Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [147]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
```

In [148]:
```
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB

# from sklearn.model_selection import train_test_split
from sklearn.model_selection import 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
print("DONE")
import sklearn
```

```
DONE
```

In [149]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/msk-redefining-cancer-treatment/stage2_test_variants.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/training_variants.zip
/kaggle/input/msk-redefining-cancer-treatment/stage2_sample_submission.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/test_variants.zip
/kaggle/input/msk-redefining-cancer-treatment/stage1_solution_filtered.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/stage_2_private_solution.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/test_text.zip
/kaggle/input/msk-redefining-cancer-treatment/stage2_test_text.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/training_text.zip
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [150]:
```
data = pd.read_csv('/kaggle/input/msk-redefining-cancer-treatment/training_variants.zip')
# data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[150]:

	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 [151]:
```
# note the separator in this file
data_text =pd.read_csv("/kaggle/input/msk-redefining-cancer-treatment/training_text.zip",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
# data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[151]:

	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 [152]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Error loading stopwords: <urlopen error [Errno -3]
[nltk_data]     Temporary failure in name resolution>
```

Out[152]: False

In [153]:
```
# 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 [154]:
```
#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 : 61.562429000000066 seconds
```

In [155]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[155]:

	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 [156]:
```
result[result.isnull().any(axis=1)]
```

Out[156]:

	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 [157]:
```
tempresult=result.copy()
```

In [158]:
```
tempresult
```

Out[158]:

	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...
...	...	...	...	...	...
3316	3316	RUNX1	D171N	4	introduction myelodysplastic syndromes mds het...
3317	3317	RUNX1	A122*	1	introduction myelodysplastic syndromes mds het...
3318	3318	RUNX1	Fusions	1	runt related transcription factor 1 gene runx1...
3319	3319	RUNX1	R80C	4	runx1 aml1 gene frequent target chromosomal tr...
3320	3320	RUNX1	K83E	4	frequent mutations associated leukemia recurre...

3321 rows × 5 columns

In [159]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [160]:
```
result['TEXT'].isnull()
```

Out[160]:
```
0       False
1       False
2       False
3       False
4       False
        ...
3316    False
3317    False
3318    False
3319    False
3320    False
Name: TEXT, Length: 3321, dtype: bool
```

In [161]:
```
result[result['ID']==1109]
```

Out[161]:

	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 [162]:
```
result.Variation
```

Out[162]:
```
0       Truncating Mutations
1                      W802*
2                      Q249E
3                      N454D
4                      L399V
                  ...
3316                   D171N
3317                   A122*
3318                 Fusions
3319                    R80C
3320                    K83E
Name: Variation, Length: 3321, dtype: object
```

In [163]:
```
result.Gene
```

Out[163]:
```
0       FAM58A
1          CBL
2          CBL
3          CBL
4          CBL
           ...
3316     RUNX1
3317     RUNX1
3318     RUNX1
3319     RUNX1
3320     RUNX1
Name: Gene, Length: 3321, dtype: object
```

In [164]:
```
result.Variation
```

Out[164]:
```
0       Truncating Mutations
1                      W802*
2                      Q249E
3                      N454D
4                      L399V
                  ...
3316                   D171N
3317                   A122*
3318                 Fusions
3319                    R80C
3320                    K83E
Name: Variation, Length: 3321, dtype: object
```

In [165]:
```
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)
```

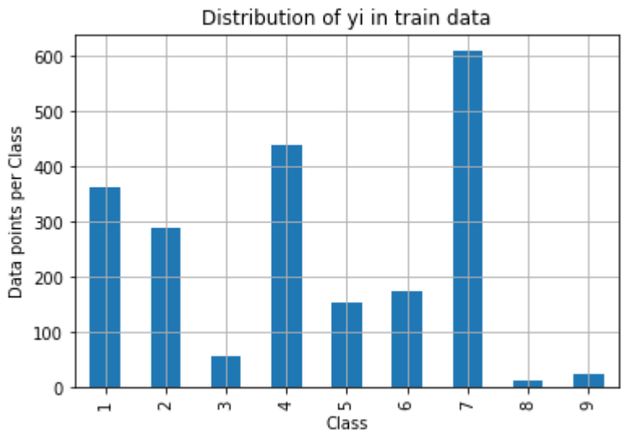
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [166]:
```
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])
```

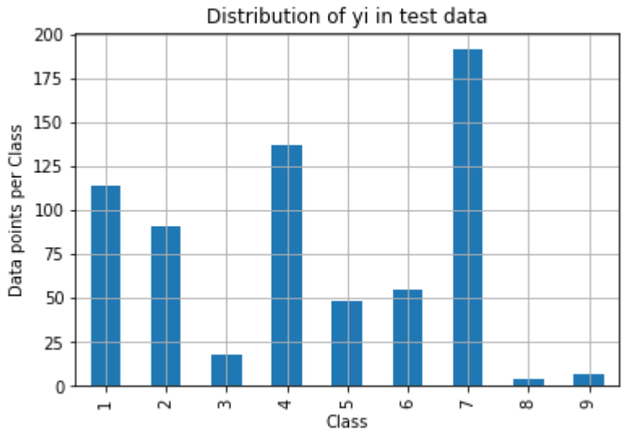
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

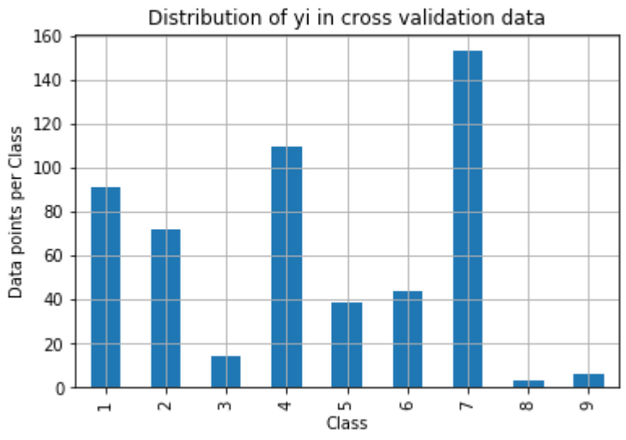
In [167]:
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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


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

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

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

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




```
Number of data points in class 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

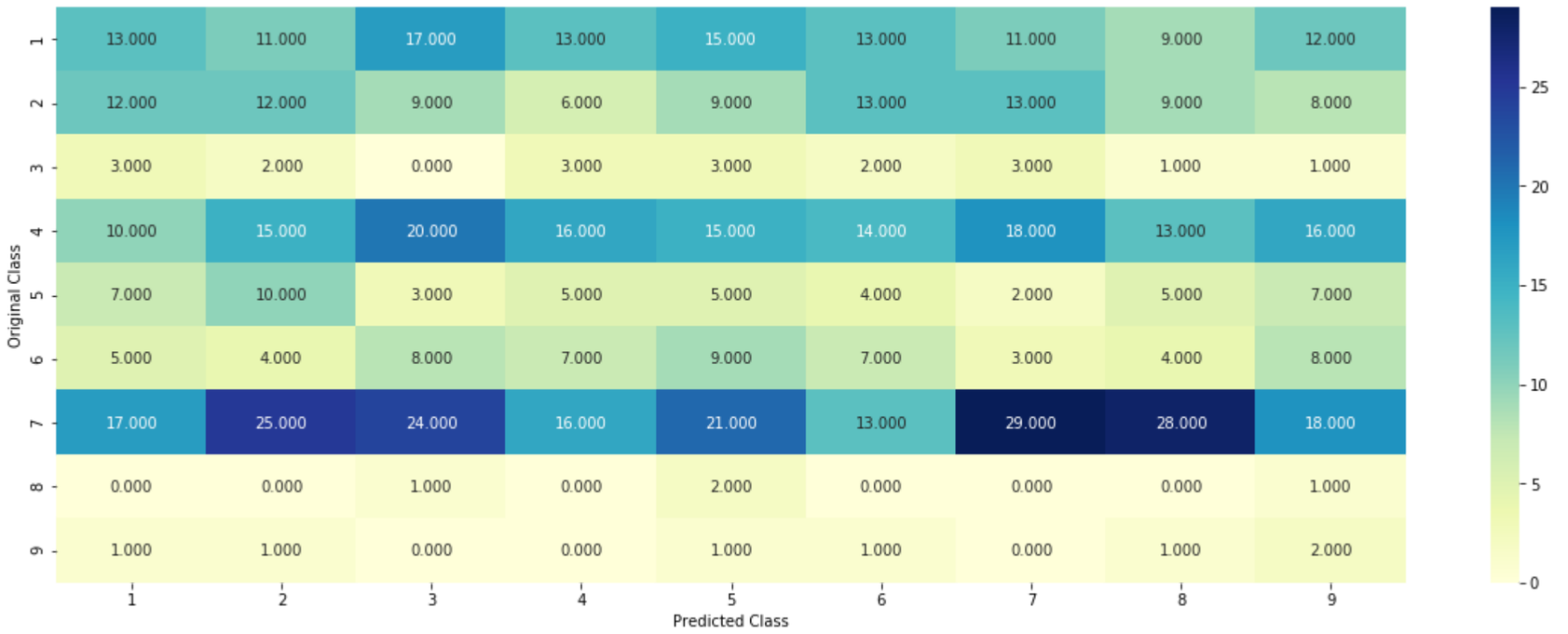
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [168]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in 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 that 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, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

