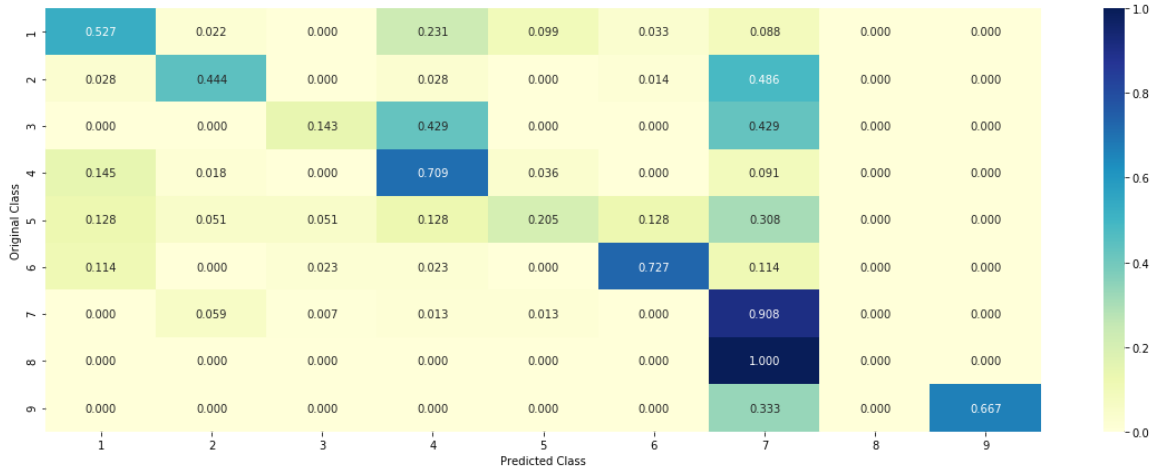
```
Log loss : 1.145449045638047
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```




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




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



6.2.3 Sample test point-1</h4>

In [99]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[3.200e-03 8.483e-01 4.000e-04 8.000e-04
2.800e-03 9.000e-04 1.425e-01
  1.000e-03 0.000e+00]]
Actual Class : 2
--------------------------------------------------
44 Text feature [able] present in test data point [True]
117 Text feature [79] present in test data point [True]
122 Text feature [active] present in test data point [True]
125 Text feature [adjust] present in test data point [True]
132 Text feature [9b] present in test data point [True]
213 Text feature [85] present in test data point [True]
315 Text feature [072] present in test data point [True]
389 Text feature [32] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.2.4. Sample test point-2

In [100]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[2.917e-01 1.607e-01 6.600e-03 1.740e-02
3.083e-01 1.425e-01 6.860e-02
  4.000e-03 2.000e-04]]
Actual Class : 6
--------------------------------------------------
42 Text feature [90] present in test data point [True]
260 Text feature [aberrations] present in test data point [True]
381 Text feature [41] present in test data point [True]
482 Text feature [101] present in test data point [True]
486 Text feature [17q25] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

Additional feature engineering

1.0 Data

In [1]:

```
import pandas as pd
result= pd.read_csv('result.csv')
```

In [3]:

```
result.head(5)
```

Out[3]:

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

2.0 Feature Engineering

2.1 Creating a feature that combines Gene & Variation

In [4]:

```
#combining gene & variation into a single feature
result['Gene+Variation']= result['Gene']+" "+result['Variation']
```

In [5]:

```
result.head(5)
```

Out[5]:

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

2.2 Creating a feature with word count in each text

In [6]:

```
# Creating feature 'TEXT_wordcount' which consists of number of words in each text
count=[len(row.split()) for row in result.TEXT.values]
```

In [7]:

```
print(len(count))
```

```
3321
```

In [8]:

```
result['TEXT_wordcount']=count
```

In [9]:

```
result.head(5)
```

Out[9]:

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

3.0 Splitting the data

In [10]:

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

# split the data into test and train by maintaining same distribution of output varaibl
e 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, t
est_size=0.2)
# split the train data into train and cross validation by maintaining same distribution
of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, t
est_size=0.2)

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
```

4.0 Encoding gene, variation & text features

4.1 Gene

In [74]:

```
# 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 [75]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encodin
g method. The shape of gene feature: (2124, 238)
```

4.2 Variation

In [76]:

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

In [77]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot en
coding method. The shape of Variation feature:", train_variation_feature_onehotCoding.s
hape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-
hot encoding method. The shape of Variation feature: (2124, 1947)
```

4.3 Text

In [86]:

```
# building a CountVectorizer with all the words that occured minimum 10 times in train
 data
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numb
er 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 : 5000
```

In [87]:

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

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

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

In [88]:

```
train_text_feature_onehotCoding.shape
```

Out[88]:

```
(2124, 5000)
```

5.0 Univariate analysis of the engineered features

5.1 Gene+Variation

In [18]:

```
comb=[len(row.split()) for row in result['Gene+Variation'].values]
```

In [19]:

```
print(max(comb))
```

```
2
```

- From the above cell, since max len of any row in the 'Gene+Variation' feature is 2, it is a categorical feature

In [20]:

```
unique = train_df['Gene+Variation'].value_counts()
print('Number of Unique values :', unique.shape[0])
# the top 10 genes that occured most
print(unique.head(10))
```

```
Number of Unique values : 2124
TSC2 S1653P               1
ERBB4 Fusions             1
PTPN11 D61N               1
EGFR A750P                1
BRCA2 V1306I              1
PTEN D92E                 1
PMS2 E5K                  1
MSH6 V509A                1
ROS1 LRIG3-ROS1_Fusion    1
CDKN2A P48L               1
Name: Gene+Variation, dtype: int64
```

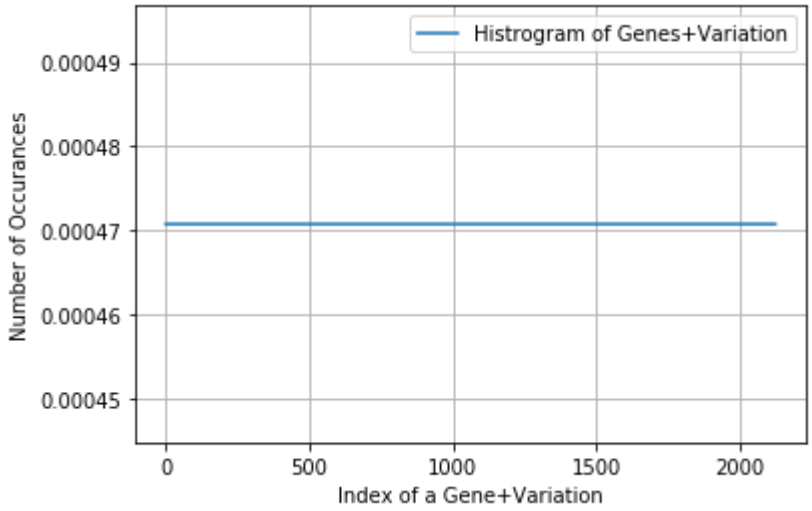
In [21]:

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

```
Ans: There are 2124 different categories of genes & variation in the train
data, and they are distibuted as follows
```

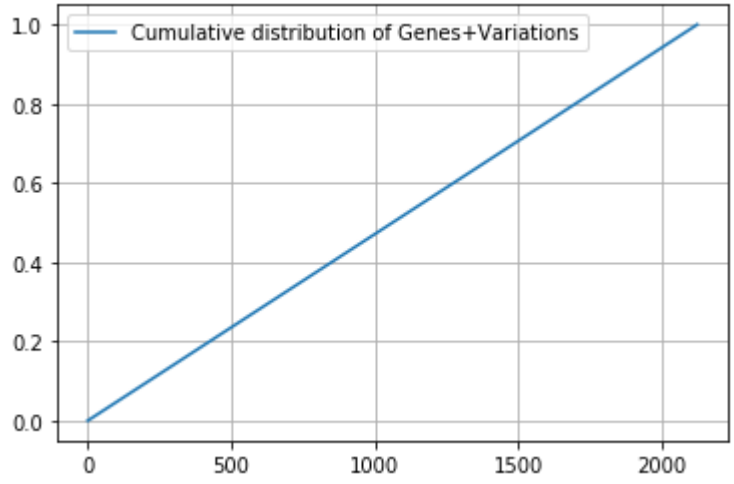
In [22]:

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



In [23]:

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



- None of the values in this feature have repeated

5.1.1 Encoding

In [80]:

```
gene_var_vectorizer = TfidfVectorizer()
train_gene_var_feature_onehotCoding = gene_var_vectorizer.fit_transform(train_df['Gene+
Variation'])
test_gene_var_feature_onehotCoding = gene_var_vectorizer.transform(test_df['Gene+Variat
ion'])
cv_gene_var_feature_onehotCoding = gene_var_vectorizer.transform(cv_df['Gene+Variation'
])
```

In [81]:

```
train_gene_var_feature_onehotCoding= normalize(train_gene_var_feature_onehotCoding, axi
s=0)
cv_gene_var_feature_onehotCoding= normalize(cv_gene_var_feature_onehotCoding, axis=0)
test_gene_var_feature_onehotCoding= normalize(test_gene_var_feature_onehotCoding, axis=
0)
```

In [82]:

```
train_gene_var_feature_onehotCoding.shape
```

Out[82]:

```
(2124, 2166)
```

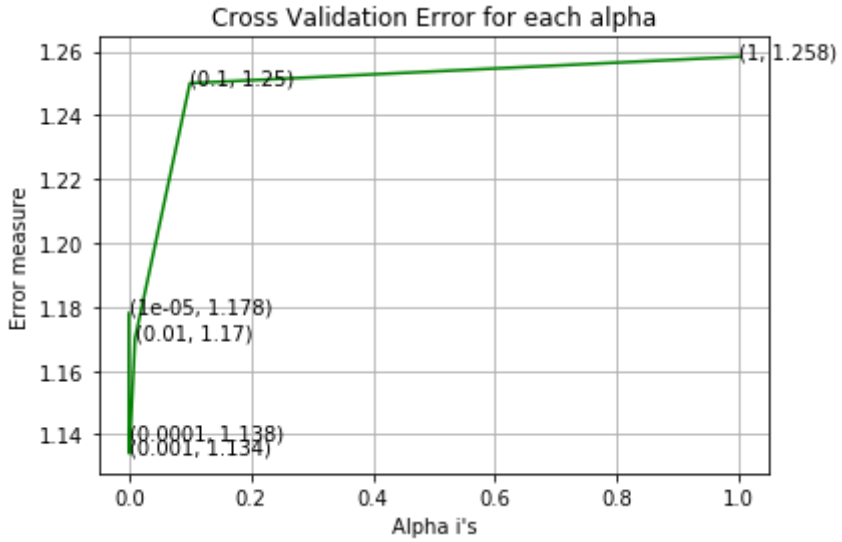
5.1.2 Checking for the feature's influence on the response label

In [27]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
klearn.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='opt
imal', 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 Gradie
nt 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_var_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_var_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_var_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, lab
els=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_var_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_var_feature_onehotCoding, y_train)

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

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

```
For values of alpha =  1e-05 The log loss is: 1.1779979661579758
For values of alpha =  0.0001 The log loss is: 1.1383347022562214
For values of alpha =  0.001 The log loss is: 1.1340800100756958
For values of alpha =  0.01 The log loss is: 1.1700386272918484
For values of alpha =  0.1 The log loss is: 1.2500400994109426
For values of alpha =  1 The log loss is: 1.2583265469377447
```




```
For values of best alpha =  0.001 The train log loss is: 0.378125916469000
2
For values of best alpha =  0.001 The cross validation log loss is: 1.1340
800100756958
For values of best alpha =  0.001 The test log loss is: 1.2223146214779916
```

- This feature is useful as the test log loss <2.5(Random model)

6.0 Word count

In [28]:

```
# Normalizing the feature
from sklearn.preprocessing import MinMaxScaler
normalizer = MinMaxScaler()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer.fit(train_df['TEXT_wordcount'].values.reshape(-1,1))

train_count_norm = normalizer.transform(train_df['TEXT_wordcount'].values.reshape(-1,1
))
cv_count_norm = normalizer.transform(cv_df['TEXT_wordcount'].values.reshape(-1,1))
test_count_norm = normalizer.transform(test_df['TEXT_wordcount'].values.reshape(-1,1))
```

In [29]:

```
print(train_count_norm.shape)
```

```
(2124, 1)
```

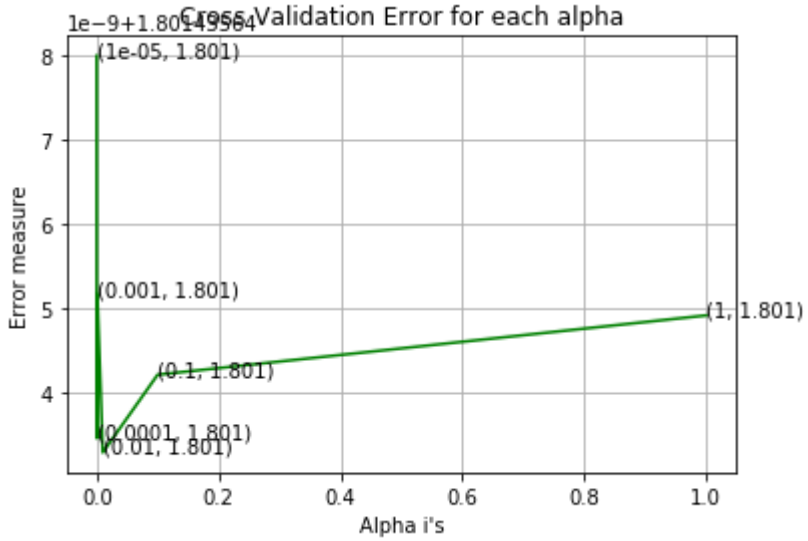
6.1 Checking for the feature's influence on the response label

In [30]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
klearn.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='opt
imal', 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 Gradie
nt 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_count_norm, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_count_norm, y_train)
    predict_y = sig_clf.predict_proba(cv_count_norm)
    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, lab
els=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_count_norm, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_count_norm, y_train)

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

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

