Personalized cancer diagnosis

Note:Applying tf-idf features for all the models

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

classification problem

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
```

```
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of
the model_selection module into which all the refactored classes and functi
ons are moved. Also note that the interface of the new CV iterators are dif
ferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training_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[3]:

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

3.1.3. Preprocessing of text

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


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

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

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

In [5]:
```
#text processing stage.
start_time = time.clock()
```

```
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "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 : 171.9264596384564 seconds
```

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

Out[6]:

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

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

Out[7]:

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

1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

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

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

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

# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same 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, test_size=0.2)
```

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

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

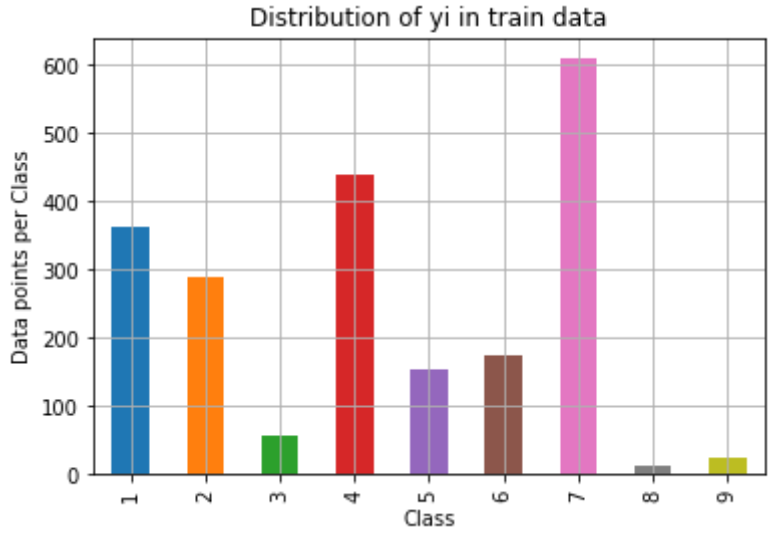
```
Number of data points in train data: 2124
Number of data points in test data: 665
```

```
Number of data points in cross validation data: 532
```

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

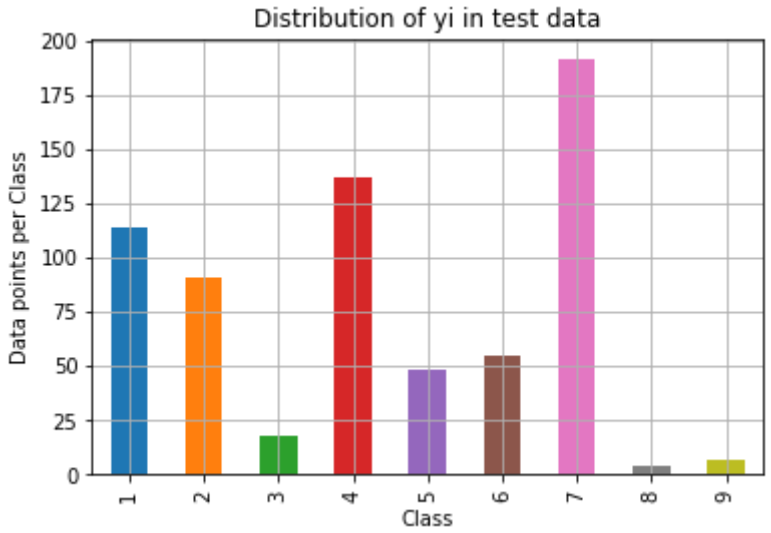
In [12]:
```
# it returns a dict, keys as class labels and values as the number of data poi
nts in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
```

```
    print('Number of data points in class', i+1, ':',test_class_distribution.v
alues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*10
0), 3), '%)')

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

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

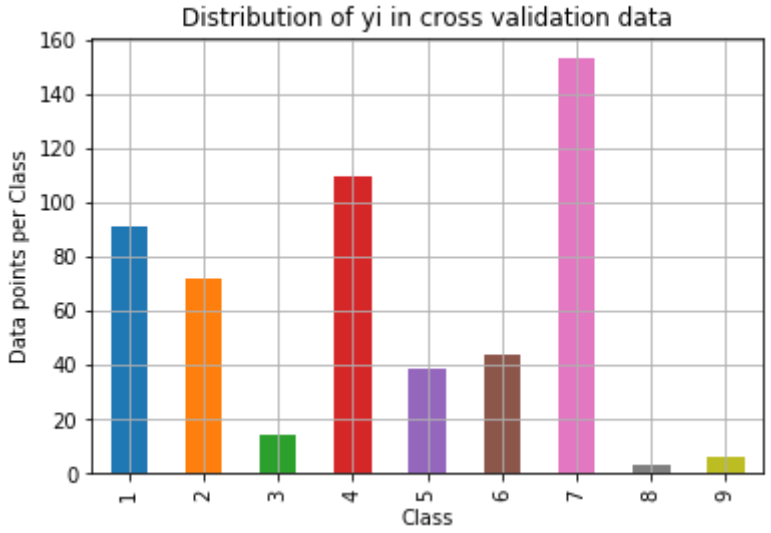



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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

```
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
------------------------------------------------------------------------
-----
```




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




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

3.2 Prediction using a 'Random' Model

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

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

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

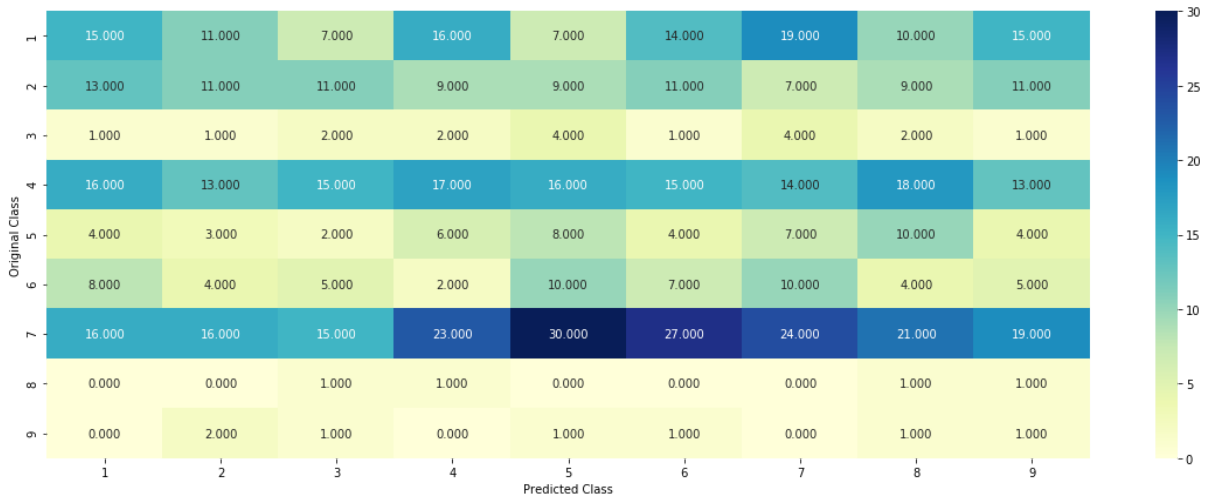
```
    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to
 rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                         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

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

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

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

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

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

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

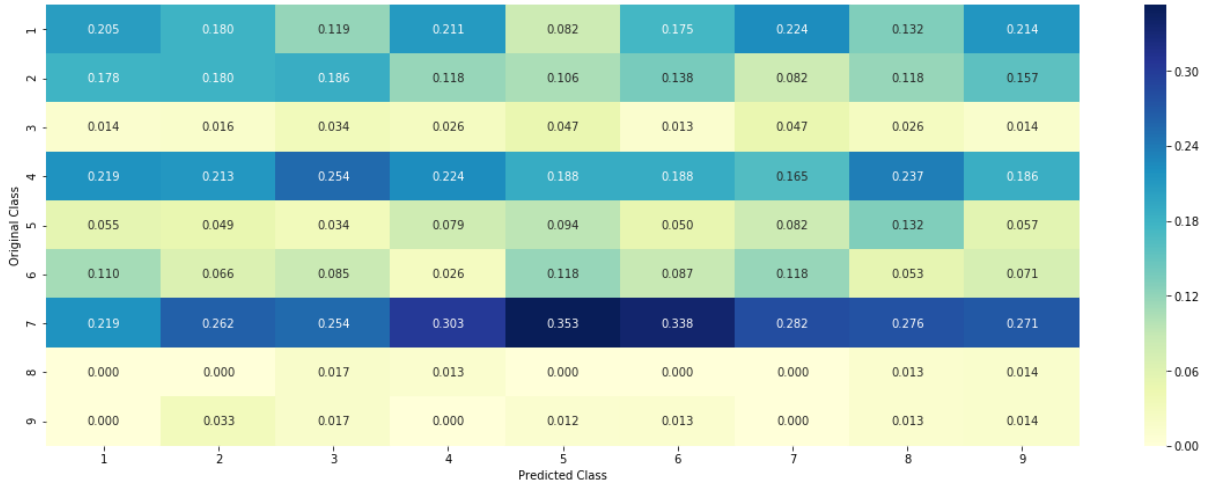
In [14]:

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

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


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

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

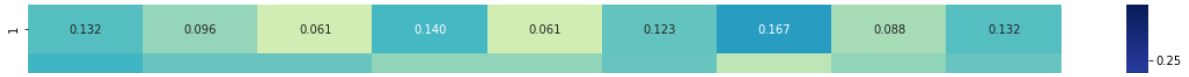
```
Log loss on Cross Validation Data using Random Model 2.425719864779295
Log loss on Test Data using Random Model 2.490323591509003
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [15]:

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

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
```

```
    # Deletion                                43
    # Amplification                           43
    # Fusions                                 22
    # Overexpression                           3
    # E17K                                     3
    # Q61L                                     3
    # S222D                                    2
    # P130S                                    2
    # ...
    # }
    value_count = train_df[feature].value_counts()
    #print(value_count)

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

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

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

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

        # we are adding the gene/variation to the dict as key and vec as value
```

```
        gv_dict[i]=vec
        #print(gv_dict)
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818
18177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.
03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.06122448979591836, 0.0612244897959
18366, 0.27040816326530615, 0.06122448979591836, 0.066326530612244902, 0.0510
20408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818
1818181818177, 0.06818181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.05
6818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060
6060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069
182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920
5295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.2715
2317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333
33334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.299
99999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

```
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

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

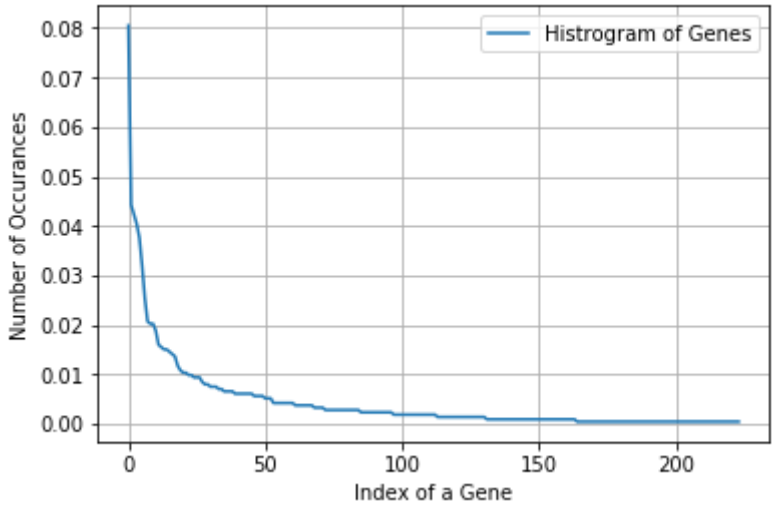
```
Number of Unique Genes : 224
BRCA1     171
TP53       94
EGFR       90
BRCA2      86
PTEN       80
KIT        68
BRAF       54
ERBB2      44
PDGFRA     43
ALK        43
Name: Gene, dtype: int64
```

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

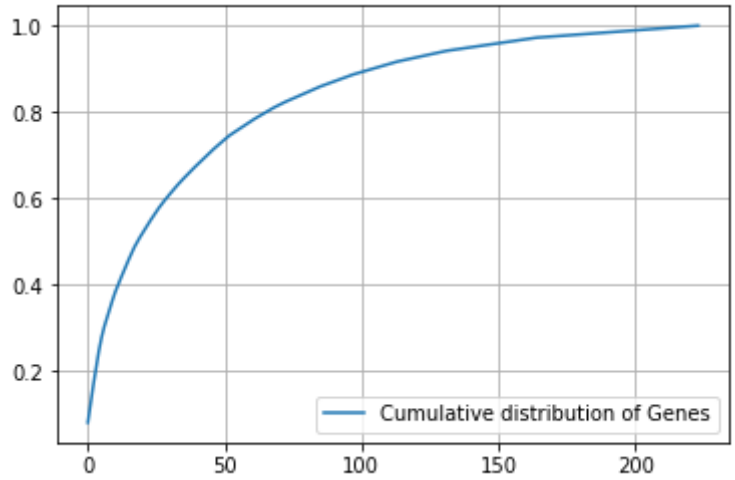
```
Ans: There are 224 different categories of genes in the train data, and the
y are distibuted as follows
```

In [18]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
```

```
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

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

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

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
829     ERCC3
470      TP53
2793    BRCA2
2095    CDK12
```

```
563     SMAD3
Name: Gene, dtype: object
```

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

Out[24]:
```
['abl1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'atm',
 'atr',
 'aurka',
 'aurkb',
 'axin1',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
```

```
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
```

```
'foxp1',
'fubp1',
'gata3',
'gli1',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm4',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
```

```
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3r1',
'pik3r2',
'pim1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
```

```
 'ret',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'sdhb',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'xpo1',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot enco
```

```
ding method. The shape of gene feature:", train_gene_feature_onehotCoding.shap
e)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 223)
```

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

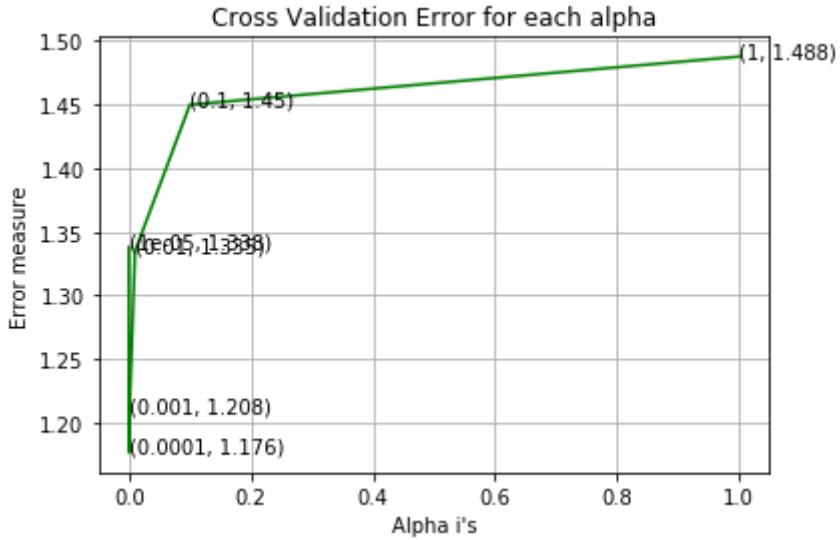
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3378095526619624
For values of alpha =  0.0001 The log loss is: 1.1764001343444395
For values of alpha =  0.001 The log loss is: 1.208084116248677
For values of alpha =  0.01 The log loss is: 1.3346740137595157
For values of alpha =  0.1 The log loss is: 1.4498866905614662
For values of alpha =  1 The log loss is: 1.487659262330749
```




```
For values of best alpha =  0.0001 The train log loss is: 1.046797313527243
5
For values of best alpha =  0.0001 The cross validation log loss is: 1.1764
001343444395
For values of best alpha =  0.0001 The test log loss is: 1.2345466238842773
```

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

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

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  224  g
enes in train dataset?
Ans
```

```
1. In test data 632 out of 665 : 95.03759398496241
2. In cross validation data 514 out of  532 : 96.61654135338345
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

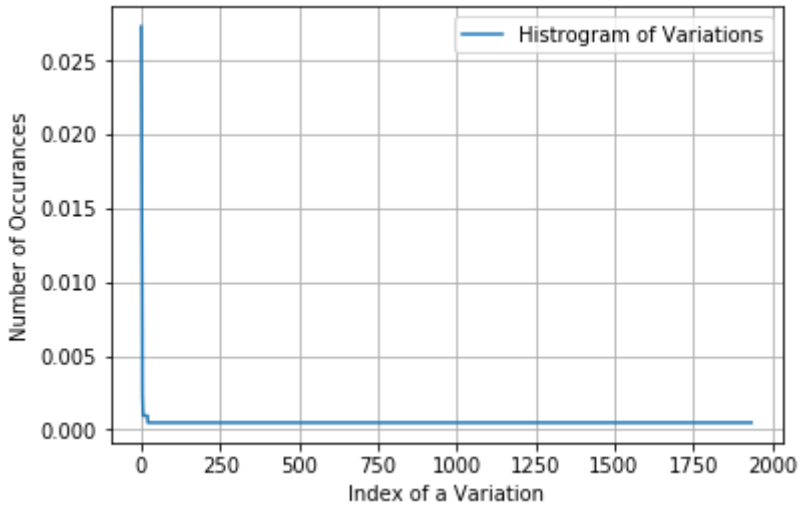
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1935
Truncating_Mutations    58
Deletion                48
Amplification           45
Fusions                 21
Overexpression           5
Q61R                     3
G12S                     2
T58I                     2
ETV6-NTRK3_Fusion        2
I31M                     2
Name: Variation, dtype: int64
```

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

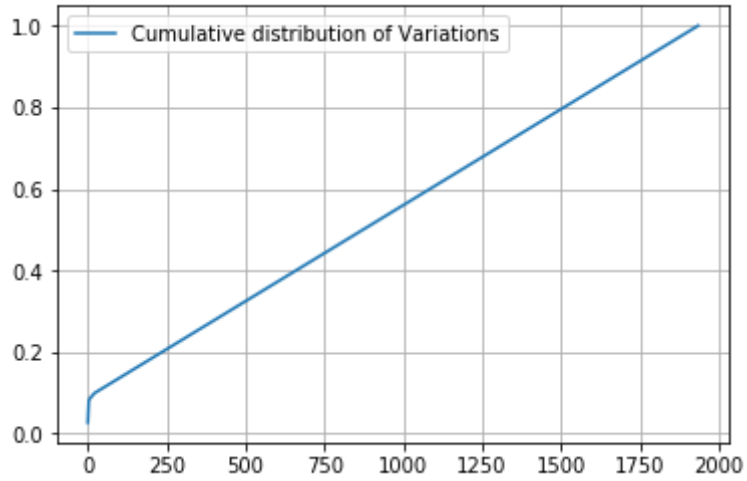
```
Ans: There are 1935 different categories of variations in the train data, a
nd they are distibuted as follows
```