In [169]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix2(test_y, predict_y,logLoss,missClassified):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))

    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, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

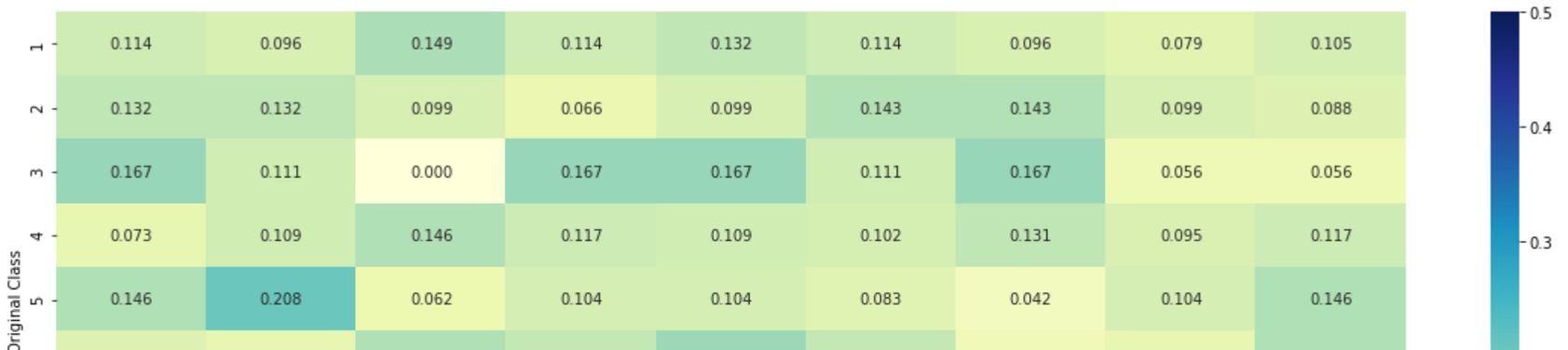
In [170]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.554551553643458
Log loss on Test Data using Random Model 2.4879102108070916
-------------------- Confusion matrix --------------------
```




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


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



3.3 Univariate Analysis

In [171]:
```
# 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 class1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature 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 [172]:
```
uniGenes=train_df['Gene'].value_counts()
```

In [173]:
```
uniGenes
```

Out[173]:
```
BRCA1     156
TP53      112
BRCA2      85
EGFR       82
PTEN       79
          ...
IL7R        1
ERCC3       1
SHOC2       1
RAD51B      1
RIT1        1
Name: Gene, Length: 229, dtype: int64
```

In [174]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 229
BRCA1    156
TP53     112
BRCA2     85
EGFR      82
PTEN      79
BRAF      64
KIT       62
ALK       47
ERBB2     44
FGFR2     35
Name: Gene, dtype: int64
```

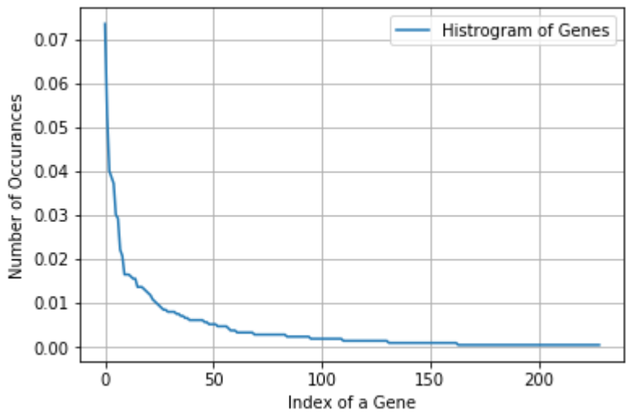
In [175]:
```
unique_genes.shape
```

Out[175]:
```
(229,)
```

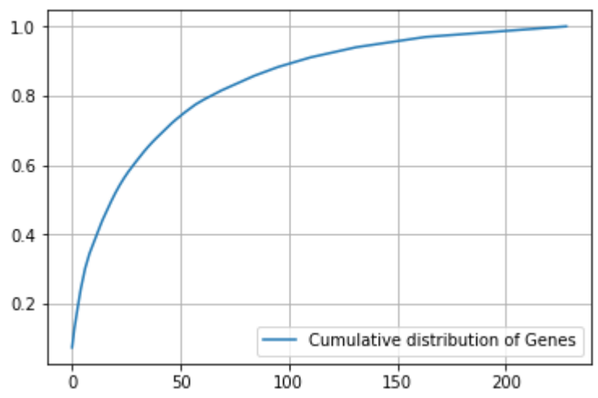
In [176]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, and they are distibuted as follows",)
```

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

In [177]:
```
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 [178]:
```
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.

GENE - RESPONSE CODING

In [179]:
```
# #response-coding of the Gene feature
# # alpha is used for laplace smoothing
# alpha = 1
# # train gene feature
# train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# # test gene feature
# test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# # cross validation gene feature
# cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [180]:
```
# cv_gene_feature_responseCoding[0]
```

In [181]:
```
# cv_gene_feature_responseCoding[0].sum()
```

In [182]:
```
# print("train_gene_feature_responseCoding is converted feature using respone coding method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

GENE - TFIDF (TOP 1000) CODING

In [183]:
```
# one-hot encoding of Gene feature.
## gene_vectorizer = CountVectorizer()
gene_vectorizer = TfidfVectorizer(max_features=1000)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
print('train_gene_feature_onehotCoding',train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding (2124, 229)
```

In [184]:
```
dd=train_gene_feature_onehotCoding.todense()
```

In [185]:
```
dd[0]
```

Out[185]:
```
matrix([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0.]])
```

In [186]:
```
dd.shape
```

Out[186]:
```
(2124, 229)
```

In [187]:
```
train_df['Gene'].head()
```

Out[187]:
```
650     CDKN2A
2714      BRAF
887     PDGFRA
2708      BRAF
1368      AKT2
Name: Gene, dtype: object
```

In [188]:
```
gene_vectorizer.get_feature_names()
```

Out[188]: ['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fat1',
 'fbxw7',
 'fgf3',
 'fgf4',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt3',
 'foxa1',
 'foxl2',
 'foxp1',
 'gata3',
 'gli1',
 'gna11',
 'gnas',
 'h3f3a',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'il7r',
 'jak1',
 'jak2',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'kmt2a',
 'kmt2c',
 'knstrn',
 'kras',
 'lats1',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'men1',

```
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'runx1',
'rxra',
'rybp',
'sdhb',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stag2',
'stat3',
'stk11',
'tcf3',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'whsc1',
'xrcc2',
'yap1']
```

In [189]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

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

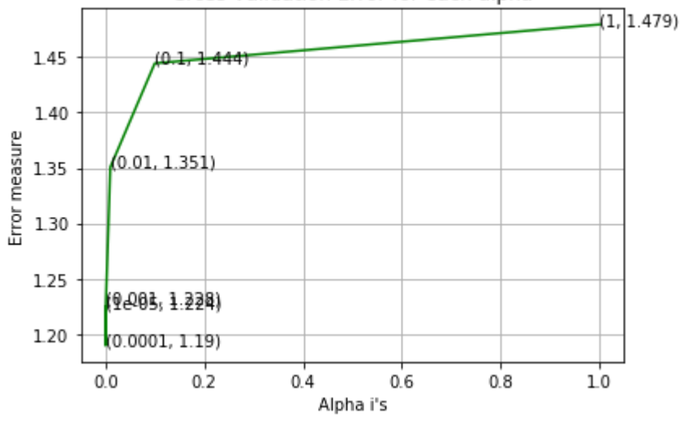
Q4. How good is this gene feature in predicting y_i?