```
For values of alpha =  1e-05 The log loss is: 1.8014356480000075
For values of alpha =  0.0001 The log loss is: 1.8014356434635619
For values of alpha =  0.001 The log loss is: 1.8014356451658382
For values of alpha =  0.01 The log loss is: 1.8014356432949388
For values of alpha =  0.1 The log loss is: 1.8014356442173822
For values of alpha =  1 The log loss is: 1.8014356449203064
```




```
For values of best alpha =  0.01 The train log loss is: 1.809621999401289
For values of best alpha =  0.01 The cross validation log loss is: 1.80143
56432949388
For values of best alpha =  0.01 The test log loss is: 1.822818160559836
```

- This feature is useful as the test log loss <2.5 (Random model)

7.0 Preparing data for the model

In [39]:

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

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

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

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

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in t
wo 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, yticklabel
s=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, yticklabel
s=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, yticklabel
s=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [40]:

```
#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 will provide the array of probabilities belongs to ea
ch 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 [41]:

```
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 [42]:

```
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

8.0 Stacking the features

In [89]:

```
# merging gene, variance, text , gene+variation & word_count features

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_f
eature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feat
ure_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_on
ehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCo
ding,
                               train_gene_var_feature_onehotCoding, train_count_norm)).
tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCodin
g,
                              test_gene_var_feature_onehotCoding, test_count_norm)).tocs
r()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding,
                            cv_gene_var_feature_onehotCoding, cv_count_norm)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [90]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCo
ding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodi
ng.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_o
nehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 9352)
(number of data points * number of features) in test data =  (665, 9352)
(number of data points * number of features) in cross validation data = (5
32, 9352)
```

I shall be applying logistic regression because of the dimensionality is high & the history of its good performances in previous models

9.0 Logistic Regression

9.1 With Class balancing

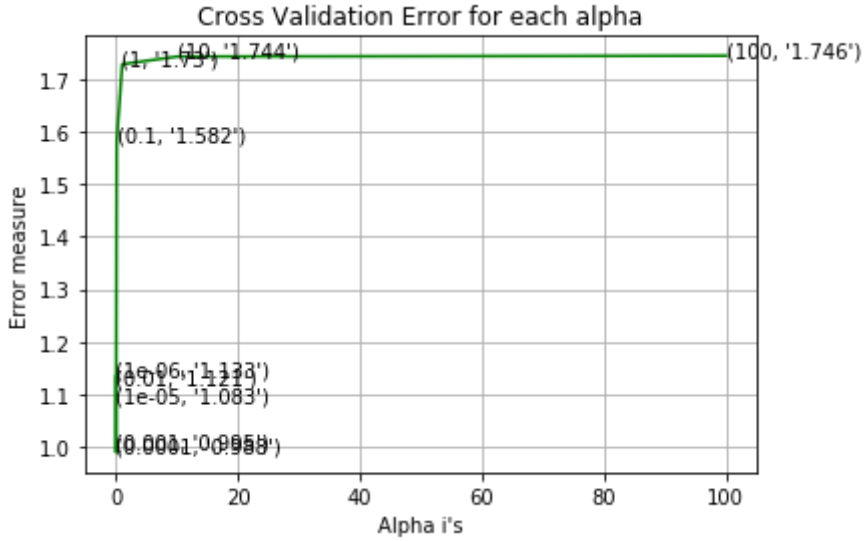
9.1.1 Hyper paramter tuning

In [91]:

```
# 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', los
s='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.133326896586863
for alpha = 1e-05
Log Loss : 1.0831517455996202
for alpha = 0.0001
Log Loss : 0.988238267689689
for alpha = 0.001
Log Loss : 0.995007325981068
for alpha = 0.01
Log Loss : 1.1208714811854623
for alpha = 0.1
Log Loss : 1.5823847613791564
for alpha = 1
Log Loss : 1.7297247513003642
for alpha = 10
Log Loss : 1.7443676570832565
for alpha = 100
Log Loss : 1.7459760481401971
```