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



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

```
[0.02730697 0.04990584 0.07109228 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [33]:
```
print("train_variation_feature_responseCoding is a converted feature using the  response coding method. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [34]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [35]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1967)
```

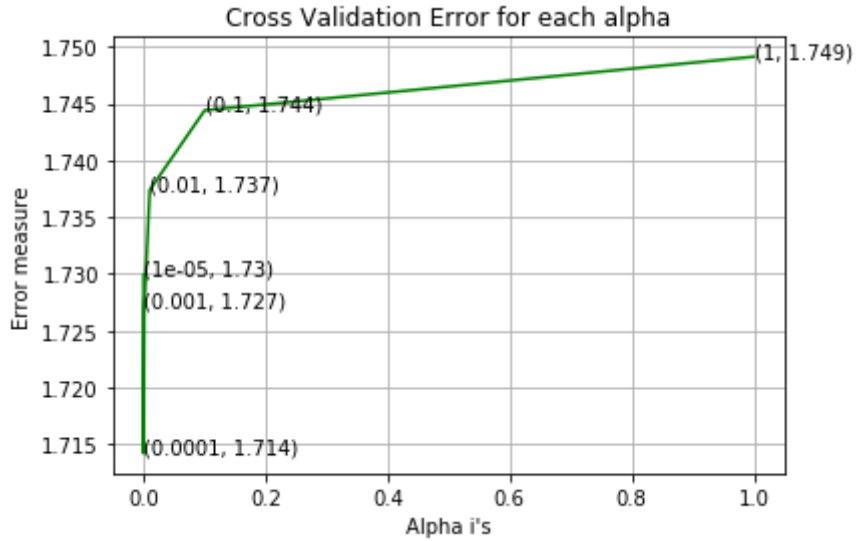
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

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

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

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

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


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

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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.72990030139168
For values of alpha =  0.0001 The log loss is: 1.714177247643087
For values of alpha =  0.001 The log loss is: 1.7271332948411606
For values of alpha =  0.01 The log loss is: 1.7373988727901746
For values of alpha =  0.1 The log loss is: 1.7444111329314724
For values of alpha =  1 The log loss is: 1.7491388556709164
```



```
For values of best alpha =  0.0001 The train log loss is: 0.680945486031652
6
For values of best alpha =  0.0001 The cross validation log loss is: 1.7141
77247643087
For values of best alpha =  0.0001 The test log loss is: 1.68478025440576
```

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

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

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

```
Q12. How many data points are covered by total  1935  genes in test and cro
ss validation data sets?
Ans
1. In test data 75 out of 665 : 11.278195488721805
2. In cross validation data 55 out of  532 : 10.338345864661653
```

3.2.3 Univariate Analysis on Text Feature

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

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

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

In [39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_di
ct.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(
row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [40]:
```
# building a TfidfVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [41]:
```
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 [42]:
```
#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 [43]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train
_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [44]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=0)

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

```
s=0)

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

In [45]:
```
#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 [46]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({250.97448889678301: 1, 174.54146273958983: 1, 136.63141254666957:
1, 128.88104638285526: 1, 128.09509303076126: 1, 115.73245006461485: 1, 11
4.10760028387367: 1, 113.83071923032917: 1, 110.97026329146155: 1, 109.3722
5582732955: 1, 106.09781489187644: 1, 87.84492653498518: 1, 86.927052051061
5: 1, 80.61993400668246: 1, 80.27215972379902: 1, 79.25890458046287: 1, 78.
25637714503685: 1, 77.96459114956828: 1, 77.72266795499122: 1, 76.747089108
70454: 1, 75.51486308611113: 1, 74.14343748704921: 1, 70.08650945525923: 1,
68.76370171096725: 1, 67.93214484014105: 1, 66.34917892788076: 1, 65.780761
01373348: 1, 65.71695411357604: 1, 65.57671266442227: 1, 63.41082329818685
5: 1, 63.04800544318809: 1, 62.179760074690364: 1, 61.97270691839961: 1, 5
8.87575417501205: 1, 57.974780799407235: 1, 56.18708152875718: 1, 55.465376
220344744: 1, 53.42570644800495: 1, 53.3639309975293: 1, 51.03364211093596
5: 1, 50.77068066010372: 1, 49.381606471472686: 1, 49.239248855998916: 1, 4
9.029306477098984: 1, 48.46185057783871: 1, 48.23712207069236: 1, 47.827427
66392606: 1, 47.49504958975509: 1, 46.47864387200395: 1, 45.35952677786522:
1, 45.05410489959667: 1, 44.34746911919273: 1, 43.58050026568977: 1, 43.199
31060802032: 1, 42.71298956435653: 1, 42.3575974819074: 1, 42.2274777722487
4: 1, 42.05571928565234: 1, 42.007933004782856: 1, 41.6478968620431: 1, 41.
26599579624064: 1, 41.258680629694545: 1, 41.23243174146234: 1, 41.19351777
447203: 1, 40.15509621499002: 1, 40.10006224095261: 1, 39.98383763750618:
1, 39.66850388851157: 1, 39.53323573404737: 1, 39.38830625464793: 1, 39.242
59865174743: 1, 38.7604560636323: 1, 38.387646262177185: 1, 38.081709586070
03: 1, 37.73833035750314: 1, 36.98938395229111: 1, 36.530795469814755: 1, 3
6.41661784563156: 1, 36.41402947536723: 1, 35.600314094277515: 1, 35.408898
46567387: 1, 35.24542193241886: 1, 34.99429603320866: 1, 34.93850750273641
4: 1, 34.842987799262595: 1, 34.315714902994046: 1, 34.26504550473307: 1, 3
4.150528366583245: 1, 33.66335277214282: 1, 33.440186843378854: 1, 33.29462
7237200785: 1, 33.03635184676564: 1, 32.74205674030732: 1, 32.5850961291034
8: 1, 32.33133529770985: 1, 32.282853221610424: 1, 32.252648444262135: 1, 3
2.22515343671329: 1, 32.199320781806435: 1, 32.14797848666396: 1, 32.030188
```

21443937: 1, 31.82611958582165: 1, 31.76699296999333: 1, 31.23553878864261
4: 1, 31.23398172851855: 1, 31.22470362083654: 1, 31.212090527010336: 1, 3
1.094288263740093: 1, 31.07190141875079: 1, 30.88078049548536: 1, 30.877029
71023909: 1, 30.698473452943812: 1, 30.611172136060276: 1, 30.3753464180181
4: 1, 30.15647603551912: 1, 29.985993474613903: 1, 29.905981975843023: 1, 2
9.900687177466885: 1, 29.407635860708165: 1, 29.38879323431208: 1, 29.26587
8736384984: 1, 29.242998761499642: 1, 29.21281062195647: 1, 29.083645658352
108: 1, 29.005881342171325: 1, 28.77391216553765: 1, 28.710979176190836: 1,
28.624102740034694: 1, 28.16203975350315: 1, 28.092224830423916: 1, 27.9827
6114395993: 1, 27.979339207137432: 1, 27.954355174925333: 1, 27.89105888808
6512: 1, 27.865888810626455: 1, 27.554381529199905: 1, 27.316696029484376:
1, 27.235292471355248: 1, 27.03709327628687: 1, 26.735782509023977: 1, 26.6
47721920091985: 1, 26.232274745553323: 1, 26.224588137653956: 1, 26.2127180
4081604: 1, 26.03393858078406: 1, 25.98843852154526: 1, 25.874091000897884:
1, 25.801692106003646: 1, 25.667140773565205: 1, 25.48847075704563: 1, 25.4
84953003244428: 1, 25.471239333958202: 1, 25.422868916834073: 1, 25.3735681
51298343: 1, 25.34201248707071: 1, 25.1723196739963: 1, 25.064939643740672:
1, 25.035272808735893: 1, 24.964030818220284: 1, 24.938935325058228: 1, 24.
76373510264125: 1, 24.702847342152648: 1, 24.695077077157595: 1, 24.6322901
35300504: 1, 24.561311070072: 1, 24.459300944710233: 1, 24.418146154697837:
1, 24.261806491188185: 1, 24.24610290403323: 1, 24.126039825643808: 1, 24.0
3391082034403: 1, 23.925161134527087: 1, 23.91516370682415: 1, 23.876113667
37626: 1, 23.869173972957096: 1, 23.718538069877663: 1, 23.664626148355207:
1, 23.65422744499908: 1, 23.61322236664656: 1, 23.328485294027974: 1, 22.99
5847297793077: 1, 22.933102860369665: 1, 22.931204162846875: 1, 22.92780342
3247504: 1, 22.87265532228309: 1, 22.854846866889922: 1, 22.84405779733327:
1, 22.739077714841105: 1, 22.73301341549052: 1, 22.72952957902873: 1, 22.70
803949312401: 1, 22.6781120790463: 1, 22.676259104564988: 1, 22.60995957768
224: 1, 22.584716555865892: 1, 22.583164668458977: 1, 22.531828787054316:
1, 22.526796828505287: 1, 22.51204506914783: 1, 22.45711500503111: 1, 22.40
690619130437: 1, 22.28709367957466: 1, 22.22635777918922: 1, 22.14288561015
3595: 1, 22.033632616888763: 1, 21.98893579365445: 1, 21.981942515167418:
1, 21.908298233622727: 1, 21.86323394915258: 1, 21.774001003722645: 1, 21.7
52110120407316: 1, 21.683962795407023: 1, 21.664864986595884: 1, 21.6520002
68130628: 1, 21.52143945656207: 1, 21.41715662208851: 1, 21.30719835440747:
1, 21.257225742396272: 1, 21.14439567529214: 1, 21.124123379304223: 1, 21.1
06814181306845: 1, 21.096236803882025: 1, 21.028919978585474: 1, 21.0088320
19670653: 1, 20.926101020352064: 1, 20.918004257896943: 1, 20.8921584987773
6: 1, 20.890952716625023: 1, 20.883091207805915: 1, 20.81989072465399: 1, 2
0.81673234822051: 1, 20.75378344950737: 1, 20.684035287151268: 1, 20.651889
218209817: 1, 20.64119008535637: 1, 20.558017014189137: 1, 20.4847365924899
73: 1, 20.477078317484825: 1, 20.387198326866237: 1, 20.248242776663684: 1,
20.234428675696503: 1, 20.226983555984518: 1, 20.144986635307223: 1, 20.116
848413267178: 1, 20.096758470926325: 1, 20.077804756797185: 1, 20.052241502
39518: 1, 20.001664415284132: 1, 19.918053903402622: 1, 19.8672722313957:
1, 19.839302191001362: 1, 19.771934908444823: 1, 19.769116959408013: 1, 19.

759817229937454: 1, 19.72060343626407: 1, 19.71331151515343: 1, 19.65442404
172376: 1, 19.604177744142675: 1, 19.559703048340857: 1, 19.51383015090211:
1, 19.500915635183933: 1, 19.430291136538706: 1, 19.40424988634793: 1, 19.3
48620404913415: 1, 19.341636851216197: 1, 19.3083485497978: 1, 19.209690222
897734: 1, 19.208034861790477: 1, 19.145607959793793: 1, 19.13985804199166
2: 1, 19.095691223846472: 1, 19.08881133246902: 1, 19.073181231938836: 1, 1
9.064262703595542: 1, 19.03038996107544: 1, 18.96737791995555: 1, 18.891091
987170945: 1, 18.88997773493345: 1, 18.880283183672212: 1, 18.8448745637468
5: 1, 18.802048074085217: 1, 18.78545314759974: 1, 18.714624655760016: 1, 1
8.67146466318531: 1, 18.66749048181439: 1, 18.641006818317223: 1, 18.523127
9549239: 1, 18.519980453430275: 1, 18.513139102063874: 1, 18.46976441094431
5: 1, 18.45111420134752: 1, 18.426472587490007: 1, 18.386724401784182: 1, 1
8.35736948313352: 1, 18.34976146172891: 1, 18.346900956563154: 1, 18.306556
09288666: 1, 18.26953134464094: 1, 18.256433119258045: 1, 18.2187923201432
1: 1, 18.217804669686696: 1, 18.161238935361652: 1, 18.043640594025682: 1,
18.021118567643885: 1, 18.015819436345126: 1, 17.994068012737685: 1, 17.939
23238027773: 1, 17.886734401776845: 1, 17.865610298996334: 1, 17.8018609833
0378: 1, 17.780278114423886: 1, 17.74204817946307: 1, 17.71768571294173: 1,
17.674553754857882: 1, 17.62910852088004: 1, 17.525042221515587: 1, 17.5100
79688334205: 1, 17.50207603820357: 1, 17.498202569622382: 1, 17.49308936214
0907: 1, 17.484564409726183: 1, 17.47272082729595: 1, 17.467324160394597:
1, 17.46579662552791: 1, 17.435276773779023: 1, 17.413140751049653: 1, 17.3
94898527018974: 1, 17.391745682211308: 1, 17.36054668637901: 1, 17.29929249
505919: 1, 17.279951179745048: 1, 17.279217012649532: 1, 17.27477238895832:
1, 17.190588054718788: 1, 17.149851491171283: 1, 17.048670042995543: 1, 17.
011770585701868: 1, 17.000523054901812: 1, 16.955049326492823: 1, 16.900839
285098968: 1, 16.887594468797772: 1, 16.876141588517132: 1, 16.872836036478
425: 1, 16.816456471265415: 1, 16.806728379117864: 1, 16.79705461143645: 1,
16.782473644762174: 1, 16.77554210237415: 1, 16.69710875948054: 1, 16.69499
1990612852: 1, 16.684308317343586: 1, 16.679333166093123: 1, 16.63731067638
8644: 1, 16.610060852182038: 1, 16.595738975936595: 1, 16.47495681944273:
1, 16.46884383392688: 1, 16.41354804776118: 1, 16.365615792780456: 1, 16.34
015279773669: 1, 16.324377638228185: 1, 16.309018151428344: 1, 16.275806205
01217: 1, 16.234040161083634: 1, 16.232792583749255: 1, 16.1529501263007:
1, 16.149526742355636: 1, 16.073540552693043: 1, 16.062141007815377: 1, 16.
048370355959673: 1, 16.02472194212913: 1, 16.02247091039438: 1, 16.01634553
7992486: 1, 15.983320962580239: 1, 15.978406734192165: 1, 15.97022367502999
7: 1, 15.950351765192375: 1, 15.925384121496045: 1, 15.897262990585027: 1,
15.88533626504182: 1, 15.86864208487432: 1, 15.861892467233485: 1, 15.85447
835082501: 1, 15.796442213959244: 1, 15.78568807639014: 1, 15.7799205420780
63: 1, 15.779429272341748: 1, 15.756471202833865: 1, 15.712547280509318: 1,
15.708762620459868: 1, 15.688153502571238: 1, 15.597907457965361: 1, 15.574
126783286514: 1, 15.544040425252888: 1, 15.478894867235987: 1, 15.459705310
924177: 1, 15.381070925802558: 1, 15.378080083646633: 1, 15.36343606255430
8: 1, 15.325607353731115: 1, 15.317909646726939: 1, 15.27672813889081: 1, 1
5.253045265356846: 1, 15.238223342208492: 1, 15.227278856306008: 1, 15.2223

74297356222: 1, 15.198149278804863: 1, 15.189520811263096: 1, 15.1377620712
94368: 1, 15.127883955277454: 1, 15.118233953788707: 1, 15.105660941069427:
1, 15.103565628383711: 1, 15.021959393559301: 1, 14.90079997309457: 1, 14.8
91484833083537: 1, 14.843704237177393: 1, 14.795601896177152: 1, 14.7884797
5432534: 1, 14.767700944764407: 1, 14.726732731799661: 1, 14.72061076906797
8: 1, 14.710484724701319: 1, 14.702520200408841: 1, 14.697342929315766: 1,
14.696400637957765: 1, 14.670937550117003: 1, 14.665152473282507: 1, 14.657
79793898286: 1, 14.629219581520399: 1, 14.628084188075546: 1, 14.6171841881
51128: 1, 14.590665001322666: 1, 14.575464280480471: 1, 14.567973108602807:
1, 14.559521276374571: 1, 14.528181971959617: 1, 14.515855772824297: 1, 14.
49404149854266: 1, 14.485574884559892: 1, 14.483279994958666: 1, 14.4780382
83562825: 1, 14.41083831095038: 1, 14.37842461328394: 1, 14.29916398765530
7: 1, 14.266713191617711: 1, 14.2556820318842: 1, 14.246444740865574: 1, 1
4.228248766558865: 1, 14.223530785852379: 1, 14.186957927488317: 1, 14.1761
54955871983: 1, 14.149570514347799: 1, 14.140695846979149: 1, 14.0908870074
3698: 1, 14.03419085627931: 1, 14.023662571519868: 1, 14.021009771949283:
1, 14.014054641528421: 1, 13.987969494997294: 1, 13.954126385472984: 1, 13.
943751944668414: 1, 13.937075176841546: 1, 13.927744875130202: 1, 13.914521
863904309: 1, 13.904882295506022: 1, 13.897305723618814: 1, 13.870832553532
32: 1, 13.840296867183453: 1, 13.819167621041318: 1, 13.81654197260631: 1,
13.816416940301982: 1, 13.808406681325135: 1, 13.805847300890035: 1, 13.803
101338693592: 1, 13.782607423886239: 1, 13.735277458193865: 1, 13.714976895
872436: 1, 13.708912496843029: 1, 13.706694478830032: 1, 13.61775534975658:
1, 13.602507360846094: 1, 13.56127913789924: 1, 13.55624970510057: 1, 13.53
6132424082078: 1, 13.528995731835801: 1, 13.511244622972034: 1, 13.51025415
7677764: 1, 13.492649880676112: 1, 13.443329625950925: 1, 13.43916418162960
2: 1, 13.420000949643235: 1, 13.402015095886375: 1, 13.391903523720893: 1,
13.387250954059411: 1, 13.37924723552351: 1, 13.36794716298075: 1, 13.32914
1708931717: 1, 13.270875833703695: 1, 13.216650340089982: 1, 13.20487085459
3882: 1, 13.203922327052362: 1, 13.115108774066268: 1, 13.094781975369198:
1, 13.033435727209602: 1, 13.029548373298054: 1, 12.988897893657361: 1, 12.
942171185925337: 1, 12.937396094976025: 1, 12.932101738032584: 1, 12.931013
438200328: 1, 12.91892595612592: 1, 12.905087489768782: 1, 12.9041881773104
63: 1, 12.9039999259935: 1, 12.873779053114136: 1, 12.827624594156882: 1, 1
2.785632160146797: 1, 12.767713217230021: 1, 12.763894724399766: 1, 12.7608
16845264191: 1, 12.715956318282919: 1, 12.680948876376613: 1, 12.6757534639
08398: 1, 12.671134681521417: 1, 12.662073275446062: 1, 12.593917754045458:
1, 12.591561871737442: 1, 12.579090758698399: 1, 12.573238189002046: 1, 12.
556907562526675: 1, 12.549898215188202: 1, 12.54647175704122: 1, 12.5427862
98429677: 1, 12.540942543823506: 1, 12.508235850573838: 1, 12.5070326209894
92: 1, 12.505548695347874: 1, 12.47866274846104: 1, 12.465472429434591: 1,
12.463536765467047: 1, 12.449538992352203: 1, 12.440207591437328: 1, 12.406
068302090437: 1, 12.379304200444112: 1, 12.359468235180522: 1, 12.323677727
125736: 1, 12.302939885387701: 1, 12.302850827127964: 1, 12.29849003215254
1: 1, 12.288401404264697: 1, 12.28777046738073: 1, 12.226188233994963: 1, 1
2.222830262734464: 1, 12.19740242772281: 1, 12.155475615463887: 1, 12.14019