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

In [190]:
```
from sklearn.calibration import CalibratedClassifierCV
```

In [191]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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


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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2242999745910275
For values of alpha =  0.0001 The log loss is: 1.1895316351528993
For values of alpha =  0.001 The log loss is: 1.227967248393791
For values of alpha =  0.01 The log loss is: 1.3506030310237316
For values of alpha =  0.1 The log loss is: 1.4444184021922941
For values of alpha =  1 The log loss is: 1.4794946945589382
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0068861544124488
For values of best alpha =  0.0001 The cross validation log loss is: 1.1895316351528993
For values of best alpha =  0.0001 The test log loss is: 1.174261081726695
```

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 [192]:
```
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], ":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  229  genes in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 512 out of  532 : 96.2406015037594
```

GENE - ONE - HOT CODING

In [193]:
```
# one-hot encoding of Gene feature.
gene_vectorizer_onehot = CountVectorizer(ngram_range=(1,2))
# gene_vectorizer = TfidfVectorizer(max_features=1000)
train_gene_feature_onehotCoding_onehot = gene_vectorizer_onehot.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_onehot = gene_vectorizer_onehot.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_onehot = gene_vectorizer_onehot.transform(cv_df['Gene'])
print('train_gene_feature_onehotCoding_onehot',train_gene_feature_onehotCoding_onehot.shape)
```

```
train_gene_feature_onehotCoding_onehot (2124, 229)
```

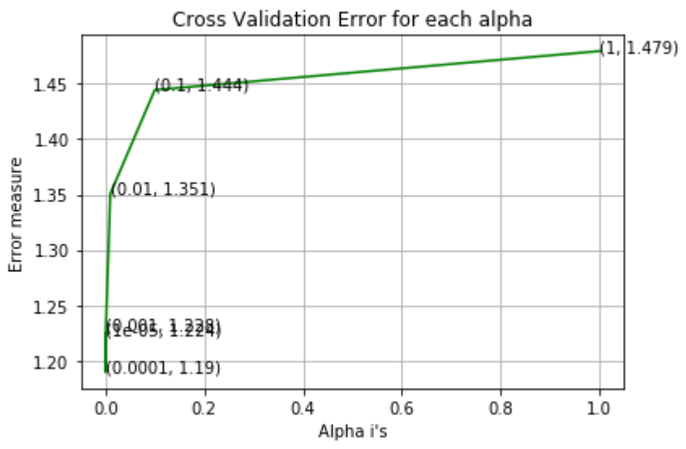
Q4. How good is this gene feature in predicting y_i?

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

In [194]:
```
from sklearn.calibration import CalibratedClassifierCV
```

In [195]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
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_onehot, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding_onehot, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding_onehot)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2242999745910275
For values of alpha =  0.0001 The log loss is: 1.1895316351528993
For values of alpha =  0.001 The log loss is: 1.227967248393791
For values of alpha =  0.01 The log loss is: 1.3506030310237316
For values of alpha =  0.1 The log loss is: 1.4444184021922941
For values of alpha =  1 The log loss is: 1.4794946945589382
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0068861544124488
For values of best alpha =  0.0001 The cross validation log loss is: 1.1895316351528993
For values of best alpha =  0.0001 The test log loss is: 1.174261081726695
```

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?

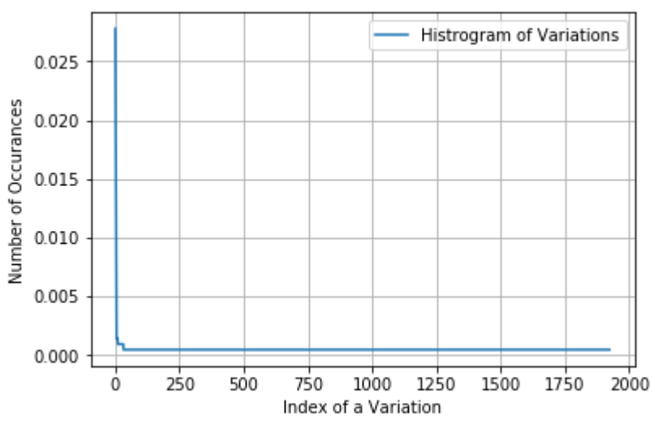
In [196]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1924
Truncating_Mutations    59
Amplification           47
Deletion                41
Fusions                 21
Overexpression           5
E17K                     3
G12V                     3
Q61L                     3
Q61R                     3
T58I                     3
Name: Variation, dtype: int64
```

In [197]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the train data, and they are distibuted as follows",)
```

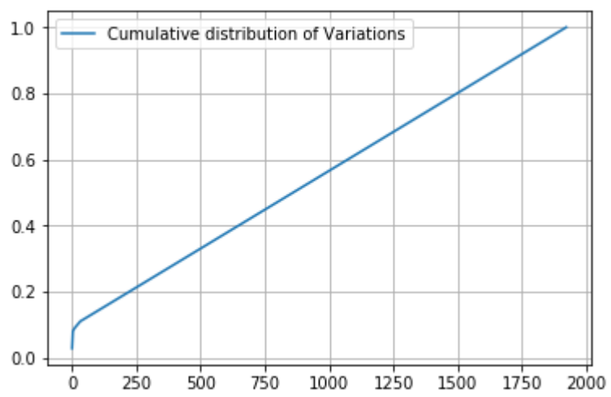
```
Ans: There are 1924 different categories of variations in the train data, and they are distibuted as follows
```

In [198]:
```
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 [199]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02777778 0.04990584 0.06920904 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

VARIATION - RESPONSE CODING

In [200]:
```
# # 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 [201]:
```
# print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Variation feature:", train_variation_feature_responseCoding.
shape)
```

VARIATION - TF-IDF CODING

In [202]:
```
### one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(max_features=1000)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [203]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape
)
```

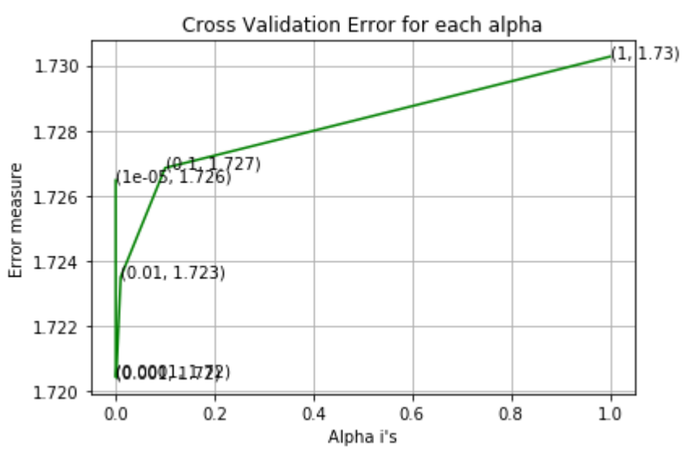
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1000)
```

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

Let's build a model just like the earlier!

In [204]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7264664030573131
For values of alpha =  0.0001 The log loss is: 1.7204337097200115
For values of alpha =  0.001 The log loss is: 1.7203878876346896
For values of alpha =  0.01 The log loss is: 1.7234841315237566
For values of alpha =  0.1 The log loss is: 1.7268452303826953
For values of alpha =  1 The log loss is: 1.7302707058582965
```




```
For values of best alpha =  0.001 The train log loss is: 1.3973075841944802
For values of best alpha =  0.001 The cross validation log loss is: 1.7203878876346896
For values of best alpha =  0.001 The test log loss is: 1.699017936595068
```

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 [205]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1924  genes in test and cross validation data sets?
Ans
1. In test data 65 out of 665 : 9.774436090225564
2. In cross validation data 53 out of  532 : 9.962406015037594
```

VARIATION - One Hot (UNI and BI-GRAM)

In [206]:
```
# one-hot encoding of variation feature.
variation_vectorizer_bigram = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding_bigram = variation_vectorizer_bigram.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_bigram = variation_vectorizer_bigram.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_bigram = variation_vectorizer_bigram.transform(cv_df['Variation'])
```

In [207]:
```
print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method (UNI and BIGRAM). The shape of Variation feature:", train_variation_feature_on
ehotCoding_bigram.shape)
```

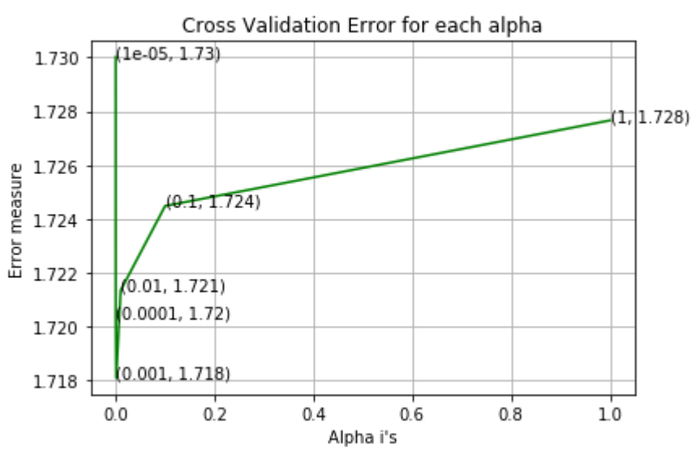
```
train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method (UNI and BIGRAM). The shape of Variation feature: (2124, 2062)
```

Q 10. UNI and BI-GRAM How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [208]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding_bigram, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding_bigram)

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7300080344436586
For values of alpha =  0.0001 The log loss is: 1.720358484476592
For values of alpha =  0.001 The log loss is: 1.7180912906149886
For values of alpha =  0.01 The log loss is: 1.721347317328174
For values of alpha =  0.1 The log loss is: 1.7244844669909951
For values of alpha =  1 The log loss is: 1.7276575422956066
```




```
For values of best alpha =  0.001 The train log loss is: 1.1048643747565163
For values of best alpha =  0.001 The cross validation log loss is: 1.7180912906149886
For values of best alpha =  0.001 The test log loss is: 1.697875190350332
```