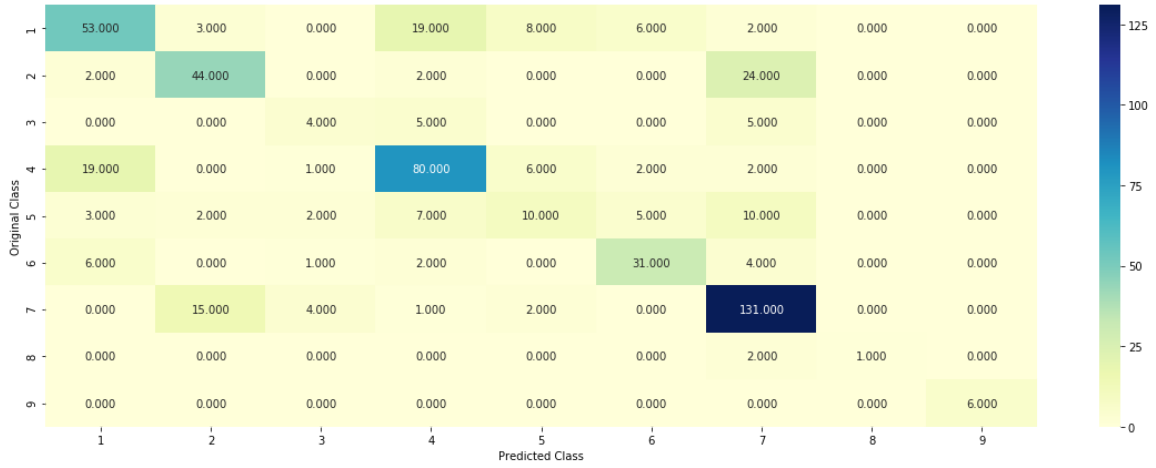

```
For values of best alpha =  0.0001 The train log loss is: 0.38789371818814
655
For values of best alpha =  0.0001 The cross validation log loss is: 0.988
238267689689
For values of best alpha =  0.0001 The test log loss is: 1.015722205031617
```

9.1.2 Testing the model with best hyper paramters</h4>

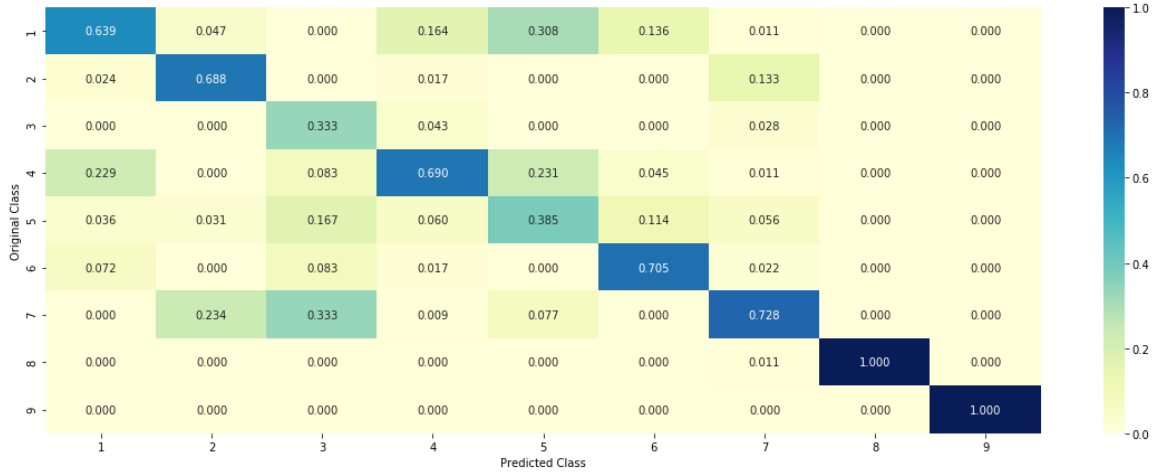
In [93]:

```
# 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)
```