7808368427: 1, 12.127377499060087: 1, 12.091980510967163: 1, 12.08148636501
3495: 1, 12.07905079242403: 1, 12.075175691699826: 1, 12.052188572174414:
1, 12.051681718735269: 1, 12.034553945336613: 1, 12.030493181199434: 1, 12.
022807027325381: 1, 12.014379894318377: 1, 12.004475497445274: 1, 11.981099
56820938: 1, 11.970325399660734: 1, 11.950153571679659: 1, 11.9497011426041
68: 1, 11.941152840904541: 1, 11.92047079485083: 1, 11.872245942320577: 1,
11.870720674689053: 1, 11.86299610186469: 1, 11.841812097426283: 1, 11.8150
93809879006: 1, 11.795139782935735: 1, 11.780990230473519: 1, 11.7594514940
82654: 1, 11.749213514663504: 1, 11.706213070076792: 1, 11.684104506316215:
1, 11.649700072442771: 1, 11.639310581067582: 1, 11.635911356337502: 1, 11.
617673608882578: 1, 11.597166591988698: 1, 11.596523660832913: 1, 11.584392
060052354: 1, 11.579013748694985: 1, 11.577515812886954: 1, 11.569006849058
2: 1, 11.556482706082683: 1, 11.526791647003943: 1, 11.519620791556054: 1,
11.500112186593011: 1, 11.499613958174985: 1, 11.477479086471222: 1, 11.461
61820055438: 1, 11.461366585491062: 1, 11.429628774432318: 1, 11.4149819260
70125: 1, 11.37253007727928: 1, 11.367427495048123: 1, 11.332139250917944:
1, 11.327379661818712: 1, 11.286516628969675: 1, 11.257737361329868: 1, 11.
246443582822677: 1, 11.245914791965053: 1, 11.226133340059441: 1, 11.202394
213096596: 1, 11.195872303697962: 1, 11.176406834925155: 1, 11.169959756512
984: 1, 11.140428687322997: 1, 11.138734278607421: 1, 11.068723350196844:
1, 11.056742561415952: 1, 11.054723208711337: 1, 11.007074276895915: 1, 11.
004835107716367: 1, 10.969136469497293: 1, 10.967275653769239: 1, 10.965192
602942157: 1, 10.961620296107826: 1, 10.95833282756293: 1, 10.9470171428978
5: 1, 10.929600525899241: 1, 10.926464548976753: 1, 10.906090031154651: 1,
10.900515645419476: 1, 10.89978620548973: 1, 10.895368589534497: 1, 10.8858
09487758056: 1, 10.882969331476646: 1, 10.866374956157333: 1, 10.8654333510
69978: 1, 10.817234423532957: 1, 10.815122532151962: 1, 10.810692132857154:
1, 10.806146052150508: 1, 10.796724769378304: 1, 10.796211383637477: 1, 10.
774052069933827: 1, 10.767861228661513: 1, 10.765823013584267: 1, 10.759049
462515609: 1, 10.749992739613225: 1, 10.745242290793424: 1, 10.741852107620
083: 1, 10.73665167327275: 1, 10.699118673195029: 1, 10.697972099326968: 1,
10.693577786330561: 1, 10.693453614149636: 1, 10.690134651179832: 1, 10.662
54026470461: 1, 10.620691710853754: 1, 10.583273644743146: 1, 10.5690844543
98059: 1, 10.552050450038584: 1, 10.550391157084196: 1, 10.54309482951379:
1, 10.54194010916262: 1, 10.522944459147228: 1, 10.513944430644875: 1, 10.5
13921022508692: 1, 10.497290373069662: 1, 10.496603275805734: 1, 10.4869283
8648741: 1, 10.485971744824687: 1, 10.485103992102196: 1, 10.48461638780055
2: 1, 10.445679356023703: 1, 10.439544328311799: 1, 10.427771919065632: 1,
10.412221421813651: 1, 10.388087946189462: 1, 10.386292600899287: 1, 10.374
115525395181: 1, 10.35163401802414: 1, 10.351526314155105: 1, 10.3435340405
08745: 1, 10.319232254852194: 1, 10.311381288213735: 1, 10.261512977377144:
1, 10.261286966849278: 1, 10.256415271021195: 1, 10.246623917812258: 1, 10.
228904546546548: 1, 10.217775893735832: 1, 10.216630728355637: 1, 10.199828
284697471: 1, 10.198315477499712: 1, 10.184585867236626: 1, 10.175458166566
084: 1, 10.169172845290364: 1, 10.16627921120524: 1, 10.144640002213908: 1,
10.140585216547853: 1, 10.129906864133496: 1, 10.124798093010224: 1, 10.095

994800900067: 1, 10.090541559753827: 1, 10.077953503559637: 1, 10.073404756
662423: 1, 10.053775716392648: 1, 10.04732147337755: 1, 10.030252010333767:
1, 10.028980926734855: 1, 10.025119890811991: 1, 10.011815542730236: 1, 10.
010585920645699: 1, 9.975211735196318: 1, 9.961867172094347: 1, 9.961767336
861605: 1, 9.956709329560985: 1, 9.955776759889464: 1, 9.94202728729468: 1,
9.939777166279104: 1, 9.936499716310657: 1, 9.932984476556747: 1, 9.8995894
17701003: 1, 9.867427141960512: 1, 9.85785543081681: 1, 9.857610734159241:
1, 9.845000800918509: 1, 9.842775434468747: 1, 9.830154815517282: 1, 9.8199
00390583959: 1, 9.791785997403561: 1, 9.78841529602375: 1, 9.78123570322687
5: 1, 9.77887445385297: 1, 9.77837324246854: 1, 9.77145632784668: 1, 9.7457
41098392923: 1, 9.740776095640364: 1, 9.722555744383103: 1, 9.6975661794227
93: 1, 9.697056133157147: 1, 9.684542356943139: 1, 9.64289957662968: 1, 9.6
28497173146712: 1, 9.628099607599868: 1, 9.622136331785265: 1, 9.5931156974
83773: 1, 9.586599311856105: 1, 9.562728727881986: 1, 9.559793616505697: 1,
9.539859551005145: 1, 9.534449958751551: 1, 9.499799309178417: 1, 9.4989465
47999573: 1, 9.495511755290785: 1, 9.471753387670697: 1, 9.468347588415337:
1, 9.468346682169841: 1, 9.460047438671683: 1, 9.448627485819275: 1, 9.4458
26937930613: 1, 9.436854981924547: 1, 9.420498191761906: 1, 9.4115541348266
4: 1, 9.389888455394052: 1, 9.375956533003393: 1, 9.333635112676296: 1, 9.3
28251475675424: 1, 9.303747142812425: 1, 9.297371358295571: 1, 9.2875888887
34788: 1, 9.269468040131011: 1, 9.260138527157947: 1, 9.259910817622698: 1,
9.256482438239967: 1, 9.252970114176886: 1, 9.2473403502702: 1, 9.238067933
640135: 1, 9.236212944241384: 1, 9.221422635293377: 1, 9.215331342364781:
1, 9.185619637690714: 1, 9.176086611146177: 1, 9.174277346185157: 1, 9.1637
09443257488: 1, 9.15750281929443: 1, 9.155730564628216: 1, 9.1439063592345
6: 1, 9.128055008134487: 1, 9.12692746429295: 1, 9.123475372611633: 1, 9.11
58991722354: 1, 9.115754375154284: 1, 9.114527463405501: 1, 9.1111155491854
58: 1, 9.097884542264298: 1, 9.093709644081907: 1, 9.087721305165784: 1, 9.
084979135204577: 1, 9.080066088981065: 1, 9.079102302470567: 1, 9.039538836
31929: 1, 9.039024420375268: 1, 9.029438503587118: 1, 9.019296941390039: 1,
8.999220842866034: 1, 8.990998529415032: 1, 8.985941049952281: 1, 8.9737737
05771748: 1, 8.962922579195288: 1, 8.94979379243505: 1, 8.945579228448375:
1, 8.94525976260644: 1, 8.93950139205546: 1, 8.931080174640352: 1, 8.929102
842414034: 1, 8.927303166690832: 1, 8.927104707021746: 1, 8.91837940742625
6: 1, 8.91691670527281: 1, 8.915956578830865: 1, 8.853194232002798: 1, 8.84
965966558511: 1, 8.810200701358061: 1, 8.808228794955923: 1, 8.797114146411
673: 1, 8.796712772021028: 1, 8.795997892232341: 1, 8.795258858509174: 1,
8.795182799270863: 1, 8.78974687443102: 1, 8.787479688899147: 1, 8.77210820
6706886: 1, 8.748433200489702: 1, 8.731710512065193: 1, 8.71448424275532:
1, 8.713827754589405: 1, 8.712252969752283: 1, 8.706803083087472: 1, 8.7046
43080464246: 1, 8.700274432902487: 1, 8.699816868568295: 1, 8.6988811338521
37: 1, 8.689892928153277: 1, 8.663872580662796: 1, 8.655003212312868: 1, 8.
64131755949753: 1, 8.635853961015991: 1, 8.61854718175026: 1, 8.60298379656
2471: 1, 8.587645817628363: 1, 8.555204736057123: 1, 8.554037645700733: 1,
8.543783279553328: 1, 8.53556764286705: 1, 8.535120347454749: 1, 8.51663918
2946477: 1, 8.492184881650108: 1, 8.487334086251353: 1, 8.480581216719429:

```
1, 8.475125139457889: 1, 8.473087292035759: 1, 8.464035659576407: 1, 8.4519
01647165835: 1, 8.447226418091347: 1, 8.443389000256568: 1, 8.4242291821446
54: 1, 8.419353375883624: 1, 8.398053295508296: 1, 8.375111741255099: 1, 8.
371229781651197: 1, 8.338557734451282: 1, 8.323502774002709: 1, 8.313591079
675026: 1, 8.312733162101921: 1, 8.290270137348488: 1, 8.28421452538208: 1,
8.253969284423652: 1, 8.231081008489689: 1, 8.229788103295228: 1, 8.2218891
62994815: 1, 8.21697746864332: 1, 8.21472792012188: 1, 8.211589918518982:
1, 8.207916119248306: 1, 8.195114585919509: 1, 8.181245060528488: 1, 8.1782
17893719765: 1, 8.164290981703564: 1, 8.11369438629134: 1, 8.1100049344487
8: 1, 8.10659330031369: 1, 8.082687185153821: 1, 8.079938401219403: 1, 8.05
5659879913849: 1, 8.051173153431247: 1, 8.047400679549394: 1, 8.04733927481
3277: 1, 8.047087224918846: 1, 8.013457716479929: 1, 8.01257985903732: 1,
8.008355173082268: 1, 8.006695810859947: 1, 8.00314504487479: 1, 7.99741382
9973269: 1, 7.997192690139188: 1, 7.9934758944281326: 1, 7.984069982209207:
1, 7.970112194148691: 1, 7.964935344767025: 1, 7.939123142508416: 1, 7.9279
07841255623: 1, 7.9219946628714: 1, 7.9066349638768045: 1, 7.87534137334378
5: 1, 7.868334540111941: 1, 7.868189640165547: 1, 7.863345091154835: 1, 7.8
556530274087875: 1, 7.851419513069988: 1, 7.838172503271764: 1, 7.836836870
292321: 1, 7.829932651106367: 1, 7.823509852237258: 1, 7.798471861671086:
1, 7.789349033111301: 1, 7.786200390412455: 1, 7.775505012922524: 1, 7.7722
30055914027: 1, 7.760570567519384: 1, 7.753667055971727: 1, 7.7284089157654
51: 1, 7.719644212874219: 1, 7.705695881009058: 1, 7.690385805841623: 1, 7.
689913146031865: 1, 7.6850481348597155: 1, 7.682041805042266: 1, 7.65336448
4112959: 1, 7.651021185651142: 1, 7.618136211364001: 1, 7.614272235650247:
1, 7.581512435028614: 1, 7.576817880991117: 1, 7.54445178014697: 1, 7.53472
254419125: 1, 7.528436186880111: 1, 7.524354979874791: 1, 7.51253261467355
8: 1, 7.482141695808252: 1, 7.479713588723235: 1, 7.453347842248788: 1, 7.4
30897046225833: 1, 7.430690527159423: 1, 7.416748432284766: 1, 7.4057549835
59322: 1, 7.393899748277145: 1, 7.380673856316553: 1, 7.364642000952354: 1,
7.364410852309323: 1, 7.32281082393477: 1, 7.2986414645850655: 1, 7.2986087
42485639: 1, 7.281514185962552: 1, 7.228442336213966: 1, 7.197568632186416:
1, 7.194917846168703: 1, 7.191395714227412: 1, 7.1059338978259525: 1, 7.090
838617822934: 1, 7.057260434476651: 1, 7.0430113659599005: 1, 7.03930353761
6054: 1, 7.0262859131534405: 1, 7.009202843678382: 1, 7.008797038156714: 1,
6.996817135002211: 1, 6.994379436306449: 1, 6.9518057158525535: 1, 6.927545
640365672: 1, 6.915245233637761: 1, 6.908019636644286: 1, 6.86793087085245
8: 1, 6.840326425645116: 1, 6.793810655697384: 1, 6.78071115625506: 1, 6.70
9882054205552: 1, 6.6686786322596285: 1, 6.6524069833482455: 1, 6.591820907
0341085: 1, 6.565870033636008: 1, 6.522185612886054: 1, 6.417578992954928:
1, 6.336733089504086: 1, 5.99510987175482: 1})
```

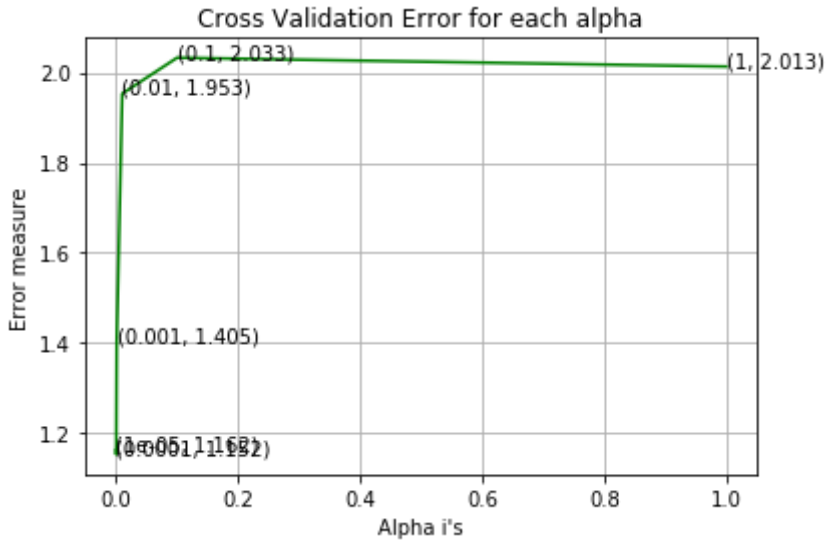
In [47]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
```

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

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

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


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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
```

```
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.1624624695462946
For values of alpha =  0.0001 The log loss is: 1.1524340458893017
For values of alpha =  0.001 The log loss is: 1.4045064133007645
For values of alpha =  0.01 The log loss is: 1.9533037038326455
For values of alpha =  0.1 The log loss is: 2.0330217096752965
For values of alpha =  1 The log loss is: 2.0134529875463216
```




```
For values of best alpha =  0.0001 The train log loss is: 0.837961448379667
2
For values of best alpha =  0.0001 The cross validation log loss is: 1.1524
340458893017
For values of best alpha =  0.0001 The test log loss is: 1.197858447587073
```

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

Ans. Yes, it seems like!

In [48]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [49]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train
 data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in
 train data")
```

```
94.0 % of word of test data appeared in train data
94.6 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [51]:
```
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 [52]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

```
                print(i, "Text feature [{}] present in test data point [{}]".f
ormat(word,yes_no))

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

Stacking the three types of features

In [53]:

```
# merging gene, variance and text features

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

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

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

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

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


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

```
ation_feature_responseCoding))

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

In [54]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3190)
(number of data points * number of features) in test data =  (665, 3190)
(number of data points * number of features) in cross validation data = (53
2, 3190)
```

In [55]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_r
esponseCoding.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (53
2, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

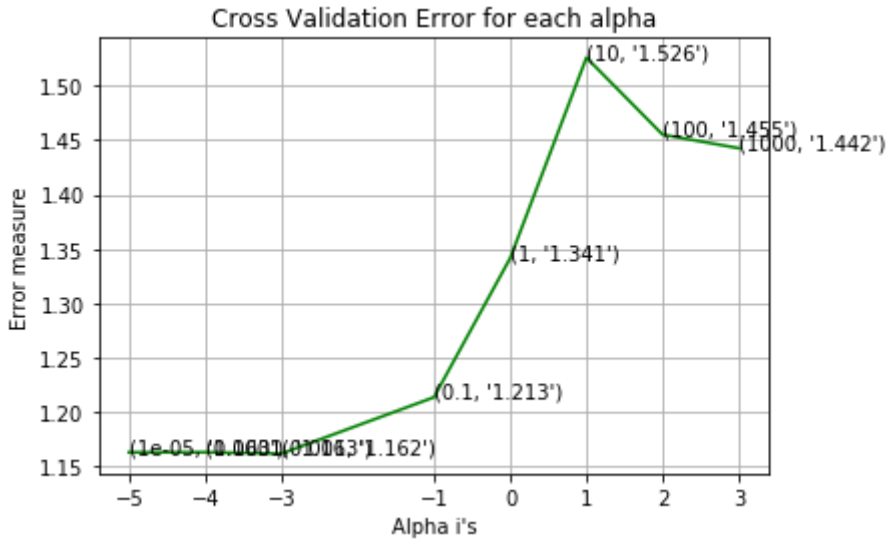
In [56]:

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

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
```

```
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 1.16287570043862
for alpha = 0.0001
Log Loss : 1.1629586100825284
for alpha = 0.001
Log Loss : 1.1616706955649716
for alpha = 0.1
Log Loss : 1.2134786009687712
for alpha = 1
Log Loss : 1.3413822758630012
```

```
for alpha = 10
Log Loss : 1.5257559357990274
for alpha = 100
Log Loss : 1.454907765069671
for alpha = 1000
Log Loss : 1.4424833078315864
```