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 [209]:
```
# 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
```

RESPONSE CODING

In [210]:
```
# import math
# #https://stackoverflow.com/a/1602964
# def get_text_responsecoding(df):
#     text_feature_responseCoding = np.zeros((df.shape[0],9))
#     for i in range(0,9):
#         row_index = 0
#         for index, row in df.iterrows():
#             sum_prob = 0
#             for word in row['TEXT'].split():
#                 sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
#             text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
#             row_index += 1
#     return text_feature_responseCoding
```

TEXT - TFIDF (TOP 1000) CODING

In [211]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
## text_vectorizer = CountVectorizer(min_df=3)
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

TEXT - BOW (CountVectorizer) UNI and BI-GRAMS (TEXT)

In [212]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer_bigram = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding_bigram = text_vectorizer_bigram.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features_bigram= text_vectorizer_bigram.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
train_text_fea_counts_bigram = train_text_feature_onehotCoding_bigram.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_bigram = dict(zip(list(train_text_features_bigram),train_text_fea_counts_bigram))


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

```
Total number of unique words in train data : 787338
```

In [213]:
```
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 [214]:
```
# #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 [215]:
```
# # https://stackoverflow.com/a/16202486
# # we convert each row values such that they sum to 1
# train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
# test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
# cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1)).T
```

In [216]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [217]:
```
####
```

In [218]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding_bigram = normalize(train_text_feature_onehotCoding_bigram, axis=0)

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

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

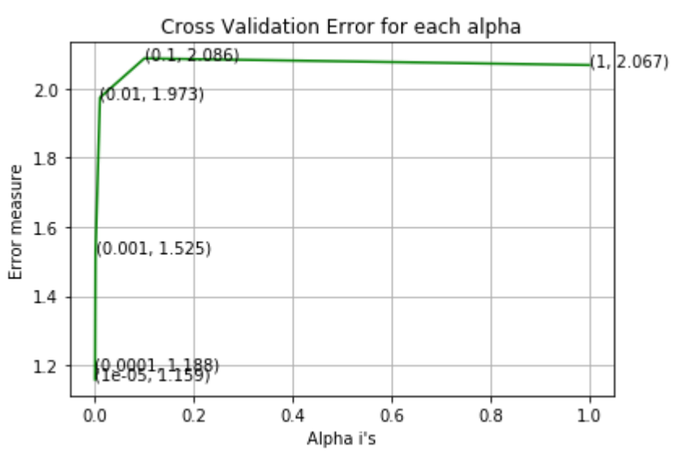
In [219]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [220]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