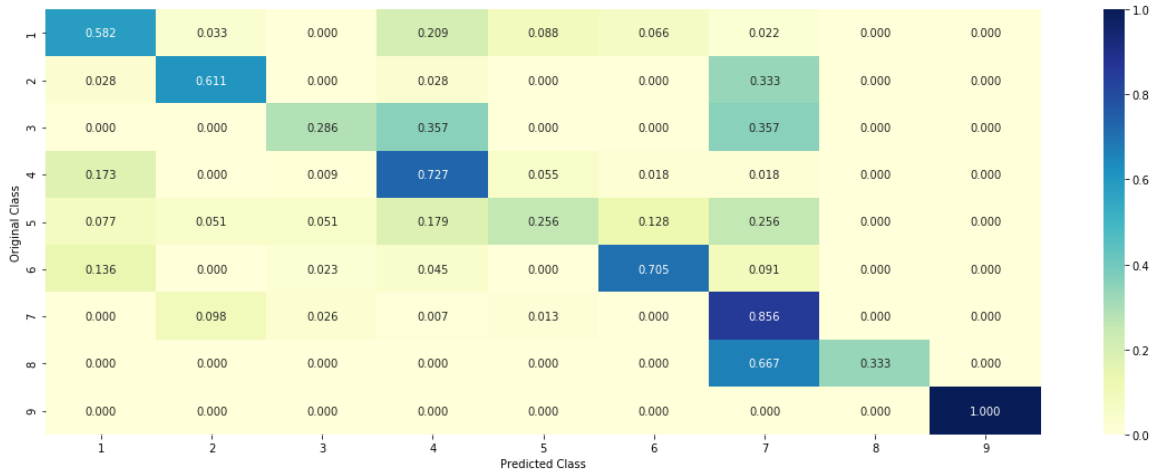
```
Log loss : 0.988238267689689
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```




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




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



Feature Importance</h4>

In [94]:

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

9.1.4 Sample test point-1</h5>

In [95]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[2.800e-03 8.766e-01 7.000e-04 7.000e-04
3.200e-03 9.000e-04 1.075e-01
  7.500e-03 0.000e+00]]
Actual Class : 2
--------------------------------------------------
59 Text feature [able] present in test data point [True]
115 Text feature [79] present in test data point [True]
120 Text feature [adjust] present in test data point [True]
124 Text feature [active] present in test data point [True]
137 Text feature [9b] present in test data point [True]
200 Text feature [85] present in test data point [True]
325 Text feature [072] present in test data point [True]
396 Text feature [32] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

9.1.5 Sample test point-2</h5>

In [96]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.2859 0.1587 0.0086 0.0195 0.3296 0.1437
0.0507 0.0029 0.0004]]
Actual Class : 6
--------------------------------------------------
58 Text feature [90] present in test data point [True]
243 Text feature [aberrations] present in test data point [True]
387 Text feature [41] present in test data point [True]
458 Text feature [101] present in test data point [True]
468 Text feature [17q25] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

9.2 Without Class balancing</h3>

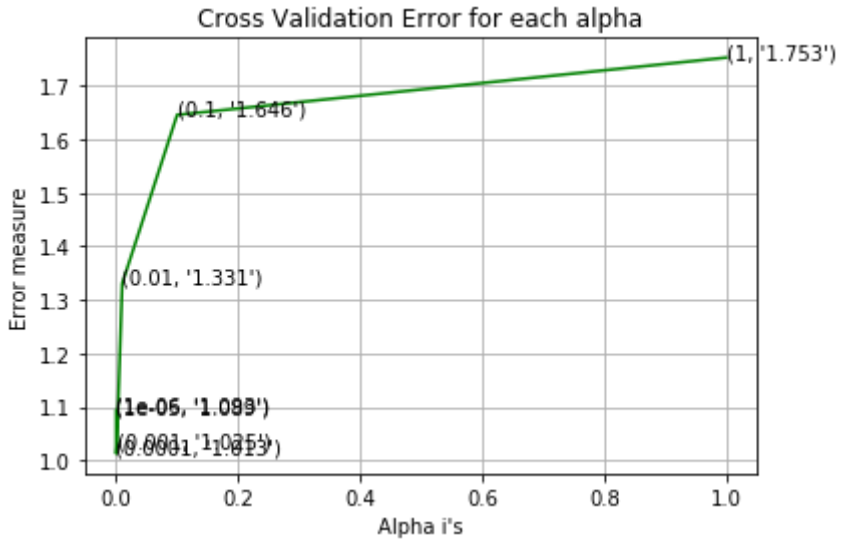
9.2.1 Hyper paramter tuning</h4>

In [97]:

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

```
for alpha = 1e-06
Log Loss : 1.0929005383197161
for alpha = 1e-05
Log Loss : 1.0890041318009342
for alpha = 0.0001
Log Loss : 1.0125538696884624
for alpha = 0.001
Log Loss : 1.0250735095338426
for alpha = 0.01
Log Loss : 1.3307722126353938
for alpha = 0.1
Log Loss : 1.6458508940362924
for alpha = 1
Log Loss : 1.7528953272871888
```



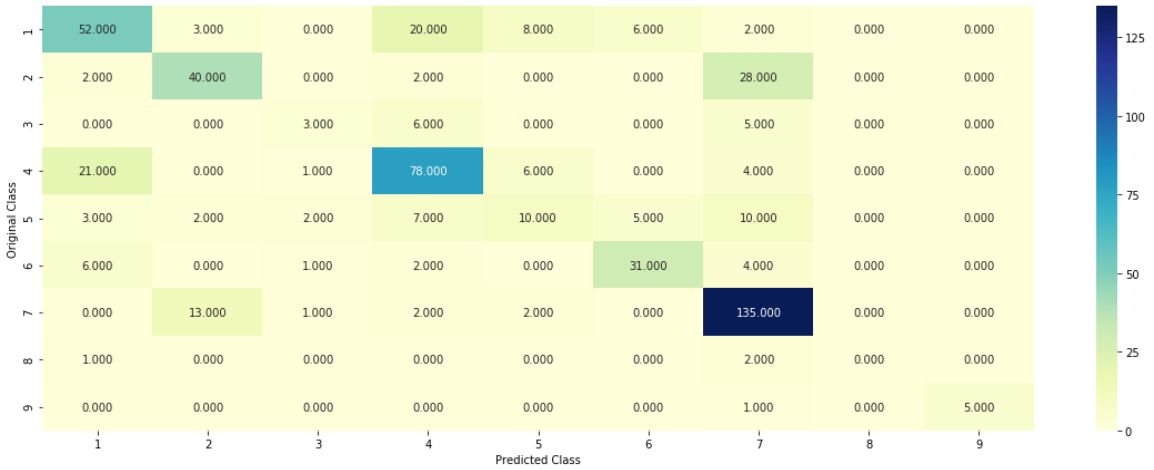