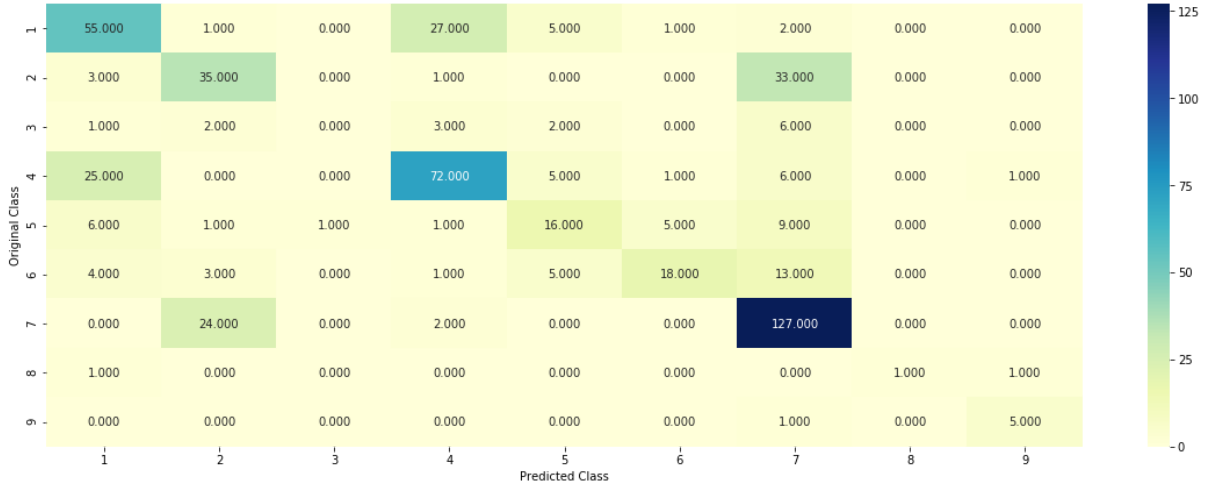

```
For values of best alpha =  0.001 The train log loss is: 0.517992560928667
For values of best alpha =  0.001 The cross validation log loss is: 1.16167
06955649716
For values of best alpha =  0.001 The test log loss is: 1.2272225840419233
```

4.1.1.2. Testing the model with best hyper paramters

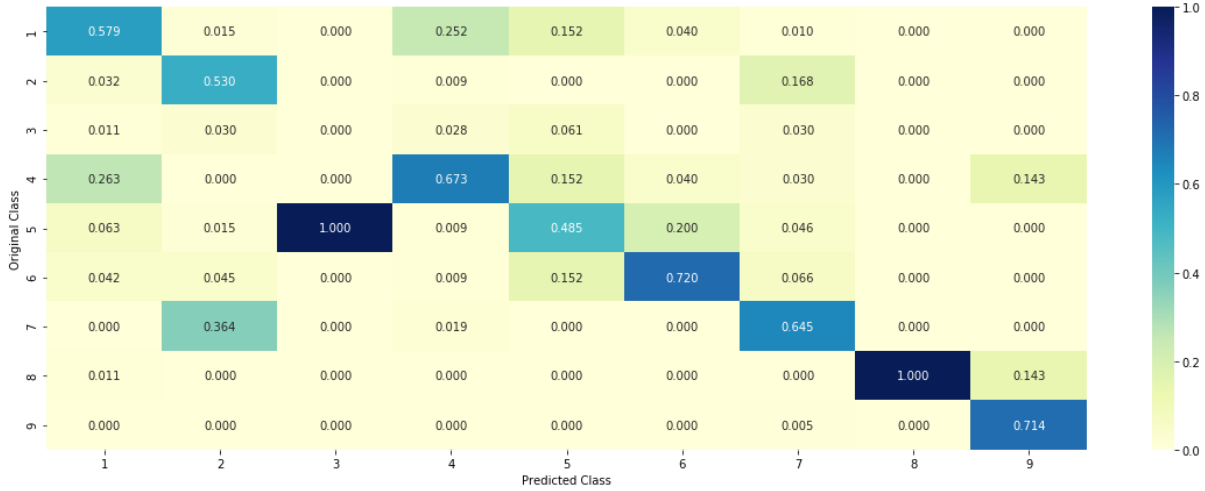
In [57]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

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

```
lessons/naive-bayes-algorithm-1/
# ----------------------


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

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

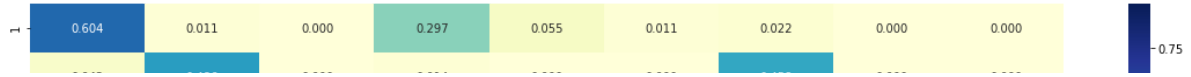
```
Log Loss : 1.1616706955649716
Number of missclassified point : 0.3815789473684211
--------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [58]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5459 0.051  0.0117 0.0726 0.0423 0.1691
0.0994 0.0048 0.0031]]
Actual Class : 6
--------------------------------------------------
12 Text feature [one] present in test data point [True]
13 Text feature [type] present in test data point [True]
14 Text feature [results] present in test data point [True]
15 Text feature [protein] present in test data point [True]
16 Text feature [two] present in test data point [True]
17 Text feature [table] present in test data point [True]
18 Text feature [function] present in test data point [True]
19 Text feature [therefore] present in test data point [True]
20 Text feature [binding] present in test data point [True]
21 Text feature [also] present in test data point [True]
22 Text feature [wild] present in test data point [True]
23 Text feature [determined] present in test data point [True]
24 Text feature [however] present in test data point [True]
26 Text feature [loss] present in test data point [True]
29 Text feature [role] present in test data point [True]
30 Text feature [large] present in test data point [True]
31 Text feature [region] present in test data point [True]
32 Text feature [discussion] present in test data point [True]
33 Text feature [may] present in test data point [True]
34 Text feature [using] present in test data point [True]
35 Text feature [result] present in test data point [True]
36 Text feature [functions] present in test data point [True]
```

```
37 Text feature [either] present in test data point [True]
38 Text feature [used] present in test data point [True]
39 Text feature [specific] present in test data point [True]
40 Text feature [containing] present in test data point [True]
41 Text feature [25] present in test data point [True]
42 Text feature [least] present in test data point [True]
43 Text feature [three] present in test data point [True]
44 Text feature [four] present in test data point [True]
45 Text feature [likely] present in test data point [True]
46 Text feature [data] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [multiple] present in test data point [True]
50 Text feature [analysis] present in test data point [True]
51 Text feature [based] present in test data point [True]
52 Text feature [including] present in test data point [True]
53 Text feature [suggest] present in test data point [True]
55 Text feature [addition] present in test data point [True]
56 Text feature [performed] present in test data point [True]
57 Text feature [important] present in test data point [True]
58 Text feature [15] present in test data point [True]
60 Text feature [shown] present in test data point [True]
61 Text feature [observed] present in test data point [True]
62 Text feature [defined] present in test data point [True]
63 Text feature [studies] present in test data point [True]
64 Text feature [first] present in test data point [True]
65 Text feature [affect] present in test data point [True]
66 Text feature [present] present in test data point [True]
67 Text feature [sequence] present in test data point [True]
68 Text feature [effect] present in test data point [True]
71 Text feature [30] present in test data point [True]
73 Text feature [significant] present in test data point [True]
75 Text feature [several] present in test data point [True]
76 Text feature [proteins] present in test data point [True]
78 Text feature [gene] present in test data point [True]
79 Text feature [genetic] present in test data point [True]
82 Text feature [human] present in test data point [True]
83 Text feature [different] present in test data point [True]
85 Text feature [sufficient] present in test data point [True]
86 Text feature [similar] present in test data point [True]
87 Text feature [introduction] present in test data point [True]
90 Text feature [structure] present in test data point [True]
91 Text feature [associated] present in test data point [True]
92 Text feature [tested] present in test data point [True]
93 Text feature [specificity] present in test data point [True]
94 Text feature [cancer] present in test data point [True]
96 Text feature [transcription] present in test data point [True]
```

```
98 Text feature [compared] present in test data point [True]
99 Text feature [32] present in test data point [True]
Out of the top  100  features  70 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1236 0.0533 0.0104 0.0994 0.0327 0.0312
0.6424 0.0043 0.0027]]
Actual Class : 2
--------------------------------------------------
17 Text feature [activation] present in test data point [True]
18 Text feature [activated] present in test data point [True]
19 Text feature [kinase] present in test data point [True]
20 Text feature [cells] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [expressing] present in test data point [True]
24 Text feature [contrast] present in test data point [True]
25 Text feature [factor] present in test data point [True]
26 Text feature [presence] present in test data point [True]
27 Text feature [growth] present in test data point [True]
28 Text feature [also] present in test data point [True]
29 Text feature [10] present in test data point [True]
30 Text feature [independent] present in test data point [True]
31 Text feature [signaling] present in test data point [True]
32 Text feature [shown] present in test data point [True]
33 Text feature [however] present in test data point [True]
36 Text feature [constitutive] present in test data point [True]
37 Text feature [treatment] present in test data point [True]
38 Text feature [addition] present in test data point [True]
39 Text feature [mutations] present in test data point [True]
```

```
40 Text feature [activating] present in test data point [True]
41 Text feature [compared] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [cell] present in test data point [True]
44 Text feature [well] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [inhibitors] present in test data point [True]
47 Text feature [may] present in test data point [True]
48 Text feature [higher] present in test data point [True]
49 Text feature [potential] present in test data point [True]
50 Text feature [constitutively] present in test data point [True]
51 Text feature [suggest] present in test data point [True]
52 Text feature [inhibition] present in test data point [True]
53 Text feature [increased] present in test data point [True]
54 Text feature [although] present in test data point [True]
55 Text feature [treated] present in test data point [True]
56 Text feature [recently] present in test data point [True]
57 Text feature [3b] present in test data point [True]
58 Text feature [found] present in test data point [True]
59 Text feature [sensitive] present in test data point [True]
60 Text feature [oncogenic] present in test data point [True]
61 Text feature [without] present in test data point [True]
64 Text feature [pathways] present in test data point [True]
65 Text feature [inhibited] present in test data point [True]
66 Text feature [described] present in test data point [True]
67 Text feature [3a] present in test data point [True]
68 Text feature [survival] present in test data point [True]
69 Text feature [absence] present in test data point [True]
70 Text feature [results] present in test data point [True]
71 Text feature [mutation] present in test data point [True]
72 Text feature [increase] present in test data point [True]
73 Text feature [proliferation] present in test data point [True]
74 Text feature [tyrosine] present in test data point [True]
75 Text feature [total] present in test data point [True]
76 Text feature [concentrations] present in test data point [True]
77 Text feature [showed] present in test data point [True]
78 Text feature [activate] present in test data point [True]
79 Text feature [reported] present in test data point [True]
80 Text feature [enhanced] present in test data point [True]
81 Text feature [mutant] present in test data point [True]
82 Text feature [consistent] present in test data point [True]
83 Text feature [phosphorylation] present in test data point [True]
84 Text feature [figure] present in test data point [True]
85 Text feature [studies] present in test data point [True]
86 Text feature [fig] present in test data point [True]
87 Text feature [observed] present in test data point [True]
```

```
88 Text feature [20] present in test data point [True]
89 Text feature [two] present in test data point [True]
90 Text feature [respectively] present in test data point [True]
91 Text feature [mechanism] present in test data point [True]
92 Text feature [including] present in test data point [True]
93 Text feature [12] present in test data point [True]
94 Text feature [therapeutic] present in test data point [True]
95 Text feature [induced] present in test data point [True]
96 Text feature [different] present in test data point [True]
97 Text feature [identified] present in test data point [True]
98 Text feature [previous] present in test data point [True]
99 Text feature [effective] present in test data point [True]
Out of the top  100  features  79 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

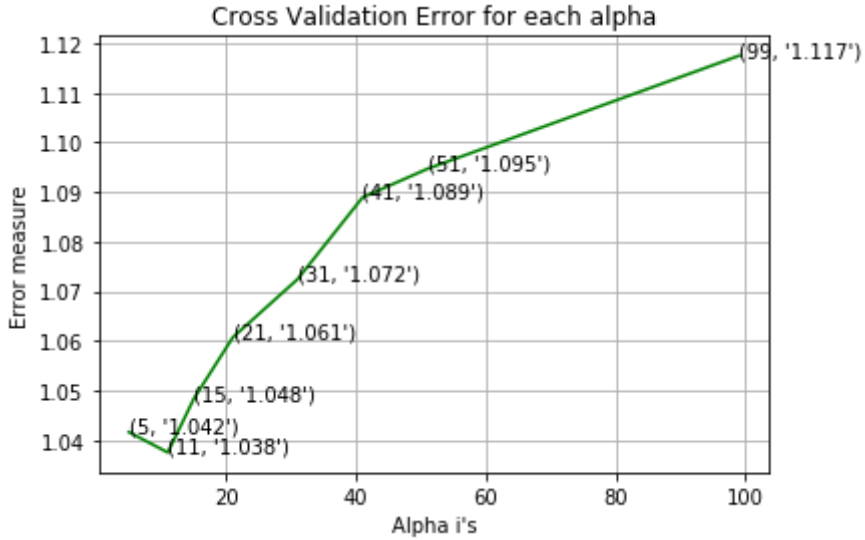
In [60]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', cv=3)
#
```

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


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

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


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

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

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

```
for alpha = 5
Log Loss : 1.041624469030325
for alpha = 11
Log Loss : 1.0375175608550034
for alpha = 15
Log Loss : 1.0483324933811542
for alpha = 21
Log Loss : 1.0606831884088854
for alpha = 31
Log Loss : 1.072435787347024
for alpha = 41
Log Loss : 1.0888957586496217
for alpha = 51
Log Loss : 1.0946624675646328
for alpha = 99
Log Loss : 1.1174522945010426
```

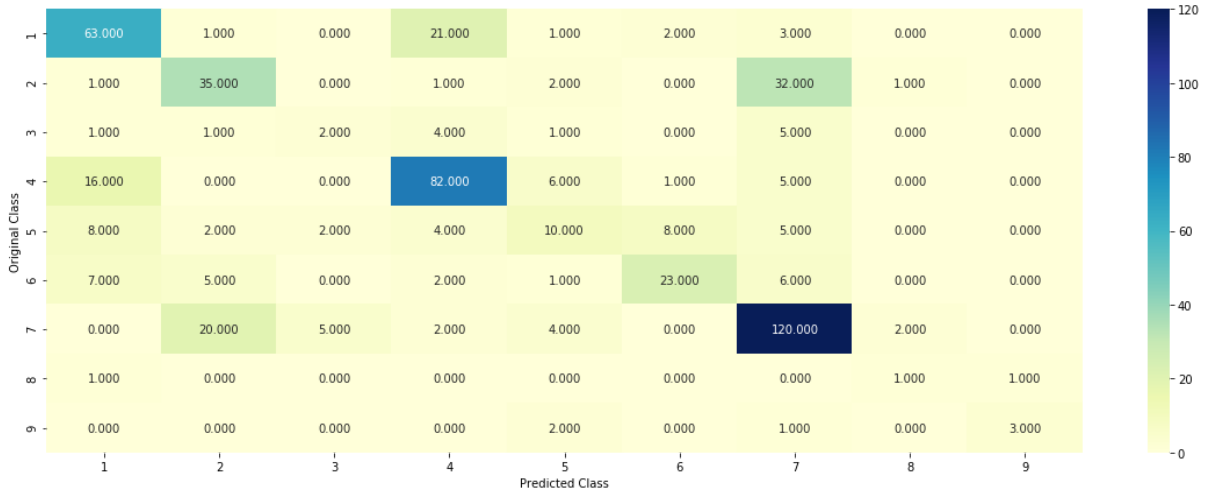



```
For values of best alpha =  11 The train log loss is: 0.6353401376086985
For values of best alpha =  11 The cross validation log loss is: 1.03751756
08550034
For values of best alpha =  11 The test log loss is: 1.0760286584291943
```

4.2.2. Testing the model with best hyper paramters

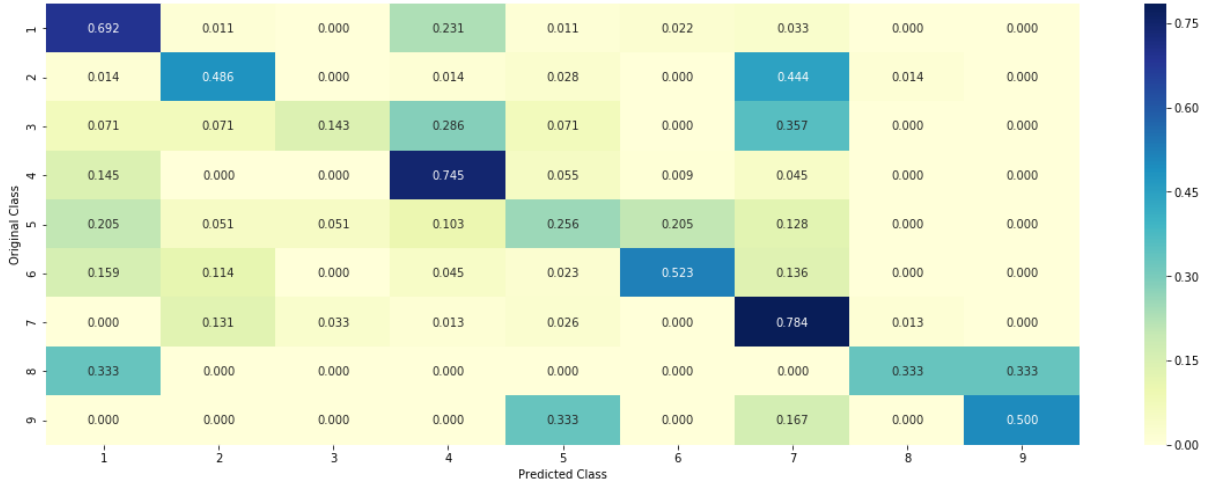
In [61]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.0375175608550034
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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

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



4.2.3.Sample Query point -1

In [62]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1,
```

```
-1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs
 to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 6
The  11  nearest neighbours of the test points belongs to classes [6 6 6 5
5 6 1 1 5 1 5]
Fequency of nearest points : Counter({6: 4, 5: 4, 1: 3})
```

4.2.4. Sample Query Point-2

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 = 100

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

```
Predicted Class : 8
Actual Class : 2
the k value for knn is 11 and the nearest neighbours of the test points bel
ongs to classes [7 7 7 8 1 2 1 1 7 6 7]
Fequency of nearest points : Counter({7: 5, 1: 3, 8: 1, 2: 1, 6: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

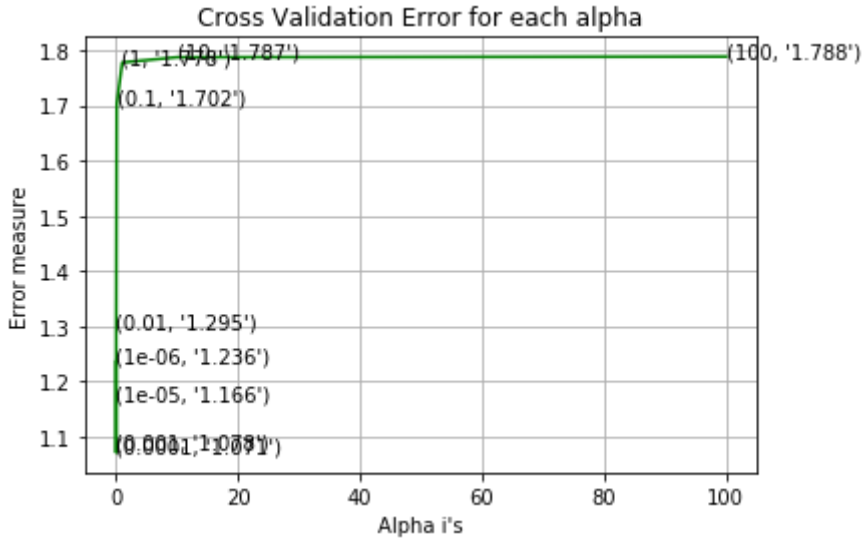
In [64]:

```
# 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-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2356915693638115
for alpha = 1e-05
Log Loss : 1.1662786582959581
for alpha = 0.0001
Log Loss : 1.0710923744441376
for alpha = 0.001
Log Loss : 1.0783210550174391
for alpha = 0.01
Log Loss : 1.2952233336658905
for alpha = 0.1
```

```
Log Loss : 1.7022183381376426
for alpha = 1
Log Loss : 1.777594793275901
for alpha = 10
Log Loss : 1.7871283136293215
for alpha = 100
Log Loss : 1.7882599811822977
```



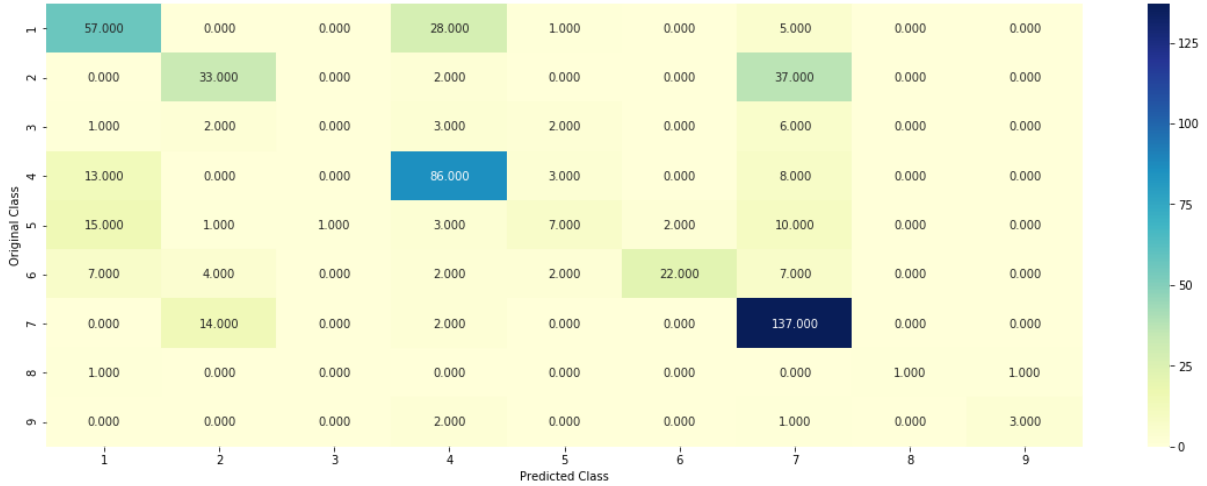

```
For values of best alpha =  0.0001 The train log loss is: 0.438434849047118
2
For values of best alpha =  0.0001 The cross validation log loss is: 1.0710
923744441376
For values of best alpha =  0.0001 The test log loss is: 1.0490000235338794
```

4.3.1.2. Testing the model with best hyper paramters

In [65]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
```

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

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

```
Log loss : 1.0710923744441376
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




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


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

4.3.1.3. Feature Importance

In [66]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
```

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

4.3.1.3.1. Correctly Classified point

In [67]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[4.690e-02 1.540e-02 2.000e-04 1.688e-01 1.
250e-02 7.135e-01 3.090e-02
  1.030e-02 1.500e-03]]
Actual Class : 6
--------------------------------------------------
70 Text feature [values] present in test data point [True]
95 Text feature [cycle] present in test data point [True]
100 Text feature [resistance] present in test data point [True]
101 Text feature [ubiquitin] present in test data point [True]
```

```
109 Text feature [basis] present in test data point [True]
112 Text feature [substitutions] present in test data point [True]
113 Text feature [substitution] present in test data point [True]
115 Text feature [concentration] present in test data point [True]
116 Text feature [interactions] present in test data point [True]
122 Text feature [57] present in test data point [True]
123 Text feature [conformation] present in test data point [True]
124 Text feature [apoptosis] present in test data point [True]
125 Text feature [showing] present in test data point [True]
134 Text feature [significant] present in test data point [True]
135 Text feature [inhibitors] present in test data point [True]
136 Text feature [site] present in test data point [True]
137 Text feature [considered] present in test data point [True]
149 Text feature [ic50] present in test data point [True]
156 Text feature [loss] present in test data point [True]
160 Text feature [population] present in test data point [True]
162 Text feature [substrate] present in test data point [True]
166 Text feature [predicted] present in test data point [True]
174 Text feature [nucleotide] present in test data point [True]
175 Text feature [studied] present in test data point [True]
182 Text feature [72] present in test data point [True]
184 Text feature [degradation] present in test data point [True]
189 Text feature [induce] present in test data point [True]
195 Text feature [helix] present in test data point [True]
197 Text feature [coding] present in test data point [True]
198 Text feature [frequent] present in test data point [True]
199 Text feature [free] present in test data point [True]
201 Text feature [phase] present in test data point [True]
205 Text feature [state] present in test data point [True]
208 Text feature [cause] present in test data point [True]
210 Text feature [associated] present in test data point [True]
211 Text feature [blue] present in test data point [True]
212 Text feature [enzyme] present in test data point [True]
213 Text feature [added] present in test data point [True]
215 Text feature [drug] present in test data point [True]
216 Text feature [s2] present in test data point [True]
218 Text feature [200] present in test data point [True]
221 Text feature [time] present in test data point [True]
224 Text feature [factors] present in test data point [True]
229 Text feature [change] present in test data point [True]
231 Text feature [42] present in test data point [True]
236 Text feature [including] present in test data point [True]
239 Text feature [identified] present in test data point [True]
240 Text feature [dose] present in test data point [True]
241 Text feature [single] present in test data point [True]
244 Text feature [affect] present in test data point [True]
```

```
246 Text feature [since] present in test data point [True]
247 Text feature [lower] present in test data point [True]
248 Text feature [atp] present in test data point [True]
251 Text feature [acids] present in test data point [True]
252 Text feature [selected] present in test data point [True]
253 Text feature [screening] present in test data point [True]
254 Text feature [signalling] present in test data point [True]
255 Text feature [red] present in test data point [True]
258 Text feature [resistant] present in test data point [True]
264 Text feature [nm] present in test data point [True]
266 Text feature [bind] present in test data point [True]
268 Text feature [gel] present in test data point [True]
271 Text feature [structural] present in test data point [True]
272 Text feature [information] present in test data point [True]
273 Text feature [family] present in test data point [True]
274 Text feature [showed] present in test data point [True]
278 Text feature [region] present in test data point [True]
291 Text feature [min] present in test data point [True]
302 Text feature [mm] present in test data point [True]
304 Text feature [direct] present in test data point [True]
305 Text feature [stability] present in test data point [True]
309 Text feature [myc] present in test data point [True]
311 Text feature [spectrum] present in test data point [True]
313 Text feature [concentrations] present in test data point [True]
317 Text feature [altered] present in test data point [True]
320 Text feature [green] present in test data point [True]
329 Text feature [four] present in test data point [True]
333 Text feature [transcriptional] present in test data point [True]
338 Text feature [development] present in test data point [True]
342 Text feature [60] present in test data point [True]
343 Text feature [factor] present in test data point [True]
346 Text feature [notably] present in test data point [True]
347 Text feature [thus] present in test data point [True]
359 Text feature [34] present in test data point [True]
365 Text feature [incubated] present in test data point [True]
368 Text feature [leading] present in test data point [True]
369 Text feature [bound] present in test data point [True]
370 Text feature [decreased] present in test data point [True]
373 Text feature [loop] present in test data point [True]
374 Text feature [amino] present in test data point [True]
375 Text feature [terminal] present in test data point [True]
376 Text feature [20] present in test data point [True]
377 Text feature [given] present in test data point [True]
379 Text feature [group] present in test data point [True]
380 Text feature [type] present in test data point [True]
386 Text feature [versus] present in test data point [True]
```

388 Text feature [malignant] present in test data point [True]
390 Text feature [44] present in test data point [True]
393 Text feature [40] present in test data point [True]
394 Text feature [important] present in test data point [True]
397 Text feature [investigated] present in test data point [True]
399 Text feature [mice] present in test data point [True]
402 Text feature [discussion] present in test data point [True]
404 Text feature [model] present in test data point [True]
410 Text feature [respectively] present in test data point [True]
411 Text feature [larger] present in test data point [True]
412 Text feature [value] present in test data point [True]
413 Text feature [non] present in test data point [True]
414 Text feature [five] present in test data point [True]
415 Text feature [buffer] present in test data point [True]
420 Text feature [reaction] present in test data point [True]
421 Text feature [pathways] present in test data point [True]
423 Text feature [regulated] present in test data point [True]
424 Text feature [members] present in test data point [True]
425 Text feature [play] present in test data point [True]
426 Text feature [likely] present in test data point [True]
427 Text feature [use] present in test data point [True]
429 Text feature [acid] present in test data point [True]
430 Text feature [residues] present in test data point [True]
431 Text feature [binding] present in test data point [True]
432 Text feature [frequency] present in test data point [True]
433 Text feature [changes] present in test data point [True]
434 Text feature [targeted] present in test data point [True]
436 Text feature [therefore] present in test data point [True]
438 Text feature [involved] present in test data point [True]
439 Text feature [occur] present in test data point [True]
441 Text feature [higher] present in test data point [True]
442 Text feature [15] present in test data point [True]
443 Text feature [52] present in test data point [True]
446 Text feature [structure] present in test data point [True]
447 Text feature [proteins] present in test data point [True]
449 Text feature [well] present in test data point [True]
450 Text feature [examined] present in test data point [True]
454 Text feature [particular] present in test data point [True]
455 Text feature [measured] present in test data point [True]
460 Text feature [fig] present in test data point [True]
461 Text feature [materials] present in test data point [True]
465 Text feature [sequence] present in test data point [True]
466 Text feature [progression] present in test data point [True]
467 Text feature [reduced] present in test data point [True]
469 Text feature [residue] present in test data point [True]
474 Text feature [calculated] present in test data point [True]

```
476 Text feature [methods] present in test data point [True]
477 Text feature [differences] present in test data point [True]
479 Text feature [second] present in test data point [True]
480 Text feature [developed] present in test data point [True]
482 Text feature [catalytic] present in test data point [True]
488 Text feature [regulation] present in test data point [True]
489 Text feature [defined] present in test data point [True]
492 Text feature [kinases] present in test data point [True]
493 Text feature [would] present in test data point [True]
Out of the top  500  features  151 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [68]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.360e-02 1.240e-02 1.000e-04 3.049e-01 2.
700e-03 0.000e+00 6.195e-01
  2.600e-02 7.000e-04]]
Actual Class : 2
--------------------------------------------------
6 Text feature [activation] present in test data point [True]
9 Text feature [downstream] present in test data point [True]
12 Text feature [constitutive] present in test data point [True]
13 Text feature [enhanced] present in test data point [True]
21 Text feature [activated] present in test data point [True]
25 Text feature [overexpression] present in test data point [True]
32 Text feature [bone] present in test data point [True]
48 Text feature [oncogenic] present in test data point [True]
55 Text feature [concentrations] present in test data point [True]
57 Text feature [insertion] present in test data point [True]
71 Text feature [3b] present in test data point [True]
73 Text feature [mapk] present in test data point [True]
90 Text feature [pathways] present in test data point [True]
```

```
96 Text feature [positive] present in test data point [True]
99 Text feature [inhibited] present in test data point [True]
100 Text feature [codon] present in test data point [True]
103 Text feature [transforming] present in test data point [True]
105 Text feature [ligand] present in test data point [True]
112 Text feature [transformed] present in test data point [True]
115 Text feature [activating] present in test data point [True]
131 Text feature [phospho] present in test data point [True]
180 Text feature [regulated] present in test data point [True]
216 Text feature [inhibitor] present in test data point [True]
223 Text feature [open] present in test data point [True]
225 Text feature [form] present in test data point [True]
232 Text feature [membrane] present in test data point [True]
239 Text feature [factor] present in test data point [True]
255 Text feature [2a] present in test data point [True]
273 Text feature [ph] present in test data point [True]
277 Text feature [activate] present in test data point [True]
288 Text feature [regions] present in test data point [True]
293 Text feature [signaling] present in test data point [True]
297 Text feature [expressing] present in test data point [True]
298 Text feature [distinct] present in test data point [True]
300 Text feature [culture] present in test data point [True]
303 Text feature [tissue] present in test data point [True]
305 Text feature [constitutively] present in test data point [True]
312 Text feature [stably] present in test data point [True]
344 Text feature [cells] present in test data point [True]
345 Text feature [phosphorylated] present in test data point [True]
348 Text feature [genomic] present in test data point [True]
360 Text feature [tumor] present in test data point [True]
370 Text feature [versus] present in test data point [True]
376 Text feature [occur] present in test data point [True]
381 Text feature [supplemental] present in test data point [True]
387 Text feature [derived] present in test data point [True]
393 Text feature [presence] present in test data point [True]
396 Text feature [2b] present in test data point [True]
403 Text feature [patient] present in test data point [True]
420 Text feature [download] present in test data point [True]
432 Text feature [mm] present in test data point [True]
435 Text feature [carried] present in test data point [True]
436 Text feature [fold] present in test data point [True]
444 Text feature [combination] present in test data point [True]
457 Text feature [gain] present in test data point [True]
458 Text feature [elevated] present in test data point [True]
459 Text feature [frequent] present in test data point [True]
460 Text feature [promote] present in test data point [True]
465 Text feature [blood] present in test data point [True]
```

```
469 Text feature [consistent] present in test data point [True]
477 Text feature [increased] present in test data point [True]
479 Text feature [mechanisms] present in test data point [True]
489 Text feature [fusion] present in test data point [True]
492 Text feature [cdna] present in test data point [True]
494 Text feature [position] present in test data point [True]
498 Text feature [like] present in test data point [True]
Out of the top  500  features  66 are present in query point
```

4.3.2. Without Class balancing

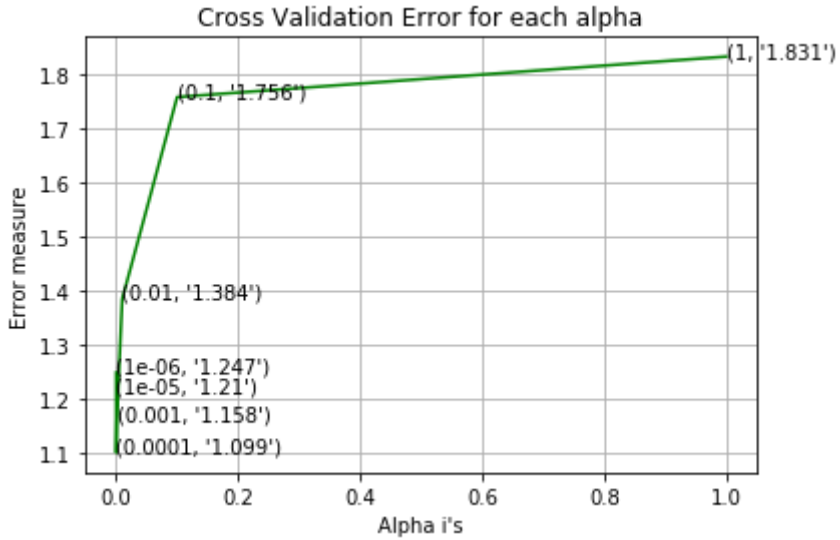
4.3.2.1. Hyper paramter tuning

In [69]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
```

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

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

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.2474700274759782
for alpha = 1e-05
Log Loss : 1.2101049504162749
for alpha = 0.0001
Log Loss : 1.0994614301859862
for alpha = 0.001
Log Loss : 1.1583193901751734
for alpha = 0.01
Log Loss : 1.3836739163454326
for alpha = 0.1
Log Loss : 1.756468189306298
for alpha = 1
Log Loss : 1.8312267244981817
```



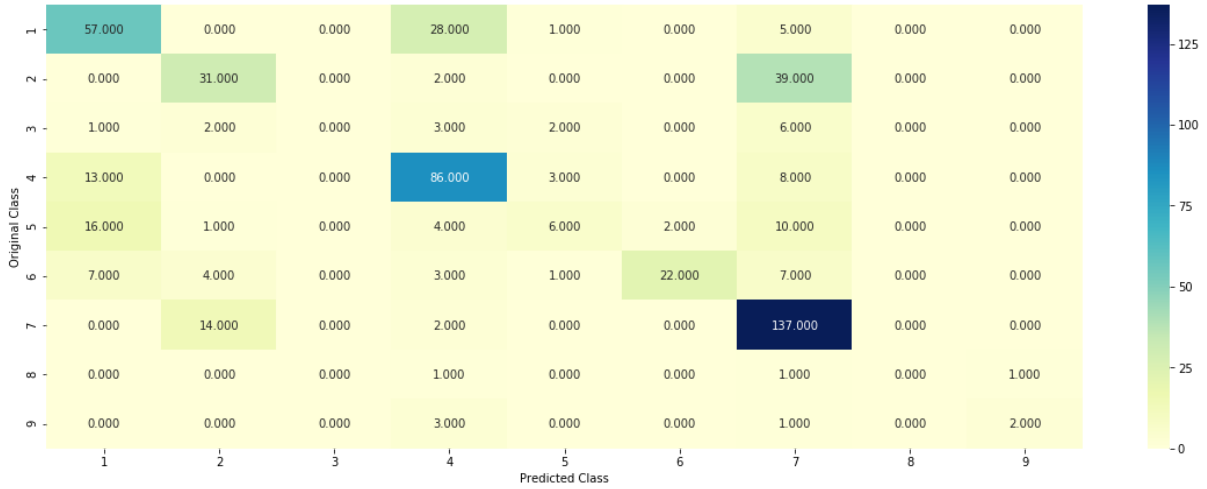

```
For values of best alpha =  0.0001 The train log loss is: 0.440294520978559
37
For values of best alpha =  0.0001 The cross validation log loss is: 1.0994
614301859862
For values of best alpha =  0.0001 The test log loss is: 1.074628594003273
```

4.3.2.2. Testing model with best hyper parameters

In [70]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
```