Counter({251.5266629275148: 1, 179.15230427758627: 1, 132.07513737577347: 1, 131.05450289578982: 1, 130.64239728251266: 1, 119.73628354209205: 1, 119.094916706055: 1, 116.5413742
5907625: 1, 112.3317550916542: 1, 105.31416702988791: 1, 102.11191164536689: 1, 92.43650858044785: 1, 90.98057234434265: 1, 88.17615383530382: 1, 80.90608111587741: 1, 80.9057652
9418246: 1, 79.89329737054796: 1, 79.79648687964983: 1, 79.24853981507883: 1, 77.71552893715882: 1, 75.81500386838981: 1, 73.96892763047592: 1, 70.79117655420265: 1, 69.142142746
9504: 1, 69.13985739707071: 1, 68.10636217854481: 1, 66.37586799278577: 1, 66.20074368287295: 1, 64.01987901483415: 1, 63.45537267111804: 1, 63.13691088163612: 1, 63.052147450448
24: 1, 61.27395064959609: 1, 59.395191241968995: 1, 59.374710637923975: 1, 59.26596648415052: 1, 57.00869992100446: 1, 56.194659938725984: 1, 55.54041136346662: 1, 55.20893222222
5556: 1, 51.98979324104539: 1, 50.79553842155517: 1, 50.31948102305426: 1, 49.2169108794288: 1, 46.73029058715863: 1, 46.44115812181137: 1, 46.28511352719237: 1, 45.9357246093297
25: 1, 44.87104027739289: 1, 44.58867162319854: 1, 43.911586820253035: 1, 43.745960636449105: 1, 43.413666317033375: 1, 43.316270389363275: 1, 42.739066573441235: 1, 42.566418921
78444: 1, 42.562323072076374: 1, 42.44900787603417: 1, 41.990257050344745: 1, 41.91110719002082: 1, 41.881988755498604: 1, 41.42083877610758: 1, 41.360848432696194: 1, 41.3006274
7454337: 1, 40.91369885423249: 1, 40.63825289189093: 1, 39.830857907040766: 1, 39.804554985056775: 1, 39.6621203404753: 1, 39.1827935617212: 1, 38.63799100746519: 1, 38.307519020
25054: 1, 38.10636565814406: 1, 37.7540608166919: 1, 37.68982508027335: 1, 37.29138719658357: 1, 36.11865583559248: 1, 35.91152488390844: 1, 35.83334985881726: 1, 35.773906520818
57: 1, 35.22615373387834: 1, 35.096749582793194: 1, 35.08485714120735: 1, 35.000678893354134: 1, 34.97483246111057: 1, 34.89070578432523: 1, 34.471061001255556: 1, 34.34114945034
238: 1, 33.7391456299185: 1, 33.64581673507555: 1, 33.481819501062596: 1, 33.251596978350854: 1, 32.51162755788297: 1, 32.390112228231324: 1, 32.37907844522694: 1, 32.35601972980
423: 1, 32.32736220094632: 1, 31.964024313974196: 1, 31.85766354829108: 1, 31.757743324471924: 1, 31.68269515204232: 1, 31.619600779230655: 1, 31.61505073749233: 1, 31.5505517834
80138: 1, 31.468426651152505: 1, 31.363994708748496: 1, 31.350377655018473: 1, 31.26680349976636: 1, 31.002089825560066: 1, 30.956062773941355: 1, 30.84816261822326: 1, 30.825156
39193008: 1, 30.524935197697822: 1, 30.50098058600711: 1, 30.478831020254354: 1, 30.10333860474517: 1, 30.076134639102147: 1, 29.894581941347457: 1, 29.78249071868323: 1, 29.7501
4643203911: 1, 29.732980934260706: 1, 29.68643964517145: 1, 29.43362589129366: 1, 29.348726747216052: 1, 28.958170589614603: 1, 28.9187292120630l: 1, 28.911372404205395: 1, 28.68
2163365305076: 1, 28.671467458408372: 1, 28.642929998251088: 1, 28.49420601497217: 1, 28.45997783156012: 1, 28.35109214236139: 1, 28.23736499726761: 1, 28.08076617049204: 1, 27.9
52319376850458: 1, 27.731894226449462: 1, 27.507770092212976: 1, 27.447846953019866: 1, 27.36229571045591: 1, 27.12128931581963: 1, 27.043541881029192: 1, 26.88333570362661: 1, 2
6.695124819484708: 1, 26.668823446281063: 1, 26.643868920334594: 1, 26.44746424835536: 1, 26.367052649586537: 1, 26.230670106255975: 1, 26.182158518435838: 1, 25.887462425532675:
1, 25.853782815133055: 1, 25.394458681793374: 1, 25.321606208466573: 1, 25.29861267275895: 1, 25.065236511622594: 1, 24.98883620094425: 1, 24.97727460606938: 1, 24.89138241982979
2: 1, 24.849296386717793: 1, 24.82274209478202: 1, 24.79701605827437: 1, 24.788894437809144: 1, 24.71587420152481: 1, 24.591281752281226: 1, 24.540245216715665: 1, 24.35712680718
874: 1, 24.252763047691943: 1, 24.04635765524125: 1, 24.043372296291817: 1, 24.03246213240046: 1, 24.024829225370116: 1, 23.978954484426428: 1, 23.818762431165144: 1, 23.74603344
548106: 1, 23.745523628925707: 1, 23.649759897561047: 1, 23.48860897272368: 1, 23.42753219066306: 1, 23.421987821312037: 1, 23.398852847148625: 1, 23.266700290377468: 1, 23.26571
1555485296: 1, 23.233982840956404: 1, 23.132480142642613: 1, 23.096130224535987: 1, 23.04408305716165: 1, 23.0351419194347: 1, 22.90478849778026: 1, 22.865240056233358: 1, 22.785
241251677398: 1, 22.728280420859445: 1, 22.7146162126797: 1, 22.706485831180665: 1, 22.70505818724799: 1, 22.70098415204202: 1, 22.697717970474955: 1, 22.659796204870222: 1, 22.6
47721816827044: 1, 22.603241770238554: 1, 22.601101288603875: 1, 22.54460959214521: 1, 22.429293680513624: 1, 22.324284755736922: 1, 22.28841376895856: 1, 22.199360108584617: 1,
22.144374494303424: 1, 22.106714324816984: 1, 22.101965832691537: 1, 22.08169726122722: 1, 22.010532189339692: 1, 21.965547615976394: 1, 21.91699164129254: 1, 21.882328962156254:
1, 21.758351037792657: 1, 21.75326547656878: 1, 21.648618147151577: 1, 21.599938051714478: 1, 21.572999125743706: 1, 21.486794440297405: 1, 21.447330382803635: 1, 21.393712249709
225: 1, 21.35917712077478: 1, 21.355806804777664: 1, 21.33155484433255: 1, 21.28816923575155: 1, 21.28432621723423: 1, 21.281236841782462: 1, 21.272340112737425: 1, 21.1734053113
2163: 1, 21.11387579092527: 1, 20.97237809477741: 1, 20.91417107328938: 1, 20.758914804387118: 1, 20.73911021842218: 1, 20.643331242370696: 1, 20.59210016670556: 1, 20.5756426327
14917: 1, 20.50914628490756: 1, 20.379919868965533: 1, 20.377219596476287: 1, 20.375631724102362: 1, 20.33428832964075: 1, 20.14978904502911: 1, 20.13640688116101: 1, 20.09265824
2518603: 1, 19.930641906504935: 1, 19.912363038840642: 1, 19.886754264248385: 1, 19.865969557495198: 1, 19.85798275129335: 1, 19.79700485374689: 1, 19.7299704534164: 1, 19.727654
745859965: 1, 19.726315658641063: 1, 19.716339076392746: 1, 19.6827224363386: 1, 19.67827958621656: 1, 19.66599022884304: 1, 19.570150861997398: 1, 19.507279277978068: 1, 19.4260
34894000438: 1, 19.389317676561483: 1, 19.330737891733364: 1, 19.317627786896303: 1, 19.317228579216955: 1, 19.267950934221023: 1, 19.264460050124182: 1, 19.25034840320849: 1, 1
9.230545209599875: 1, 19.226473540049266: 1, 19.191012028106478: 1, 19.147906336006717: 1, 19.088496807587802: 1, 19.046809892364326: 1, 19.02951479571373: 1, 19.02849811939399:
1, 19.024884000693287: 1, 19.016969926222508: 1, 18.982589561619385: 1, 18.93269827015578: 1, 18.921594496452943: 1, 18.861252804567815: 1, 18.860778440603813: 1, 18.774784979259
618: 1, 18.769404970314266: 1, 18.749866872400517: 1, 18.72450350625005: 1, 18.722315766196576: 1, 18.69815400802868: 1, 18.655390183420728: 1, 18.635791898388813: 1, 18.60270133
5614743: 1, 18.592634442091388: 1, 18.525949944023758: 1, 18.511541743297464: 1, 18.49343866907951: 1, 18.42418464918563: 1, 18.42075481234691: 1, 18.35655673455255: 1, 18.341770
669386356: 1, 18.242209836965156: 1, 18.12156467533333: 1, 18.116410199944045: 1, 18.108433241372158: 1, 18.06235006259278: 1, 17.973572013340142: 1, 17.94230710540971: 1, 17.936
66442849714: 1, 17.82141227584853: 1, 17.79918698253035: 1, 17.78098458973056: 1, 17.7485941492556: 1, 17.74760458200231: 1, 17.723926630492084: 1, 17.714112336319634: 1, 17.6699
58059189593: 1, 17.628974175533365: 1, 17.494930614015185: 1, 17.460640674649923: 1, 17.44588497770096: 1, 17.41950661201776: 1, 17.399949483581636: 1, 17.356297016148513: 1, 17.
35541551502975: 1, 17.342605221747977: 1, 17.239488029583622: 1, 17.21431327732402: 1, 17.19675181744034: 1, 17.19665908066273: 1, 17.17542022502428: 1, 17.141677793593608: 1, 1