```
For values of best alpha =  0.0001 The train log loss is: 0.38907574795367
755
For values of best alpha =  0.0001 The cross validation log loss is: 1.012
5538696884624
For values of best alpha =  0.0001 The test log loss is: 1.036459158641153
```

9.2.2 Testing model with best hyper parameters</h4>

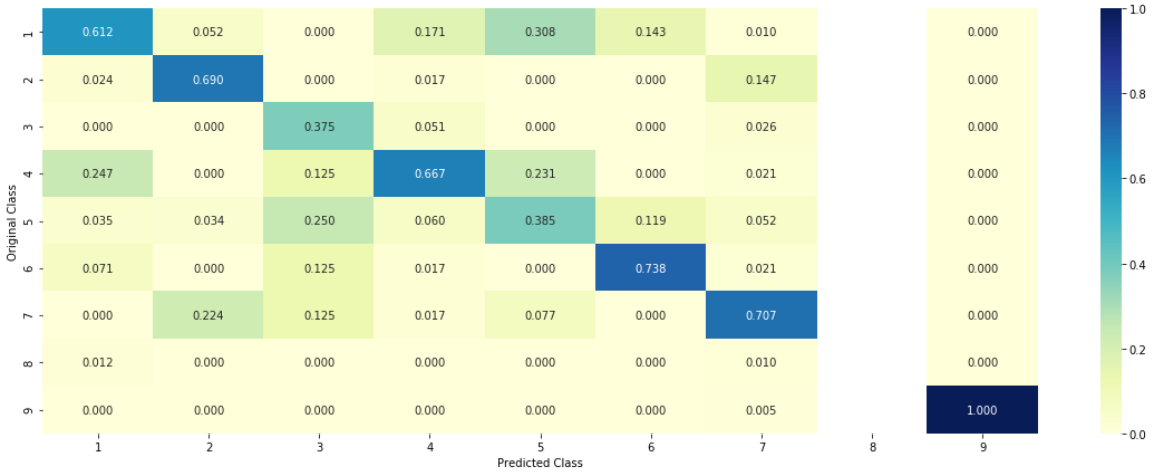
In [98]:

```
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)
```

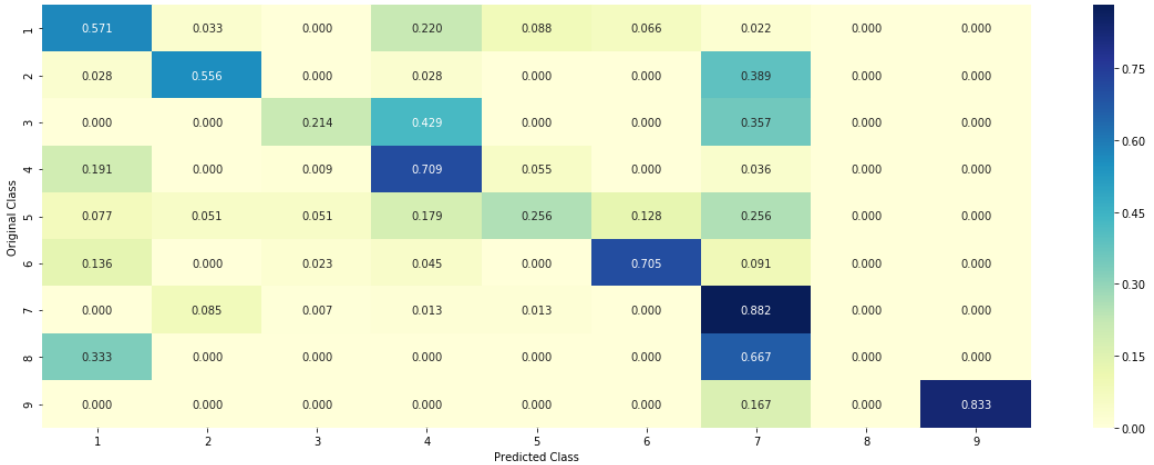
```
Log loss : 1.0125538696884624
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```




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




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



9.2.3 Feature Importance, Sample test point-1</h4>

In [99]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[3.200e-03 8.483e-01 4.000e-04 8.000e-04
2.800e-03 9.000e-04 1.425e-01
  1.000e-03 0.000e+00]]
Actual Class : 2
--------------------------------------------------
44 Text feature [able] present in test data point [True]
117 Text feature [79] present in test data point [True]
122 Text feature [active] present in test data point [True]
125 Text feature [adjust] present in test data point [True]
132 Text feature [9b] present in test data point [True]
213 Text feature [85] present in test data point [True]
315 Text feature [072] present in test data point [True]
389 Text feature [32] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