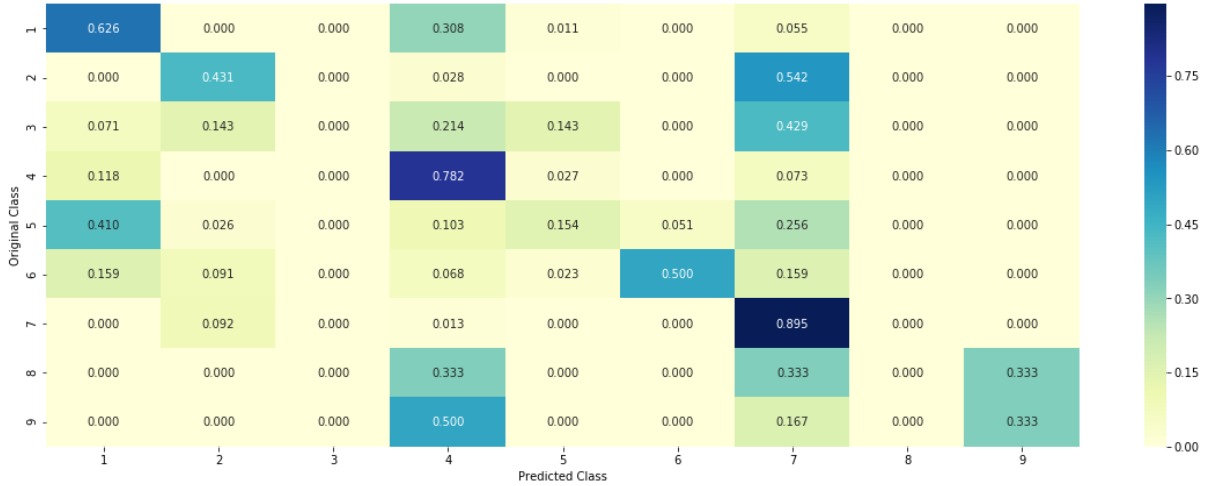
```
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.0994614301859862
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```




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

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



4.3.2.3. Feature Importance, Correctly Classified point

In [71]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
```

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

```
Predicted Class : 6
Predicted Class Probabilities: [[4.020e-02 1.650e-02 2.000e-04 1.992e-01 1.320e-02 6.717e-01 3.900e-02
  2.000e-02 1.000e-04]]
Actual Class : 6
--------------------------------------------------
68 Text feature [values] present in test data point [True]
95 Text feature [ubiquitin] present in test data point [True]
98 Text feature [resistance] present in test data point [True]
100 Text feature [cycle] present in test data point [True]
106 Text feature [basis] present in test data point [True]
107 Text feature [substitutions] present in test data point [True]
110 Text feature [substitution] present in test data point [True]
113 Text feature [concentration] present in test data point [True]
115 Text feature [57] present in test data point [True]
117 Text feature [interactions] present in test data point [True]
119 Text feature [conformation] present in test data point [True]
122 Text feature [showing] present in test data point [True]
124 Text feature [apoptosis] present in test data point [True]
128 Text feature [site] present in test data point [True]
130 Text feature [inhibitors] present in test data point [True]
132 Text feature [significant] present in test data point [True]
137 Text feature [considered] present in test data point [True]
151 Text feature [ic50] present in test data point [True]
164 Text feature [predicted] present in test data point [True]
167 Text feature [substrate] present in test data point [True]
168 Text feature [loss] present in test data point [True]
171 Text feature [population] present in test data point [True]
175 Text feature [studied] present in test data point [True]
180 Text feature [nucleotide] present in test data point [True]
184 Text feature [72] present in test data point [True]
191 Text feature [state] present in test data point [True]
192 Text feature [frequent] present in test data point [True]
193 Text feature [induce] present in test data point [True]
194 Text feature [helix] present in test data point [True]
195 Text feature [degradation] present in test data point [True]
203 Text feature [free] present in test data point [True]
204 Text feature [enzyme] present in test data point [True]
206 Text feature [coding] present in test data point [True]
208 Text feature [drug] present in test data point [True]
```

```
214 Text feature [phase] present in test data point [True]
215 Text feature [associated] present in test data point [True]
219 Text feature [cause] present in test data point [True]
220 Text feature [blue] present in test data point [True]
221 Text feature [s2] present in test data point [True]
225 Text feature [factors] present in test data point [True]
227 Text feature [change] present in test data point [True]
229 Text feature [added] present in test data point [True]
230 Text feature [42] present in test data point [True]
231 Text feature [time] present in test data point [True]
232 Text feature [200] present in test data point [True]
235 Text feature [dose] present in test data point [True]
241 Text feature [affect] present in test data point [True]
242 Text feature [atp] present in test data point [True]
247 Text feature [selected] present in test data point [True]
250 Text feature [lower] present in test data point [True]
251 Text feature [including] present in test data point [True]
254 Text feature [identified] present in test data point [True]
255 Text feature [since] present in test data point [True]
257 Text feature [single] present in test data point [True]
260 Text feature [nm] present in test data point [True]
261 Text feature [resistant] present in test data point [True]
263 Text feature [structural] present in test data point [True]
266 Text feature [bind] present in test data point [True]
268 Text feature [red] present in test data point [True]
269 Text feature [signalling] present in test data point [True]
271 Text feature [acids] present in test data point [True]
272 Text feature [screening] present in test data point [True]
275 Text feature [family] present in test data point [True]
290 Text feature [mm] present in test data point [True]
291 Text feature [information] present in test data point [True]
292 Text feature [gel] present in test data point [True]
293 Text feature [min] present in test data point [True]
297 Text feature [region] present in test data point [True]
305 Text feature [showed] present in test data point [True]
312 Text feature [altered] present in test data point [True]
316 Text feature [concentrations] present in test data point [True]
317 Text feature [spectrum] present in test data point [True]
321 Text feature [development] present in test data point [True]
324 Text feature [stability] present in test data point [True]
326 Text feature [myc] present in test data point [True]
332 Text feature [direct] present in test data point [True]
334 Text feature [green] present in test data point [True]
342 Text feature [four] present in test data point [True]
344 Text feature [60] present in test data point [True]
353 Text feature [transcriptional] present in test data point [True]
```

```
354 Text feature [factor] present in test data point [True]
355 Text feature [34] present in test data point [True]
359 Text feature [thus] present in test data point [True]
360 Text feature [bound] present in test data point [True]
361 Text feature [leading] present in test data point [True]
366 Text feature [notably] present in test data point [True]
367 Text feature [incubated] present in test data point [True]
369 Text feature [group] present in test data point [True]
371 Text feature [terminal] present in test data point [True]
375 Text feature [loop] present in test data point [True]
376 Text feature [given] present in test data point [True]
377 Text feature [versus] present in test data point [True]
380 Text feature [decreased] present in test data point [True]
381 Text feature [20] present in test data point [True]
383 Text feature [44] present in test data point [True]
384 Text feature [amino] present in test data point [True]
390 Text feature [40] present in test data point [True]
394 Text feature [respectively] present in test data point [True]
395 Text feature [malignant] present in test data point [True]
398 Text feature [important] present in test data point [True]
401 Text feature [changes] present in test data point [True]
403 Text feature [use] present in test data point [True]
405 Text feature [mice] present in test data point [True]
406 Text feature [pathways] present in test data point [True]
408 Text feature [binding] present in test data point [True]
409 Text feature [non] present in test data point [True]
412 Text feature [reaction] present in test data point [True]
413 Text feature [52] present in test data point [True]
414 Text feature [model] present in test data point [True]
416 Text feature [likely] present in test data point [True]
417 Text feature [five] present in test data point [True]
418 Text feature [type] present in test data point [True]
419 Text feature [larger] present in test data point [True]
420 Text feature [value] present in test data point [True]
421 Text feature [investigated] present in test data point [True]
423 Text feature [buffer] present in test data point [True]
425 Text feature [residues] present in test data point [True]
426 Text feature [frequency] present in test data point [True]
429 Text feature [members] present in test data point [True]
430 Text feature [targeted] present in test data point [True]
431 Text feature [occur] present in test data point [True]
432 Text feature [discussion] present in test data point [True]
436 Text feature [structure] present in test data point [True]
437 Text feature [therefore] present in test data point [True]
438 Text feature [play] present in test data point [True]
441 Text feature [particular] present in test data point [True]
```

```
443 Text feature [acid] present in test data point [True]
444 Text feature [higher] present in test data point [True]
446 Text feature [regulated] present in test data point [True]
448 Text feature [involved] present in test data point [True]
449 Text feature [developed] present in test data point [True]
450 Text feature [well] present in test data point [True]
451 Text feature [15] present in test data point [True]
452 Text feature [progression] present in test data point [True]
460 Text feature [examined] present in test data point [True]
464 Text feature [residue] present in test data point [True]
466 Text feature [would] present in test data point [True]
469 Text feature [sequence] present in test data point [True]
470 Text feature [calculated] present in test data point [True]
471 Text feature [fig] present in test data point [True]
473 Text feature [differences] present in test data point [True]
475 Text feature [methods] present in test data point [True]
476 Text feature [reduced] present in test data point [True]
477 Text feature [proteins] present in test data point [True]
478 Text feature [second] present in test data point [True]
479 Text feature [database] present in test data point [True]
480 Text feature [materials] present in test data point [True]
490 Text feature [catalytic] present in test data point [True]
492 Text feature [kinases] present in test data point [True]
495 Text feature [regulation] present in test data point [True]
498 Text feature [measured] present in test data point [True]
Out of the top  500  features  151 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [72]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.800e-02 8.600e-03 1.000e-04 3.008e-01 1.
```

```
300e-03 0.000e+00 5.893e-01
  7.190e-02 0.000e+00]]
Actual Class : 2
--------------------------------------------------
11 Text feature [downstream] present in test data point [True]
14 Text feature [activation] present in test data point [True]
18 Text feature [enhanced] present in test data point [True]
29 Text feature [bone] present in test data point [True]
37 Text feature [constitutive] present in test data point [True]
43 Text feature [activated] present in test data point [True]
45 Text feature [overexpression] present in test data point [True]
53 Text feature [concentrations] present in test data point [True]
70 Text feature [insertion] present in test data point [True]
76 Text feature [3b] present in test data point [True]
100 Text feature [inhibited] present in test data point [True]
102 Text feature [positive] present in test data point [True]
111 Text feature [oncogenic] present in test data point [True]
149 Text feature [activating] present in test data point [True]
151 Text feature [ligand] present in test data point [True]
158 Text feature [codon] present in test data point [True]
164 Text feature [pathways] present in test data point [True]
192 Text feature [transforming] present in test data point [True]
200 Text feature [mapk] present in test data point [True]
210 Text feature [membrane] present in test data point [True]
230 Text feature [transformed] present in test data point [True]
231 Text feature [phospho] present in test data point [True]
240 Text feature [inhibitor] present in test data point [True]
241 Text feature [form] present in test data point [True]
250 Text feature [regions] present in test data point [True]
254 Text feature [regulated] present in test data point [True]
278 Text feature [phosphorylated] present in test data point [True]
280 Text feature [factor] present in test data point [True]
282 Text feature [ph] present in test data point [True]
287 Text feature [open] present in test data point [True]
289 Text feature [distinct] present in test data point [True]
297 Text feature [2a] present in test data point [True]
310 Text feature [tissue] present in test data point [True]
322 Text feature [presence] present in test data point [True]
342 Text feature [expressing] present in test data point [True]
345 Text feature [tumor] present in test data point [True]
358 Text feature [versus] present in test data point [True]
364 Text feature [genomic] present in test data point [True]
371 Text feature [mm] present in test data point [True]
372 Text feature [consistent] present in test data point [True]
380 Text feature [derived] present in test data point [True]
389 Text feature [2b] present in test data point [True]
```

```
400 Text feature [patient] present in test data point [True]
401 Text feature [culture] present in test data point [True]
405 Text feature [carried] present in test data point [True]
413 Text feature [occur] present in test data point [True]
415 Text feature [cells] present in test data point [True]
419 Text feature [signaling] present in test data point [True]
427 Text feature [like] present in test data point [True]
434 Text feature [increased] present in test data point [True]
436 Text feature [fusion] present in test data point [True]
439 Text feature [additional] present in test data point [True]
441 Text feature [stably] present in test data point [True]
445 Text feature [elevated] present in test data point [True]
446 Text feature [position] present in test data point [True]
454 Text feature [activate] present in test data point [True]
459 Text feature [mechanisms] present in test data point [True]
465 Text feature [provided] present in test data point [True]
474 Text feature [right] present in test data point [True]
483 Text feature [cdna] present in test data point [True]
494 Text feature [comparison] present in test data point [True]
496 Text feature [gain] present in test data point [True]
498 Text feature [receptors] present in test data point [True]
499 Text feature [frequent] present in test data point [True]
Out of the top  500  features  64 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

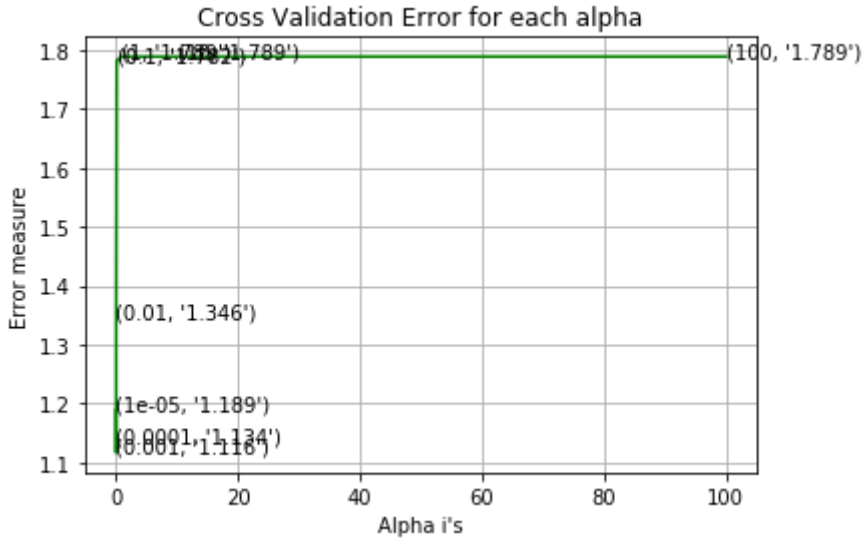
In [73]:

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

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

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

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# ------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/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(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for C = 1e-05
Log Loss : 1.1889934615847695
for C = 0.0001
Log Loss : 1.1343308508201937
for C = 0.001
Log Loss : 1.1164589559909155
for C = 0.01
Log Loss : 1.3463273699296021
for C = 0.1
Log Loss : 1.7817201202177981
for C = 1
Log Loss : 1.7885233760588644
for C = 10
Log Loss : 1.788523221374913
for C = 100
Log Loss : 1.7885232776775752
```




```
For values of best alpha =  0.001 The train log loss is: 0.5824178672027392
For values of best alpha =  0.001 The cross validation log loss is: 1.11645
89559909155
For values of best alpha =  0.001 The test log loss is: 1.1294853457298575
```

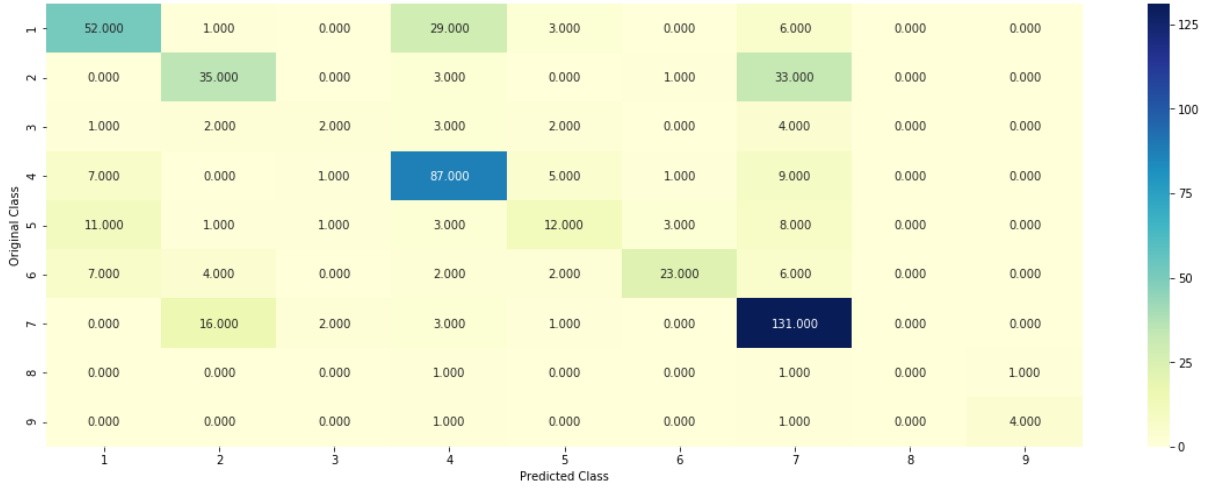
4.4.2. Testing model with best hyper parameters

In [74]:

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

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

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

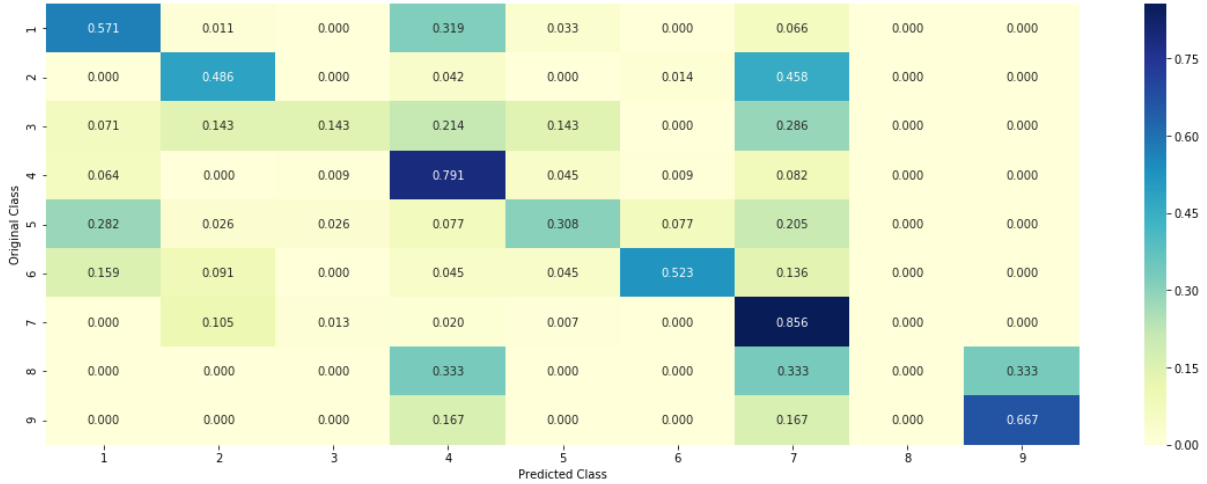
```
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight
='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)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

```
Log loss : 1.1164589559909155
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




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


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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [75]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0426 0.0204 0.0026 0.2099 0.009  0.6627
0.042  0.0081 0.0028]]
Actual Class : 6
--------------------------------------------------
3 Text feature [ubiquitin] present in test data point [True]
4 Text feature [substitutions] present in test data point [True]
```