7.086432104948425: 1, 17.082037380975397: 1, 17.069343422233974: 1, 17.06825265852097: 1, 17.042482454564198: 1, 17.04017464996032: 1, 17.025686839510037: 1, 17.011716769449016:
1, 17.00628062774282: 1, 17.004575013006807: 1, 16.998026265500652: 1, 16.996311002935112: 1, 16.97056037263327: 1, 16.961316015879618: 1, 16.937230660269464: 1, 16.8631096382564
57: 1, 16.859272888046018: 1, 16.840942545718168: 1, 16.801357063902838: 1, 16.779312892575852: 1, 16.747534397962145: 1, 16.728165820827293: 1, 16.709501008760597: 1, 16.6618042
1828248: 1, 16.66178912501689: 1, 16.61053364921637: 1, 16.600492401408506: 1, 16.527721788033205: 1, 16.520646949196284: 1, 16.494490393378467: 1, 16.431124822251547: 1, 16.4184
96514189304: 1, 16.367328717755182: 1, 16.315913082636623: 1, 16.290467350549502: 1, 16.274088973558065: 1, 16.251336407900524: 1, 16.222744108588714: 1, 16.183944378546194: 1, 1
6.179887978810743: 1, 16.162075290581278: 1, 16.128729594993775: 1, 16.120814043078862: 1, 16.021859351653042: 1, 16.00369898371679: 1, 16.000490391543273: 1, 15.98676466253012:
1, 15.929960914430525: 1, 15.80015096455562: 1, 15.775658793296207: 1, 15.67296895252745: 1, 15.657783638791951: 1, 15.634702654808828: 1, 15.601874415866307: 1, 15.5908158110123
3: 1, 15.576973199085241: 1, 15.570009395609425: 1, 15.56704453566969: 1, 15.55575656283311: 1, 15.520654755917052: 1, 15.504021527454848: 1, 15.486214100294163: 1, 15.4853930549
71974: 1, 15.47392343156497: 1, 15.469290249545534: 1, 15.40650068901793: 1, 15.371014303225682: 1, 15.362879562662304: 1, 15.361651443968517: 1, 15.35901425182025: 1, 15.3310695
7405588: 1, 15.258603399359766: 1, 15.230502191661191: 1, 15.202478656207829: 1, 15.136654688479583: 1, 15.123511387751412: 1, 15.1105673982069: 1, 15.093999851767006: 1, 15.0456
31315865041: 1, 15.044657501046064: 1, 15.02019248706174: 1, 15.013050642268889: 1, 14.985128245989861: 1, 14.950142912057858: 1, 14.94827482608006: 1, 14.928637517540258: 1, 14.
883383353433745: 1, 14.87934465791873: 1, 14.844146924932351: 1, 14.83253883196134: 1, 14.787823509084989: 1, 14.771328851923574: 1, 14.76835780473513: 1, 14.741795094549197: 1,
14.715286941298622: 1, 14.69882359546532: 1, 14.697524296560445: 1, 14.692041937883511: 1, 14.675034670996615: 1, 14.668038277701859: 1, 14.642594403437503: 1, 14.63427628803993
3: 1, 14.631615866437716: 1, 14.599109613354296: 1, 14.582860155972634: 1, 14.561911463510258: 1, 14.554373701335729: 1, 14.530363416592445: 1, 14.526283866432577: 1, 14.51119538
2223315: 1, 14.499924709730726: 1, 14.47557406994578: 1, 14.455314661693096: 1, 14.432007568839675: 1, 14.376957382210456: 1, 14.357373934892545: 1, 14.35606264606409: 1, 14.3354
0815884667: 1, 14.326083708352149: 1, 14.322988171443885: 1, 14.315135790285057: 1, 14.29142214701839: 1, 14.256016962501866: 1, 14.22481375729732: 1, 14.191904032522197: 1, 14.1
87870755858118: 1, 14.158186219149822: 1, 14.114686322515242: 1, 14.109917139123924: 1, 14.091347615126875: 1, 14.074017776989207: 1, 14.06650859727984: 1, 14.048651700557462: 1,
14.041794252939265: 1, 14.03987633307574: 1, 14.030000770738877: 1, 14.029603887400178: 1, 14.004209989731953: 1, 13.988056633703504: 1, 13.947706126590239: 1, 13.94449738064368:
1, 13.90918100320369: 1, 13.863454035951536: 1, 13.812701799955912: 1, 13.81149620844085: 1, 13.792094600585399: 1, 13.737540618146367: 1, 13.723758083406901: 1, 13.6694145376978
13: 1, 13.632790889039466: 1, 13.605241254073153: 1, 13.591028108802547: 1, 13.588511981537144: 1, 13.574542516544698: 1, 13.57065562678174: 1, 13.56122253881432: 1, 13.550030689
787764: 1, 13.494235733733936: 1, 13.468179108033734: 1, 13.454056977204909: 1, 13.43867185233727: 1, 13.419513457062758: 1, 13.37099704085222: 1, 13.338267072515636: 1, 13.31232
7963075008: 1, 13.289541061125268: 1, 13.280573716144735: 1, 13.247620544405933: 1, 13.235214181617811: 1, 13.231449730689668: 1, 13.190476393730432: 1, 13.188888229777337: 1, 1
3.162986683761012: 1, 13.147044573645717: 1, 13.099652630634655: 1, 13.036873318220438: 1, 13.021763116225996: 1, 12.999272054364122: 1, 12.964113335367662: 1, 12.95686957075689
5: 1, 12.951016523156486: 1, 12.933578809013014: 1, 12.899156721958537: 1, 12.893032418010138: 1, 12.87013909747861: 1, 12.858212019148928: 1, 12.77338211397532: 1, 12.7688646128
7794: 1, 12.7602773880054: 1, 12.700477643264536: 1, 12.67449848359785: 1, 12.65923968734682: 1, 12.619198402768966: 1, 12.597374916977094: 1, 12.594479840180238: 1, 12.571116945
445489: 1, 12.567551498203658: 1, 12.559208181299388: 1, 12.538618886321412: 1, 12.517555347798446: 1, 12.480851260392752: 1, 12.473315240716566: 1, 12.436281512909208: 1, 12.381
895562676132: 1, 12.377056678033075: 1, 12.37357011023238: 1, 12.36211330109319: 1, 12.35579707419647: 1, 12.353935796799584: 1, 12.325045295960315: 1, 12.321881952995513: 1, 12.
244181252810424: 1, 12.235705848182985: 1, 12.223393022677714: 1, 12.215777790392668: 1, 12.19072506108013: 1, 12.180720795780092: 1, 12.179276396802804: 1, 12.159395563630452:
1, 12.15436187859764: 1, 12.148682019891954: 1, 12.111810679519595: 1, 12.109560849622444: 1, 12.103306781476766: 1, 12.094170848438091: 1, 12.053173575862727: 1, 12.044030984425
127: 1, 12.011272005979638: 1, 12.00342190723525: 1, 11.950494140909143: 1, 11.914310233572804: 1, 11.913041167585254: 1, 11.910811800574983: 1, 11.908591411107146: 1, 11.8975273
86737565: 1, 11.87767398552551: 1, 11.875843253927757: 1, 11.864322157588578: 1, 11.847204988881483: 1, 11.847081758755273: 1, 11.836095778830467: 1, 11.818627168569439: 1, 11.76
066270454204: 1, 11.744748619504117: 1, 11.726115262217476: 1, 11.72469288247318: 1, 11.695197926845434: 1, 11.646714238323701: 1, 11.63334322954305: 1, 11.62212656671402: 1, 11.
62009011350267: 1, 11.618556124700344: 1, 11.601933392202586: 1, 11.593442096690524: 1, 11.586517207228988: 1, 11.585060407225853: 1, 11.573113445860784: 1, 11.570314790478639:
1, 11.569258322200774: 1, 11.563453676849816: 1, 11.558386913538333: 1, 11.556185065826151: 1, 11.463093212764557: 1, 11.462967797281282: 1, 11.41476032194289: 1, 11.357950632636
653: 1, 11.33551209013312: 1, 11.306420757541478: 1, 11.3008934667106: 1, 11.300309671752066: 1, 11.29987355686667: 1, 11.299804109928534: 1, 11.296861064226272: 1, 11.2916365517
46379: 1, 11.279220534007028: 1, 11.277587685929632: 1, 11.276392080848293: 1, 11.275257458233513: 1, 11.251734376745118: 1, 11.247908586303588: 1, 11.200013735811243: 1, 11.1925
44808103568: 1, 11.185437761405904: 1, 11.149135691070407: 1, 11.14756669278722: 1, 11.110647218203548: 1, 11.109558080925504: 1, 11.102062363430823: 1, 11.082436089770416: 1, 1
1.081645206005021: 1, 11.077175588120346: 1, 11.040865244808103: 1, 11.028894641705795: 1, 11.022691221042127: 1, 11.016773892611882: 1, 11.014484326562794: 1, 11.00305253950711
1: 1, 11.001733336974187: 1, 10.938556706810107: 1, 10.930932329067458: 1, 10.928437584399429: 1, 10.924863385505844: 1, 10.913777428900707: 1, 10.905493881927528: 1, 10.89001027
0025384: 1, 10.88466943332076: 1, 10.870244895725351: 1, 10.863599122361908: 1, 10.85962927284713: 1, 10.844667515318655: 1, 10.844383967909792: 1, 10.796640601237126: 1, 10.7962
17233515923: 1, 10.780027409541217: 1, 10.776376824567977: 1, 10.764635534064293: 1, 10.762523130461464: 1, 10.741364596882452: 1, 10.740225726143173: 1, 10.733428035266872: 1, 1
0.721714154987222: 1, 10.708944660078467: 1, 10.676056516780106: 1, 10.671833542409937: 1, 10.666138477331451: 1, 10.658925532973912: 1, 10.648697509010814: 1, 10.64001272961357