9.2.4 Feature Importance, Sample test point-2</h4>

In [100]:

```
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_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[2.917e-01 1.607e-01 6.600e-03 1.740e-02
3.083e-01 1.425e-01 6.860e-02
  4.000e-03 2.000e-04]]
Actual Class : 6
--------------------------------------------------
42 Text feature [90] present in test data point [True]
260 Text feature [aberrations] present in test data point [True]
381 Text feature [41] present in test data point [True]
482 Text feature [101] present in test data point [True]
486 Text feature [17q25] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

Summary & Conclusion

In [52]:

```
#Ref: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
print('Summary')
x.field_names = ["Vectorizer/Task","Model","Train-loss" ,"CV-loss","Test-loss","Misclas
sification rate"]
x.add_row(["None", 'Random', 'NIL', 2.503,2.561,'NIL'])
x.add_row(["---------------------------------", "--------------------", "-----------"
, "----------","----------","--------------------"])
x.add_row(["TFIDF 1K words", 'NB', 0.440, 1.187,1.187,'36.84%'])
x.add_row(["TFIDF 1K words", 'KNN', 0.729, 1.078,1.073,'39.29%'])
x.add_row(["TFIDF 1K words", 'LR(Class balanced)', 0.723, 1.078,1.023,'35.34%'])
x.add_row(["TFIDF 1K words", 'LR(Class Notbalanced)', 0.389, 1.102,0.978,'35.71%'])
x.add_row(["TFIDF 1K words", 'Linear SVM', 0.315, 1.101,0.986,'35.53%'])
x.add_row(["TFIDF 1K words", 'RF(One hot encoded)', 0.856, 1.174,1.192,'39.85%'])
x.add_row(["TFIDF 1K words", 'RF(Response encoded)', 0.062, 1.288,1.231,'44.74%'])
x.add_row(["TFIDF 1K words", 'Stacking classifier', 0.336, 1.240,1.234,'38.50%'])
x.add_row(["TFIDF 1K words", 'Max Voting classifier', 0.791, 1.196,1.183,'38.20%'])
x.add_row(["TFIDF 1K words", 'RF(One hot encoded)', 0.856, 1.174,1.192,'39.85%'])
x.add_row(["---------------------------------", "--------------------", "-----------"
, "----------","----------","--------------------"])
x.add_row(["Countvectorizer with Bi-grams", 'LR(Class balanced)', 0.726, 1.158,1.148,'3
7.03%'])
x.add_row(["Countvectorizer with Bi-grams", 'LR(Class Notbalanced)', 0.726, 1.173,1.145
,'35.53%'])
x.add_row(["---------------------------------", "--------------------", "-----------"
, "----------","----------","--------------------"])
x.add_row(["Feature Engineering", 'LR(Class balanced)', 0.388, 1.016,0.988,'32.33%'])
x.add_row(["Feature Engineering", 'LR(Class Notbalanced)', 0.389, 1.036,1.013,'33.46%'
])
print(x)
```

Summary

Vectorizer/Task	Model	Train-loss	CV-loss	Test-loss	Misclassification rate
None	Random	NIL	2.503	2.561	NIL
TFIDF 1K words	NB	0.44	1.187	1.187	36.84%
TFIDF 1K words	KNN	0.729	1.078	1.073	39.29%
TFIDF 1K words	LR(Class balanced)	0.723	1.078	1.023	35.34%
TFIDF 1K words	LR(Class Notbalanced)	0.389	1.102	0.978	35.71%
TFIDF 1K words	Linear SVM	0.315	1.101	0.986	35.53%
TFIDF 1K words	RF(One hot encoded)	0.856	1.174	1.192	39.85%
TFIDF 1K words	RF(Response encoded)	0.062	1.288	1.231	44.74%
TFIDF 1K words	Stacking classifier	0.336	1.24	1.234	38.50%
TFIDF 1K words	Max Voting classifier	0.791	1.196	1.183	38.20%
TFIDF 1K words	RF(One hot encoded)	0.856	1.174	1.192	39.85%
Countvectorizer with Bi-grams	LR(Class balanced)	0.726	1.158	1.148	37.03%
Countvectorizer with Bi-grams	LR(Class Notbalanced)	0.726	1.173	1.145	35.53%
Feature Engineering	LR(Class balanced)	0.388	1.016	0.988	32.33%
Feature Engineering	LR(Class Notbalanced)	0.389	1.036	1.013	33.46%

- The least test loss of 0.988 & the least Misclassification rate was obtained when Logistic Regression with class balanced was applied on feature engineered data. The loss can reduce further if more feature engineering is done on the text data
- On the contrary, the worst loss (1.234) & Misclassification rate(44.74%) was obtained for Stacking classifier & Random forest models