```
7 Text feature [substitution] present in test data point [True]
9 Text feature [values] present in test data point [True]
12 Text feature [showing] present in test data point [True]
106 Text feature [predicted] present in test data point [True]
107 Text feature [site] present in test data point [True]
108 Text feature [resistance] present in test data point [True]
109 Text feature [57] present in test data point [True]
115 Text feature [loss] present in test data point [True]
116 Text feature [basis] present in test data point [True]
117 Text feature [interactions] present in test data point [True]
119 Text feature [substrate] present in test data point [True]
120 Text feature [conformation] present in test data point [True]
151 Text feature [ic50] present in test data point [True]
156 Text feature [concentration] present in test data point [True]
157 Text feature [enzyme] present in test data point [True]
158 Text feature [significant] present in test data point [True]
162 Text feature [studied] present in test data point [True]
163 Text feature [atp] present in test data point [True]
166 Text feature [resistant] present in test data point [True]
183 Text feature [free] present in test data point [True]
188 Text feature [cause] present in test data point [True]
189 Text feature [considered] present in test data point [True]
190 Text feature [inhibitors] present in test data point [True]
193 Text feature [coding] present in test data point [True]
194 Text feature [dose] present in test data point [True]
196 Text feature [drug] present in test data point [True]
198 Text feature [acids] present in test data point [True]
199 Text feature [population] present in test data point [True]
201 Text feature [single] present in test data point [True]
203 Text feature [affect] present in test data point [True]
206 Text feature [cycle] present in test data point [True]
207 Text feature [frequent] present in test data point [True]
208 Text feature [state] present in test data point [True]
243 Text feature [would] present in test data point [True]
244 Text feature [associated] present in test data point [True]
245 Text feature [likely] present in test data point [True]
246 Text feature [s2] present in test data point [True]
249 Text feature [family] present in test data point [True]
251 Text feature [72] present in test data point [True]
254 Text feature [identified] present in test data point [True]
256 Text feature [apoptosis] present in test data point [True]
257 Text feature [screening] present in test data point [True]
259 Text feature [altered] present in test data point [True]
260 Text feature [model] present in test data point [True]
261 Text feature [lower] present in test data point [True]
263 Text feature [amino] present in test data point [True]
```

```
264 Text feature [nucleotide] present in test data point [True]
271 Text feature [thus] present in test data point [True]
273 Text feature [given] present in test data point [True]
274 Text feature [42] present in test data point [True]
275 Text feature [changes] present in test data point [True]
276 Text feature [use] present in test data point [True]
280 Text feature [group] present in test data point [True]
281 Text feature [spectrum] present in test data point [True]
282 Text feature [binding] present in test data point [True]
283 Text feature [factors] present in test data point [True]
288 Text feature [tumors] present in test data point [True]
289 Text feature [examined] present in test data point [True]
290 Text feature [stability] present in test data point [True]
294 Text feature [acid] present in test data point [True]
296 Text feature [four] present in test data point [True]
300 Text feature [bind] present in test data point [True]
302 Text feature [direct] present in test data point [True]
305 Text feature [occur] present in test data point [True]
307 Text feature [mm] present in test data point [True]
310 Text feature [signalling] present in test data point [True]
311 Text feature [34] present in test data point [True]
312 Text feature [factor] present in test data point [True]
313 Text feature [concentrations] present in test data point [True]
314 Text feature [frequency] present in test data point [True]
315 Text feature [carried] present in test data point [True]
316 Text feature [showed] present in test data point [True]
317 Text feature [structural] present in test data point [True]
319 Text feature [60] present in test data point [True]
320 Text feature [blue] present in test data point [True]
322 Text feature [44] present in test data point [True]
323 Text feature [since] present in test data point [True]
327 Text feature [20] present in test data point [True]
346 Text feature [added] present in test data point [True]
348 Text feature [23] present in test data point [True]
350 Text feature [min] present in test data point [True]
352 Text feature [affinity] present in test data point [True]
353 Text feature [helix] present in test data point [True]
355 Text feature [reaction] present in test data point [True]
356 Text feature [selected] present in test data point [True]
357 Text feature [time] present in test data point [True]
359 Text feature [mice] present in test data point [True]
360 Text feature [tumor] present in test data point [True]
361 Text feature [many] present in test data point [True]
364 Text feature [developed] present in test data point [True]
365 Text feature [development] present in test data point [True]
366 Text feature [nm] present in test data point [True]
```

```
367 Text feature [kinase] present in test data point [True]
370 Text feature [non] present in test data point [True]
372 Text feature [including] present in test data point [True]
373 Text feature [buffer] present in test data point [True]
375 Text feature [therefore] present in test data point [True]
376 Text feature [information] present in test data point [True]
378 Text feature [leading] present in test data point [True]
379 Text feature [versus] present in test data point [True]
380 Text feature [52] present in test data point [True]
382 Text feature [database] present in test data point [True]
383 Text feature [green] present in test data point [True]
385 Text feature [45] present in test data point [True]
386 Text feature [five] present in test data point [True]
387 Text feature [degradation] present in test data point [True]
392 Text feature [red] present in test data point [True]
393 Text feature [40] present in test data point [True]
395 Text feature [15] present in test data point [True]
398 Text feature [change] present in test data point [True]
399 Text feature [reduced] present in test data point [True]
400 Text feature [sequence] present in test data point [True]
403 Text feature [targeted] present in test data point [True]
404 Text feature [fig] present in test data point [True]
405 Text feature [gel] present in test data point [True]
408 Text feature [35] present in test data point [True]
409 Text feature [induce] present in test data point [True]
412 Text feature [differences] present in test data point [True]
414 Text feature [catalytic] present in test data point [True]
417 Text feature [200] present in test data point [True]
427 Text feature [value] present in test data point [True]
429 Text feature [normal] present in test data point [True]
430 Text feature [disease] present in test data point [True]
431 Text feature [set] present in test data point [True]
434 Text feature [21] present in test data point [True]
435 Text feature [01] present in test data point [True]
439 Text feature [distribution] present in test data point [True]
441 Text feature [26] present in test data point [True]
442 Text feature [relevant] present in test data point [True]
443 Text feature [studies] present in test data point [True]
445 Text feature [subset] present in test data point [True]
447 Text feature [loop] present in test data point [True]
450 Text feature [image] present in test data point [True]
451 Text feature [well] present in test data point [True]
453 Text feature [malignant] present in test data point [True]
455 Text feature [pathways] present in test data point [True]
458 Text feature [phase] present in test data point [True]
459 Text feature [myc] present in test data point [True]
```

```
460 Text feature [bound] present in test data point [True]
461 Text feature [play] present in test data point [True]
463 Text feature [27] present in test data point [True]
464 Text feature [incubated] present in test data point [True]
465 Text feature [overall] present in test data point [True]
468 Text feature [invitrogen] present in test data point [True]
471 Text feature [ratio] present in test data point [True]
472 Text feature [investigated] present in test data point [True]
473 Text feature [methods] present in test data point [True]
474 Text feature [least] present in test data point [True]
475 Text feature [purified] present in test data point [True]
479 Text feature [study] present in test data point [True]
483 Text feature [31] present in test data point [True]
484 Text feature [mg] present in test data point [True]
487 Text feature [form] present in test data point [True]
490 Text feature [decreased] present in test data point [True]
491 Text feature [proteins] present in test data point [True]
495 Text feature [upon] present in test data point [True]
496 Text feature [25] present in test data point [True]
498 Text feature [two] present in test data point [True]
Out of the top  500  features  160 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [76]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[5.560e-02 1.530e-02 4.700e-03 3.937e-01 6.
400e-03 2.000e-04 5.078e-01
  1.440e-02 2.000e-03]]
Actual Class : 2
--------------------------------------------------
11 Text feature [activation] present in test data point [True]
12 Text feature [constitutive] present in test data point [True]
```

19 Text feature [enhanced] present in test data point [True]
28 Text feature [activated] present in test data point [True]
29 Text feature [downstream] present in test data point [True]
34 Text feature [concentrations] present in test data point [True]
35 Text feature [bone] present in test data point [True]
39 Text feature [ligand] present in test data point [True]
41 Text feature [mapk] present in test data point [True]
43 Text feature [inhibited] present in test data point [True]
54 Text feature [oncogenic] present in test data point [True]
57 Text feature [insertion] present in test data point [True]
58 Text feature [overexpression] present in test data point [True]
59 Text feature [3b] present in test data point [True]
63 Text feature [expressing] present in test data point [True]
66 Text feature [activate] present in test data point [True]
69 Text feature [supplemental] present in test data point [True]
70 Text feature [codon] present in test data point [True]
72 Text feature [positive] present in test data point [True]
73 Text feature [constitutively] present in test data point [True]
75 Text feature [pathways] present in test data point [True]
76 Text feature [activating] present in test data point [True]
77 Text feature [inhibitor] present in test data point [True]
78 Text feature [factor] present in test data point [True]
79 Text feature [2b] present in test data point [True]
80 Text feature [ph] present in test data point [True]
238 Text feature [phospho] present in test data point [True]
239 Text feature [membrane] present in test data point [True]
242 Text feature [form] present in test data point [True]
243 Text feature [phosphorylated] present in test data point [True]
244 Text feature [regulated] present in test data point [True]
247 Text feature [presence] present in test data point [True]
249 Text feature [2a] present in test data point [True]
250 Text feature [cells] present in test data point [True]
251 Text feature [stably] present in test data point [True]
253 Text feature [signaling] present in test data point [True]
254 Text feature [fold] present in test data point [True]
255 Text feature [download] present in test data point [True]
257 Text feature [open] present in test data point [True]
258 Text feature [combination] present in test data point [True]
259 Text feature [derived] present in test data point [True]
260 Text feature [interface] present in test data point [True]
261 Text feature [elevated] present in test data point [True]
263 Text feature [transformed] present in test data point [True]
266 Text feature [tumor] present in test data point [True]
267 Text feature [increased] present in test data point [True]
424 Text feature [additional] present in test data point [True]
426 Text feature [transforming] present in test data point [True]

```
427 Text feature [culture] present in test data point [True]
428 Text feature [patient] present in test data point [True]
434 Text feature [fusion] present in test data point [True]
435 Text feature [tissue] present in test data point [True]
436 Text feature [occur] present in test data point [True]
437 Text feature [promote] present in test data point [True]
438 Text feature [increase] present in test data point [True]
439 Text feature [signals] present in test data point [True]
440 Text feature [versus] present in test data point [True]
447 Text feature [gain] present in test data point [True]
449 Text feature [lead] present in test data point [True]
451 Text feature [distinct] present in test data point [True]
453 Text feature [medium] present in test data point [True]
454 Text feature [signal] present in test data point [True]
469 Text feature [hours] present in test data point [True]
473 Text feature [regions] present in test data point [True]
474 Text feature [caused] present in test data point [True]
475 Text feature [like] present in test data point [True]
476 Text feature [genomic] present in test data point [True]
478 Text feature [blood] present in test data point [True]
479 Text feature [met] present in test data point [True]
481 Text feature [vector] present in test data point [True]
483 Text feature [cdna] present in test data point [True]
485 Text feature [position] present in test data point [True]
486 Text feature [stem] present in test data point [True]
487 Text feature [five] present in test data point [True]
490 Text feature [provided] present in test data point [True]
494 Text feature [include] present in test data point [True]
495 Text feature [24] present in test data point [True]
496 Text feature [phosphorylation] present in test data point [True]
Out of the top  500  features  78 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [77]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
```

```
tate=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_dep
th=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
```

```
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cla
sses_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.207567831078782
for n_estimators = 100 and max depth =  10
Log Loss : 1.2479658558454154
for n_estimators = 200 and max depth =  5
Log Loss : 1.2058606980238056
for n_estimators = 200 and max depth =  10
Log Loss : 1.2387098933139358
for n_estimators = 500 and max depth =  5
Log Loss : 1.1965245064675267
for n_estimators = 500 and max depth =  10
```

```
Log Loss : 1.2314807381271955
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1942447739953799
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2316620631134867
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1934679191607922
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2311596449092104
For values of best estimator =  2000 The train log loss is: 0.8622799016789
644
For values of best estimator =  2000 The cross validation log loss is: 1.19
34679191607922
For values of best estimator =  2000 The test log loss is: 1.23528107520100
04
```

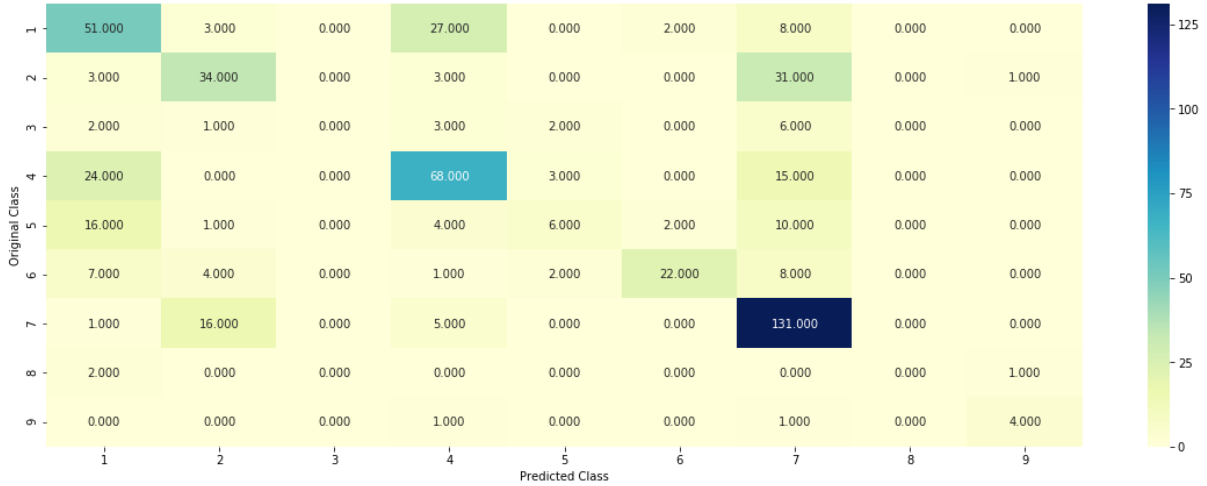
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [78]:

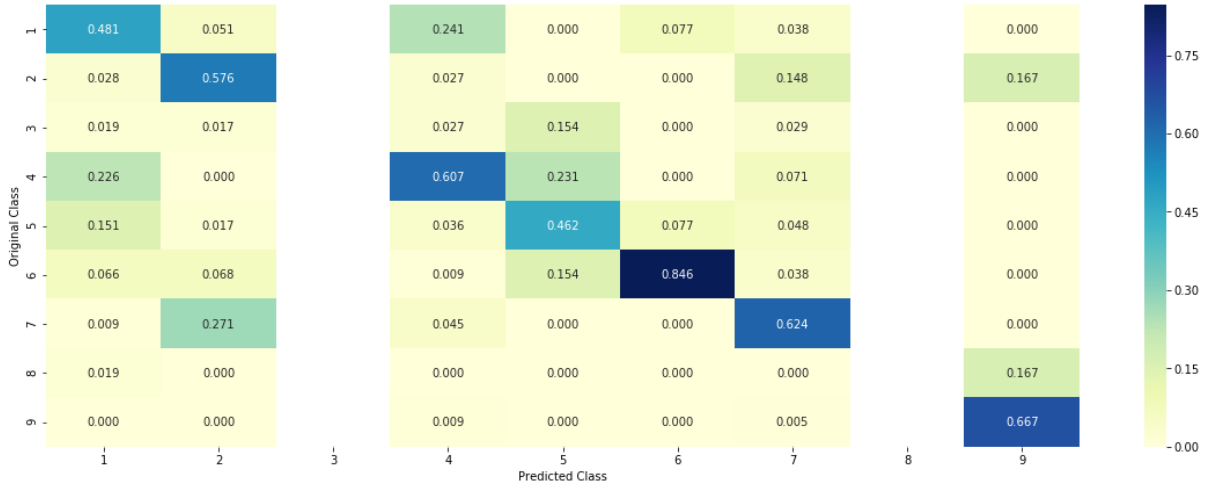
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
'gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

```
Log loss : 1.1934679191607922
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```




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




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

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [79]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion=
'gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2382 0.1174 0.0219 0.2603 0.0856 0.1205
0.1289 0.0093 0.0178]]
Actual Class : 6
```

```
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
8 Text feature [loss] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
11 Text feature [function] present in test data point [True]
13 Text feature [inhibitor] present in test data point [True]
21 Text feature [cells] present in test data point [True]
24 Text feature [growth] present in test data point [True]
25 Text feature [protein] present in test data point [True]
28 Text feature [functional] present in test data point [True]
29 Text feature [kinases] present in test data point [True]
31 Text feature [expression] present in test data point [True]
35 Text feature [variants] present in test data point [True]
38 Text feature [cell] present in test data point [True]
39 Text feature [proteins] present in test data point [True]
40 Text feature [stability] present in test data point [True]
46 Text feature [57] present in test data point [True]
59 Text feature [drug] present in test data point [True]
61 Text feature [oncogene] present in test data point [True]
66 Text feature [predicted] present in test data point [True]
69 Text feature [functions] present in test data point [True]
70 Text feature [use] present in test data point [True]
71 Text feature [resistance] present in test data point [True]
72 Text feature [response] present in test data point [True]
78 Text feature [activity] present in test data point [True]
83 Text feature [ic50] present in test data point [True]
91 Text feature [assays] present in test data point [True]
99 Text feature [information] present in test data point [True]
Out of the top  100  features  30 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [80]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1355 0.3021 0.016  0.0618 0.0545 0.044
0.3435 0.0138 0.0288]]
Actuall Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [tyrosine] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
8 Text feature [loss] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [function] present in test data point [True]
12 Text feature [therapy] present in test data point [True]
13 Text feature [inhibitor] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
16 Text feature [missense] present in test data point [True]
20 Text feature [trials] present in test data point [True]
21 Text feature [cells] present in test data point [True]
22 Text feature [constitutively] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [growth] present in test data point [True]
25 Text feature [protein] present in test data point [True]
28 Text feature [functional] present in test data point [True]
29 Text feature [kinases] present in test data point [True]
31 Text feature [expression] present in test data point [True]
32 Text feature [receptor] present in test data point [True]
33 Text feature [defective] present in test data point [True]
35 Text feature [variants] present in test data point [True]
38 Text feature [cell] present in test data point [True]
39 Text feature [proteins] present in test data point [True]
42 Text feature [inhibited] present in test data point [True]
43 Text feature [months] present in test data point [True]
44 Text feature [activate] present in test data point [True]
45 Text feature [ring] present in test data point [True]
46 Text feature [57] present in test data point [True]
48 Text feature [treated] present in test data point [True]
53 Text feature [therapeutic] present in test data point [True]
54 Text feature [phosphatase] present in test data point [True]
57 Text feature [inhibition] present in test data point [True]
```

```
58 Text feature [clinical] present in test data point [True]
62 Text feature [transforming] present in test data point [True]
63 Text feature [odds] present in test data point [True]
66 Text feature [predicted] present in test data point [True]
67 Text feature [patients] present in test data point [True]
69 Text feature [functions] present in test data point [True]
70 Text feature [use] present in test data point [True]
72 Text feature [response] present in test data point [True]
74 Text feature [proliferation] present in test data point [True]
75 Text feature [downstream] present in test data point [True]
76 Text feature [survival] present in test data point [True]
77 Text feature [potential] present in test data point [True]
78 Text feature [activity] present in test data point [True]
82 Text feature [sensitivity] present in test data point [True]
86 Text feature [expected] present in test data point [True]
88 Text feature [mapk] present in test data point [True]
89 Text feature [amplification] present in test data point [True]
90 Text feature [lines] present in test data point [True]
91 Text feature [assays] present in test data point [True]
93 Text feature [serum] present in test data point [True]
94 Text feature [expressing] present in test data point [True]
96 Text feature [ligand] present in test data point [True]
97 Text feature [combined] present in test data point [True]
Out of the top  100  features  61 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [81]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_dep
th=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cla
sses_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
```