2: 1, 10.6390917163409: 1, 10.58492487368l445: 1, 10.569775623300028: 1, 10.561100200704404: 1, 10.552957852040914: 1, 10.54243518237234: 1, 10.530089888650942: 1, 10.51419181445
1733: 1, 10.503227224313662: 1, 10.496095686533454: 1, 10.48842352617186: 1, 10.469804545386365: 1, 10.462444721895956: 1, 10.459360554933781: 1, 10.445846847279109: 1, 10.432872
949326274: 1, 10.429714930386574: 1, 10.425167143852624: 1, 10.422327378327546: 1, 10.420676050103749: 1, 10.4144362443099: 1, 10.410218306738422: 1, 10.398120343337096: 1, 10.37
79353202678: 1, 10.371584271456951: 1, 10.367466381692532: 1, 10.362195442687524: 1, 10.33730463156311: 1, 10.333064122223385: 1, 10.314868727772145: 1, 10.314516776786068: 1, 1
0.289004657225053: 1, 10.254382291922337: 1, 10.249335819299429: 1, 10.219719474991628: 1, 10.21489812701945: 1, 10.201158900287364: 1, 10.19559040362719: 1, 10.193787919620076:
1, 10.18754676412606: 1, 10.179273453685708: 1, 10.17194878792527: 1, 10.17000148794l839: 1, 10.151115558844953: 1, 10.13826513006404: 1, 10.135677333251907: 1, 10.11321838644545
7: 1, 10.0794544325709: 1, 10.067501732781366: 1, 10.052675831979874: 1, 10.045881106191167: 1, 10.033308926479709: 1, 10.029372699059376: 1, 10.024045006913608: 1, 10.0175154545
2456: 1, 10.011156410423267: 1, 9.939637061977157: 1, 9.931937164740027: 1, 9.931031052603819: 1, 9.929456874088297: 1, 9.928070035559958: 1, 9.90786016320583&: 1, 9.905733396432
758: 1, 9.905586101061418: 1, 9.89759073596686: 1, 9.894069720703026: 1, 9.876104142183042: 1, 9.87115827830354: 1, 9.858760712249355: 1, 9.854782301724795: 1, 9.848907561686602:
1, 9.847159105184872: 1, 9.825218044535946: 1, 9.822672340820944: 1, 9.808993978902514: 1, 9.805370280857733: 1, 9.79564850937911: 1, 9.786376420064348: 1, 9.782053124253379: 1,
9.77636196052525: 1, 9.760938102485747: 1, 9.75857965410724: 1, 9.748386239395655: 1, 9.738604807786855: 1, 9.706013670390664: 1, 9.699978583456758: 1, 9.694613197545234: 1, 9.68
7767050497218: 1, 9.679446521760886: 1, 9.660942504814944: 1, 9.64893952978451: 1, 9.64719238870671: 1, 9.64673120330398: 1, 9.631156473621795: 1, 9.619728832681533: 1, 9.6180849
8640399: 1, 9.614838495320361: 1, 9.607203635172139: 1, 9.596158467262542: 1, 9.59556833561141: 1, 9.593803911656233: 1, 9.59297863951938: 1, 9.58887350291116: 1, 9.5736209349172
87: 1, 9.545280120428888: 1, 9.536367087876705: 1, 9.529879636515423: 1, 9.518919598397858: 1, 9.50927123515788: 1, 9.503433858671007: 1, 9.481226453288865: 1, 9.471260814008069:
1, 9.467762986696265: 1, 9.45943497007215: 1, 9.443511923761953: 1, 9.436859603834465: 1, 9.427416072430464: 1, 9.426655319877561: 1, 9.417373158240292: 1, 9.384473204787753: 1,
9.383073426823287: 1, 9.365051183097542: 1, 9.350879756299898: 1, 9.349687257008497: 1, 9.344758738490386: 1, 9.34393614755008: 1, 9.340169813469792: 1, 9.295482548139887: 1, 9.2
88536971985897: 1, 9.264016655107454: 1, 9.255717267072578: 1, 9.23450833935926: 1, 9.226518334679378: 1, 9.216191081292427: 1, 9.209555721628293: 1, 9.203092324231346: 1, 9.1943
3873604396: 1, 9.190480724686804: 1, 9.185048051216516: 1, 9.179600102358334: 1, 9.174475643078473: 1, 9.172105439736276: 1, 9.163873541371908: 1, 9.142079639478444: 1, 9.1387645
01276917: 1, 9.136143523847474: 1, 9.130534110243536: 1, 9.128216687182354: 1, 9.123394104898741: 1, 9.12257117896861: 1, 9.11717629260703: 1, 9.109915802800096: 1, 9.08394504012
4515: 1, 9.06000621916138: 1, 9.058492747379036: 1, 9.054182443009326: 1, 9.05133389220083: 1, 9.016626629760802: 1, 9.013087586695066: 1, 9.011964469751526: 1, 9.00839973274149
1: 1, 9.00097789473375: 1, 8.986661583162764: 1, 8.970903482021718: 1, 8.963501078323555: 1, 8.956939194472467: 1, 8.956875200543625: 1, 8.951919028975587: 1, 8.945393833923411:
1, 8.94222812814055: 1, 8.934710840348924: 1, 8.914519985841137: 1, 8.903363026039123: 1, 8.888875515542534: 1, 8.877441252495057: 1, 8.876733691780128: 1, 8.85771003029548: 1,
8.855516941622794: 1, 8.855293557931866: 1, 8.853129190012579: 1, 8.845408398578234: 1, 8.845394856952359: 1, 8.844856009365337: 1, 8.834552951097823: 1, 8.833428820566045: 1, 8.
822974885219105: 1, 8.81853055160164: 1, 8.811327802167913: 1, 8.80696880773158: 1, 8.80634154840781: 1, 8.800704941049956: 1, 8.77832165948332: 1, 8.744678790744109: 1, 8.736398
983971903: 1, 8.717722027959534: 1, 8.709829508507738: 1, 8.703686869111273: 1, 8.67094316370076: 1, 8.662844079766613: 1, 8.65148410616268: 1, 8.632821351651344: 1, 8.6198984424
23185: 1, 8.61558104010093: 1, 8.600905751236045: 1, 8.59356442186401: 1, 8.592668656391194: 1, 8.580781300277605: 1, 8.541252000599362: 1, 8.53967843793726: 1, 8.52891927878829
2: 1, 8.528832201774762: 1, 8.527597844080548: 1, 8.514570389223378: 1, 8.511041248211267: 1, 8.491552907569748: 1, 8.484786689779867: 1, 8.48189281165803: 1, 8.402491137245127:
1, 8.40226215294357: 1, 8.394859661985508: 1, 8.389690012237768: 1, 8.387856197734003: 1, 8.386579328008509: 1, 8.363769669420115: 1, 8.338581292290039: 1, 8.329272832256148: 1,
8.324891694631356: 1, 8.321455208925036: 1, 8.31398115648589: 1, 8.293966834375434: 1, 8.27908107277538: 1, 8.265107451915577: 1, 8.231049474364726: 1, 8.228980628988392: 1, 8.20
8571893694705: 1, 8.204149240565265: 1, 8.203163091437842: 1, 8.192431817959898: 1, 8.184796473118093: 1, 8.178417310705795: 1, 8.178211908714385: 1, 8.174736697709884: 1, 8.1744
15092040014: 1, 8.165094015500806: 1, 8.154190234994424: 1, 8.138531636328572: 1, 8.125468895942264: 1, 8.11447665976072: 1, 8.108248007382466: 1, 8.087163693187284: 1, 8.0749293
98031522: 1, 8.073772783110051: 1, 8.0691690481117: 1, 8.054466862948669: 1, 8.020904824433616: 1, 8.017330479123759: 1, 8.011428515964509: 1, 7.969761880453169: 1, 7.95787790890
3687: 1, 7.957205792423041: 1, 7.945086925120529: 1, 7.9434678427240355: 1, 7.943189818325221: 1, 7.937798283515465: 1, 7.921686987925449: 1, 7.907176143060283: 1, 7.905550348416
1616: 1, 7.904342275749974: 1, 7.890689452594859: 1, 7.886695879092352: 1, 7.8836037357036455: 1, 7.878348003798637: 1, 7.873857178277232: 1, 7.855108511355673: 1, 7.822744934096
8445: 1, 7.815741421483876: 1, 7.806530213924364: 1, 7.775234661668178: 1, 7.7748338889064375: 1, 7.772546105732782: 1, 7.760168721159105: 1, 7.756188768705253: 1, 7.746748114366
124: 1, 7.727649924127846: 1, 7.717121356580909: 1, 7.708871882806324: 1, 7.69612401262008: 1, 7.673980703988114: 1, 7.658723532219575: 1, 7.634137515516192: 1, 7.63269407410693
9: 1, 7.6250920867935: 1, 7.624602649269309: 1, 7.6125904015703405: 1, 7.595470471931277: 1, 7.594108471726683: 1, 7.573937129235389: 1, 7.564999473775559: 1, 7.559528575523173:
1, 7.550455315879203: 1, 7.482062896642249: 1, 7.43224595961424: 1, 7.419877377024646: 1, 7.414046479808557: 1, 7.409948649008403: 1, 7.394081213858113: 1, 7.381779314972532: 1,
7.373881206577202: 1, 7.354309454573049: 1, 7.319090617282256: 1, 7.31738634891026: 1, 7.311473822042447: 1, 7.295601639176031: 1, 7.262982764280094: 1, 7.244586043792031: 1, 7.2
36291953420071: 1, 7.233479134967283: 1, 7.200987538098315: 1, 7.182560842275652: 1, 7.158314068672039: 1, 7.081915535502985: 1, 7.080501384192598: 1, 7.0745195983844305: 1, 7.06
7904182493889: 1, 7.066207107733535: 1, 7.052674673462182: 1, 6.964344667070322: 1, 6.922865143792101: 1, 6.911894506613141: 1, 6.909505191296729: 1, 6.881451731005948: 1, 6.8799
15028440873: 1, 6.827979234569558: 1, 6.826806415135855: 1, 6.7550769970033535: 1, 6.741954762120101: 1, 6.7390455825452324: 1, 6.59199589708233: 1, 6.519523229882225: 1, 6.48529
9604706934: 1})