```
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],c
v_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.236170471547618
for n_estimators = 10 and max depth =  3
Log Loss : 1.7534342040527369
for n_estimators = 10 and max depth =  5
Log Loss : 1.2716448818956851
for n_estimators = 10 and max depth =  10
Log Loss : 1.6307274795870264
for n_estimators = 50 and max depth =  2
Log Loss : 1.6770478435832084
for n_estimators = 50 and max depth =  3
Log Loss : 1.450881821110235
for n_estimators = 50 and max depth =  5
Log Loss : 1.298583829870214
for n_estimators = 50 and max depth =  10
Log Loss : 1.7090895634455723
for n_estimators = 100 and max depth =  2
Log Loss : 1.5959950596729344
for n_estimators = 100 and max depth =  3
Log Loss : 1.4640468383091543
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.3208540062485346
for n_estimators = 100 and max depth =  10
Log Loss : 1.6971194105822434
for n_estimators = 200 and max depth =  2
Log Loss : 1.6121792093552134
for n_estimators = 200 and max depth =  3
Log Loss : 1.4613898377386907
for n_estimators = 200 and max depth =  5
Log Loss : 1.3851159557408224
for n_estimators = 200 and max depth =  10
Log Loss : 1.686100699982839
for n_estimators = 500 and max depth =  2
Log Loss : 1.650160072988265
for n_estimators = 500 and max depth =  3
Log Loss : 1.5007317248550494
for n_estimators = 500 and max depth =  5
Log Loss : 1.4040914589775997
for n_estimators = 500 and max depth =  10
Log Loss : 1.7001228750826338
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6293862536457588
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5152221245311412
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3976126781406806
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7035652677428972
For values of best alpha =  10 The train log loss is: 0.06924734603949519
For values of best alpha =  10 The cross validation log loss is: 1.27164488
18956851
For values of best alpha =  10 The test log loss is: 1.3528001967251566
```

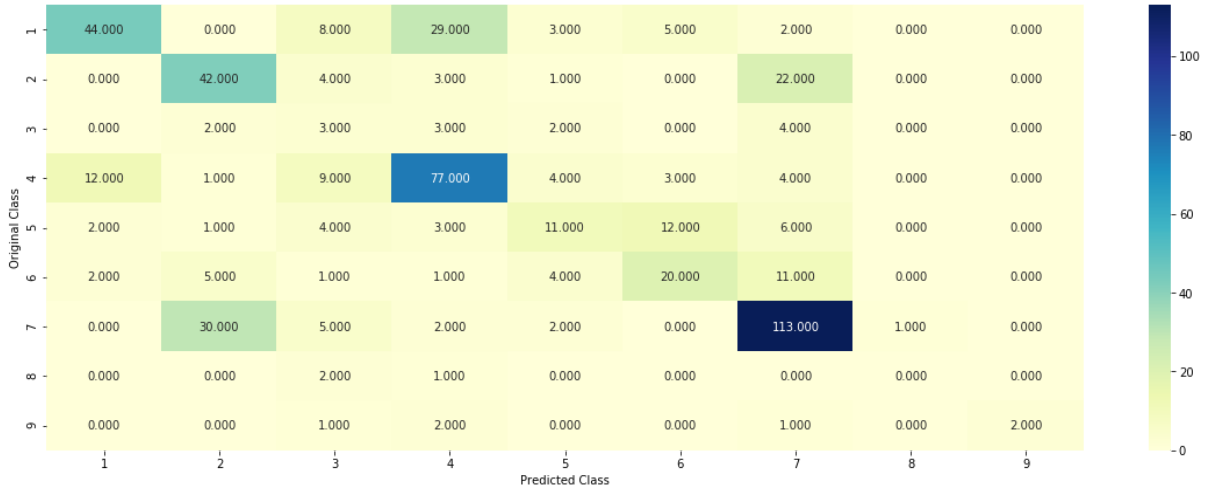
4.5.4. Testing model with best hyper parameters (Response Coding)

In [82]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=None, verbose=0, warm_start=False,
# class_weight=None)
```

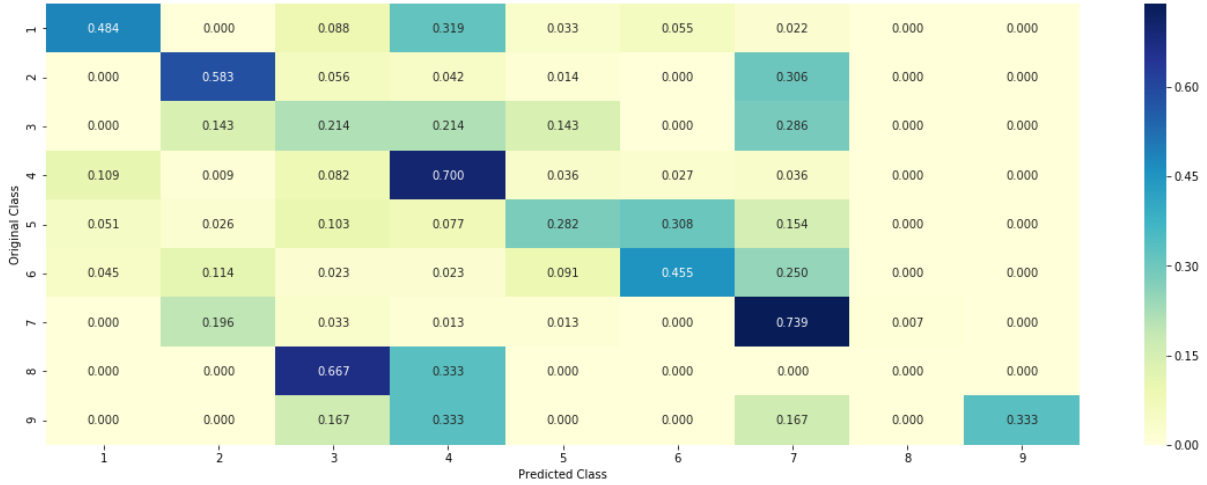
```
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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

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

```
Log loss : 1.2716448818956851
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```




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

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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [83]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion=
'gini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)
```

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2092 0.0312 0.1378 0.1006 0.2208 0.25
0.0108 0.0207 0.0188]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

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

4.5.5.2. Incorrectly Classified point

In [84]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0787 0.1289 0.2543 0.1282 0.0871 0.0611
0.0782 0.1093 0.0742]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Variation is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [85]:

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

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

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


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

```
 probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_func
tion_shape='ovr', random_state=None)

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


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

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

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

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


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='bala
nced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
```

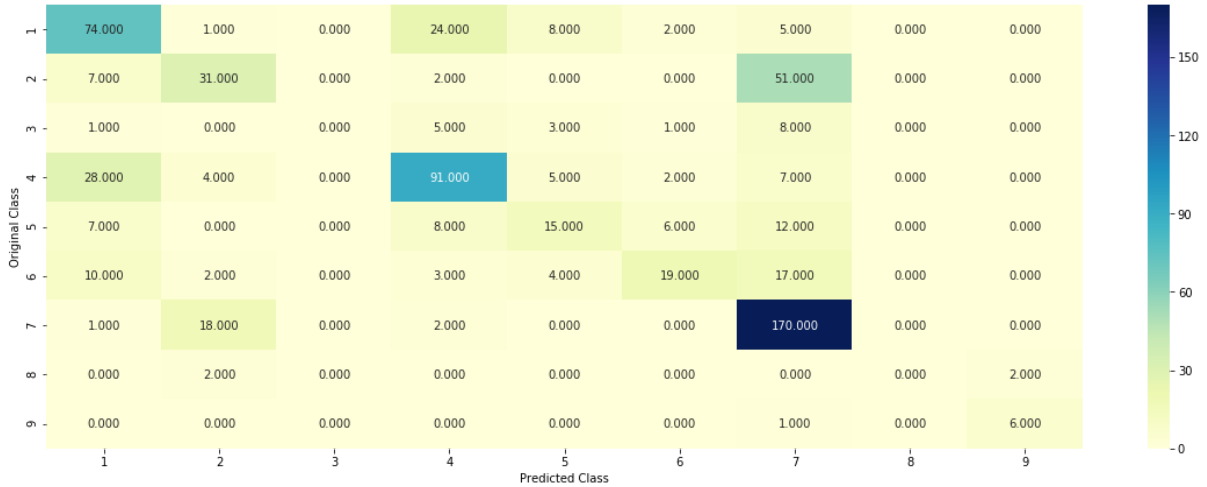
```
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

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

```
Logistic Regression :  Log Loss: 1.08
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.16
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.494
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.147
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.329
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.734
```

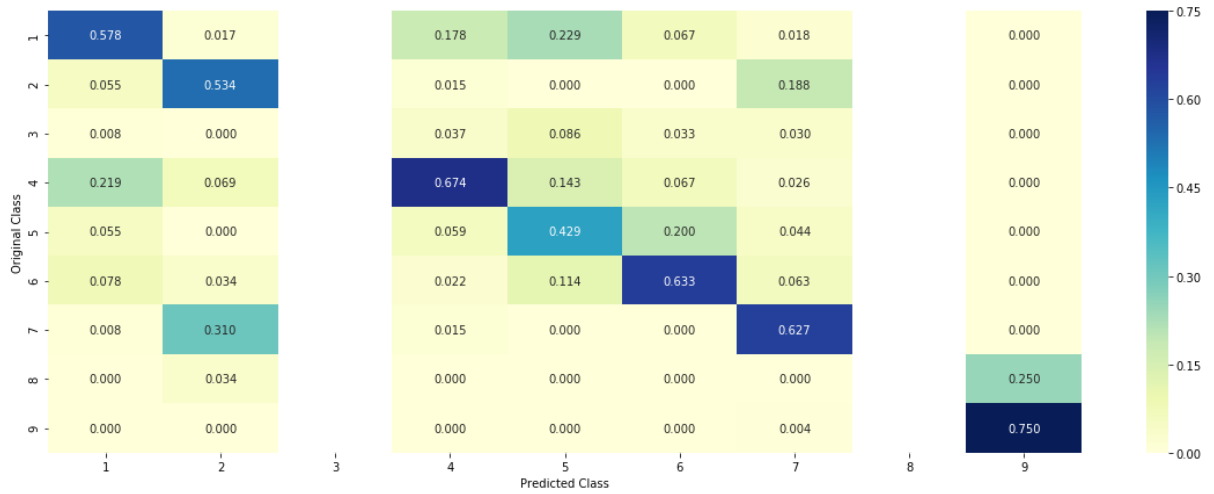
4.7.2 testing the model with the best hyper parameters

In [86]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

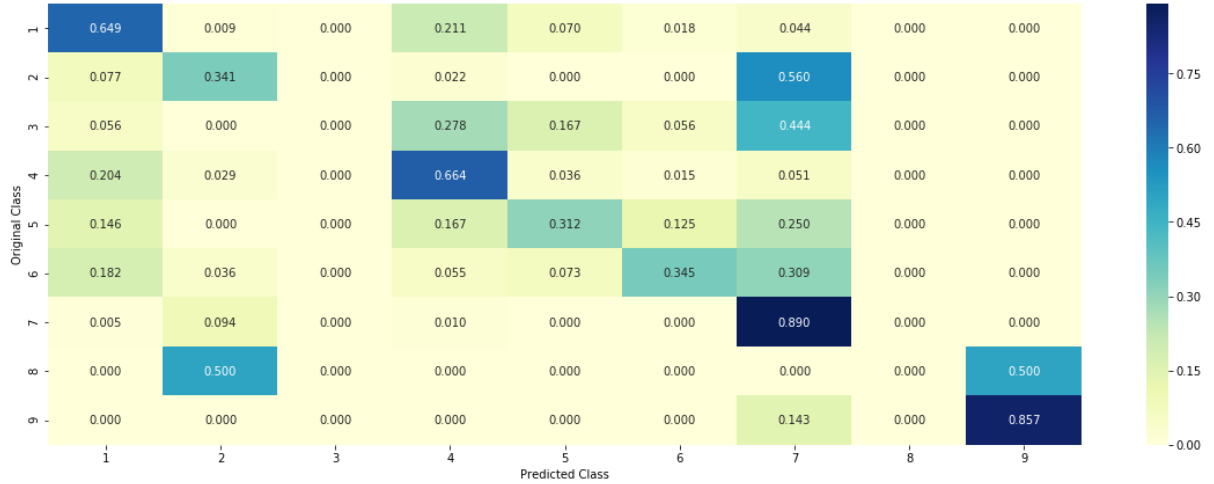
```
Log loss (train) on the stacking classifier : 0.5340893844056577
Log loss (CV) on the stacking classifier : 1.1467409208596786
Log loss (test) on the stacking classifier : 1.1969002114795977
Number of missclassified point : 0.3894736842105263
-------------------- Confusion matrix --------------------
```




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



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



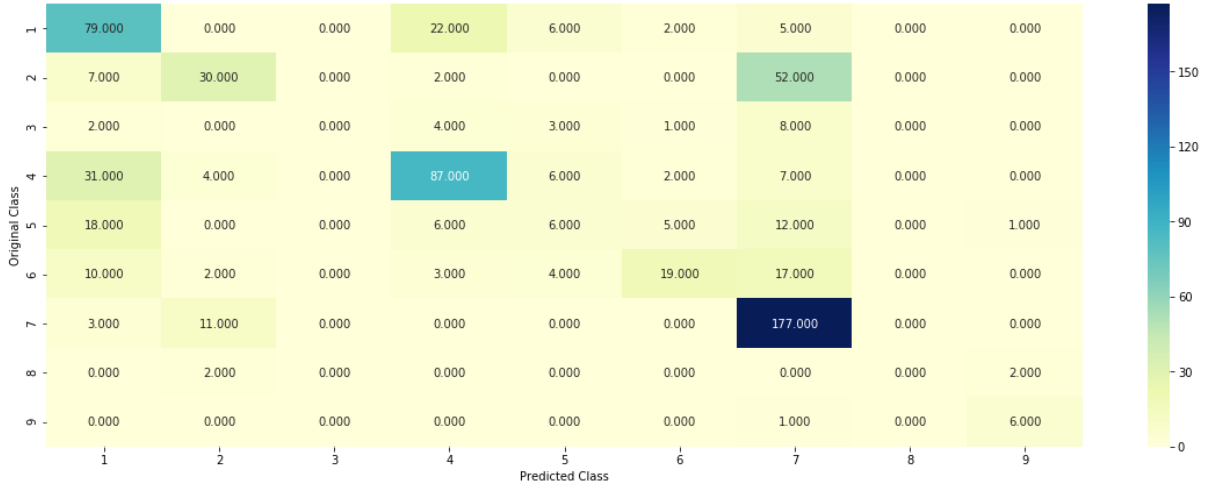
4.7.3 Maximum Voting classifier

In [87]:

```
#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)))
```

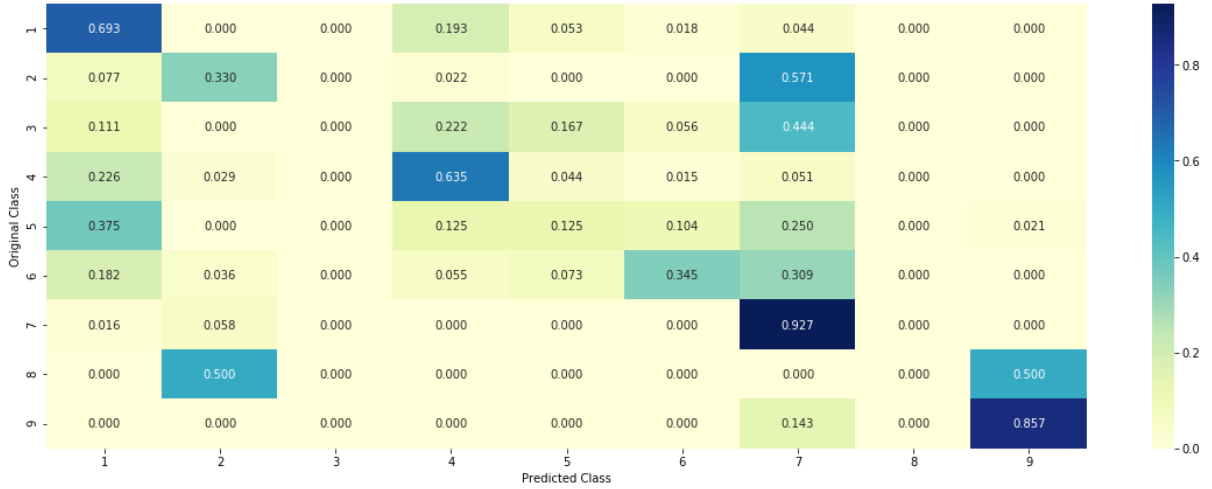
```
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predi
ct_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_
x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCodin
g))
```

```
Log loss (train) on the VotingClassifier : 0.8292639190930888
Log loss (CV) on the VotingClassifier : 1.1981288118954525
Log loss (test) on the VotingClassifier : 1.222221365611401
Number of missclassified point : 0.3924812030075188
-------------------- Confusion matrix --------------------
```




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


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



TfidfVectorization is used in every model

In [89]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No","Model","Train logloss","Cv logloss","Test logloss",
"Misclassified error"]

x.add_row(["1","Naive Bayes","0.517","1.161","1.227","0.38"])
x.add_row(["2","KNN","0.635","1.037","1.076","0.36"])
x.add_row(["3","Logistic regression with class balancing","0.438","1.07","1.0
4","0.349"])
x.add_row(["4","Logistic regression without class balancing","0.44","1.099",
"1.07","0.359"])
x.add_row(["5","Linear svm(with one hot encoding)","0.58","1.116","1.129","0.3
49"])
x.add_row(["6","Random Forest(with one hot encoding)","0.86","1.193","1.23",
"0.40"])
x.add_row(["7","Random Forest(with response coding)","0.069","1.27","1.35","0.
413"])
x.add_row(["8","Stacking classifier","0.53","1.146","1.19","0.38"])
x.add_row(["9","Maximum voting classifier","0.82","1.198","1.22","0.39"])


print(x)
```

```
+------+----------------------------------------------+--------------+-----
```

S.No	Model	Train logloss	Cv logloss	Test logloss	Misclassified error
1	Naive Bayes	0.517	1.161	1.227	0.38
2	KNN	0.635	1.037	1.076	0.36
3	Logistic regression with class balancing	0.438	1.07	1.04	0.349
4	Logistic regression without class balancing	0.44	1.099	1.07	0.359
5	Linear svm(with one hot encoding)	0.58	1.116	1.129	0.349
6	Random Forest(with one hot encoding)	0.86	1.193	1.23	0.40
7	Random Forest(with response coding)	0.069	1.27	1.35	0.413
8	Stacking classifier	0.53	1.146	1.19	0.38
9	Maximum voting classifier	0.82	1.198	1.22	0.39

Conclusion:

1)When we using TfidfVectorizer and also top 1000 features Both logistic regression(without class balancing) and linear svm (with one hot encoding) gives me low MSE:35% which comparitively lower than other models