In [221]:
```
# 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/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1591850021062553
For values of alpha =  0.0001 The log loss is: 1.187733408029477
For values of alpha =  0.001 The log loss is: 1.5247463465715998
For values of alpha =  0.01 The log loss is: 1.9731076286486344
For values of alpha =  0.1 The log loss is: 2.0864287312161776
For values of alpha =  1 The log loss is: 2.067045030249584
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7268207746590241
For values of best alpha =  1e-05 The cross validation log loss is: 1.1591850021062553
For values of best alpha =  1e-05 The test log loss is: 1.0855473928942159
```

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

Ans. Yes, it seems like !

In [222]:
```
def get_intersec_text(df):
#     df_text_vec = CountVectorizer(min_df=3)
    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
```

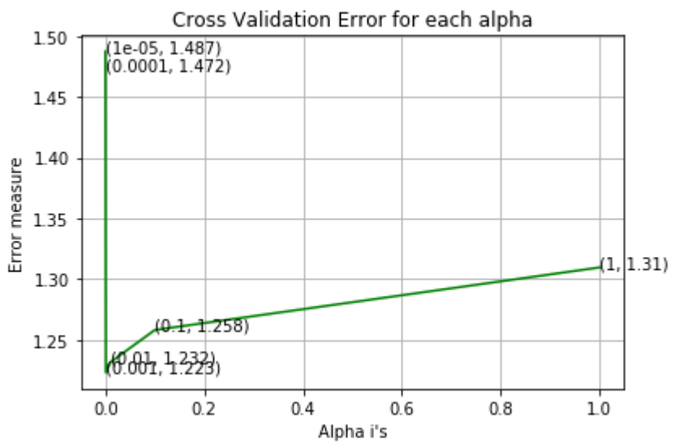
In [223]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
95.1 % of word of test data appeared in train data
94.4 % of word of Cross Validation appeared in train data
```

LOGISTIC REGRESSION FOR BOTH UNIGRAM and BIGRAM

In [224]:
```
alpha = [10 ** x for x in range(-5, 1)]

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_bigram, y_train)

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.4872251795665217
For values of alpha =  0.0001 The log loss is: 1.4717684925652321
For values of alpha =  0.001 The log loss is: 1.2230989598966655
For values of alpha =  0.01 The log loss is: 1.2315308831254446
For values of alpha =  0.1 The log loss is: 1.2580788757668264
For values of alpha =  1 The log loss is: 1.3095461411793994
```




```
For values of best alpha =  0.001 The train log loss is: 0.8571802788575004
For values of best alpha =  0.001 The cross validation log loss is: 1.2230989598966655
For values of best alpha =  0.001 The test log loss is: 1.2248680239976653
```

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

Ans. Yes, it seems like !

In [225]:
```
def get_intersec_text(df):
    df_text_vec_bigram = TfidfVectorizer(min_df=3,ngram_range=(1,2))
    df_text_fea_bigram = df_text_vec_bigram.fit_transform(df['TEXT'])
    df_text_features_bigram = df_text_vec_bigram.get_feature_names()

    df_text_fea_counts_bigram = df_text_fea_bigram.sum(axis=0).A1
    df_text_fea_dict_bigram = dict(zip(list(df_text_features_bigram),df_text_fea_counts_bigram))
    len1 = len(set(df_text_features_bigram))
    len2 = len(set(train_text_features_bigram) & set(df_text_features_bigram))
    return len1,len2
```

In [226]:
```
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")
```

```
94.098 % of word of test data appeared in train data
95.976 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [227]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    logLoss=log_loss(test_y, sig_clf.predict_proba(test_x))
    print("Log loss :",logLoss)
    # calculating the number of data points that are misclassified
    missClassified=np.count_nonzero((pred_y- test_y))/test_y.shape[0]
    print("Number of mis-classified points :", missClassified)
#     plot_confusion_matrix2(test_y, pred_y,logLoss,missClassified)
    plot_confusion_matrix(test_y, pred_y)
    return missClassified
```

In [228]:
```
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 [229]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
#     gene_count_vec = CountVectorizer()
#     var_count_vec = CountVectorizer()
#     text_count_vec = CountVectorizer(min_df=3)
    gene_count_vec = TfidfVectorizer(max_features=1000)
    var_count_vec = TfidfVectorizer(max_features=1000)
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

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

Stacking the three types of features

In [230]:
```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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


# train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
# test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
# cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

# train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_responseCoding))
# test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding))
# cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
```

In [231]:
```
## for count vectorizer
train_x_onehotCoding_count = hstack((train_gene_feature_onehotCoding_onehot,train_variation_feature_onehotCoding_bigram, train_text_feature_onehotCoding_bigram)).tocsr()
test_x_onehotCoding_count = hstack((test_gene_feature_onehotCoding_onehot,test_variation_feature_onehotCoding_bigram, test_text_feature_onehotCoding_bigram)).tocsr()
cv_x_onehotCoding_count = hstack((cv_gene_feature_onehotCoding_onehot,cv_variation_feature_onehotCoding_bigram, cv_text_feature_onehotCoding_bigram)).tocsr()
```

In [232]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2229)
(number of data points * number of features) in test data =  (665, 2229)
(number of data points * number of features) in cross validation data = (532, 2229)
```

In [233]:
```
print("One hot encoding COUNT VECTORIZER features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding_count.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding_count.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding_count.shape)
```

```
One hot encoding COUNT VECTORIZER features :
(number of data points * number of features) in train data =  (2124, 789629)
(number of data points * number of features) in test data =  (665, 789629)
(number of data points * number of features) in cross validation data = (532, 789629)
```

In [234]:
```
# print(" Response encoding features :")
# print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
# print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
# print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

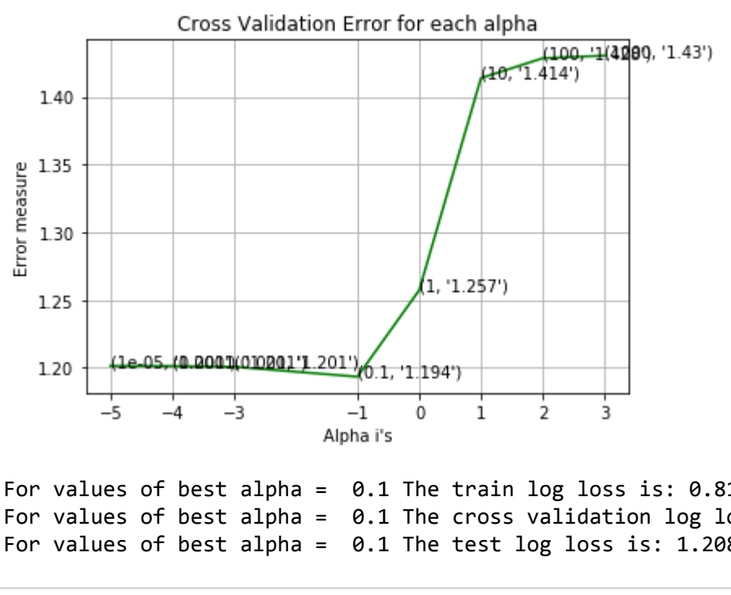
4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [235]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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


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

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


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


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

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

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

```
for alpha = 1e-05
Log Loss : 1.2014321503716936
for alpha = 0.0001
Log Loss : 1.2013574363262611
for alpha = 0.001
Log Loss : 1.200833634361808
for alpha = 0.1
Log Loss : 1.1935034649891239
for alpha = 1
Log Loss : 1.2573219575354158
for alpha = 10
Log Loss : 1.4137083163240076
for alpha = 100
Log Loss : 1.4283563851144985
for alpha = 1000
Log Loss : 1.4302869882474407
```



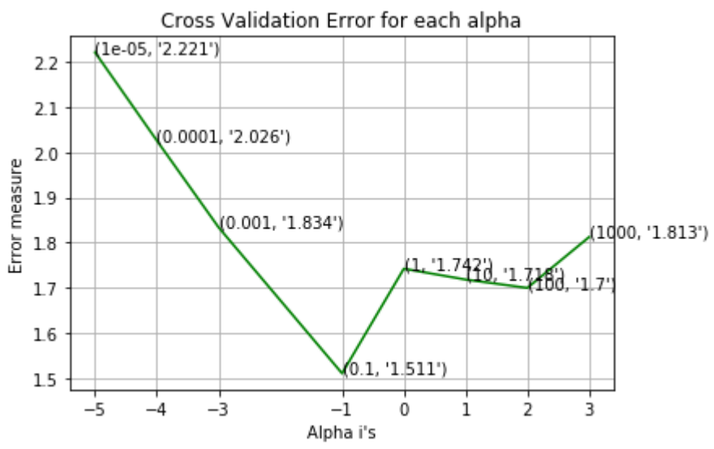

```
For values of best alpha =  0.1 The train log loss is: 0.810328423649598
For values of best alpha =  0.1 The cross validation log loss is: 1.1935034649891239
For values of best alpha =  0.1 The test log loss is: 1.2085765455210271
```

In [236]:
```
print(nbLoss_train)
print(nbLoss_cv)
print(nbLoss_test)
```

```
0.810328423649598
1.1935034649891239
1.2085765455210271
```

In [237]:
```
## WITHOUT CALIBRATION


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)
    sig_clf_probs = clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 2.2213200372124438
for alpha = 0.0001
Log Loss : 2.0262384315885478
for alpha = 0.001
Log Loss : 1.834130714688919
for alpha = 0.1
Log Loss : 1.5108757141932139
for alpha = 1
Log Loss : 1.7424278869426026
for alpha = 10
Log Loss : 1.7184577790546804
for alpha = 100
Log Loss : 1.6999764108419855
for alpha = 1000
Log Loss : 1.8125488465572428
```




```
For values of best alpha =  0.1 The train log loss is: 0.5384459958450364
For values of best alpha =  0.1 The cross validation log loss is: 1.5108757141932139
For values of best alpha =  0.1 The test log loss is: 1.423438634908445
```

4.1.1.2. Testing the model with best hyper paramters

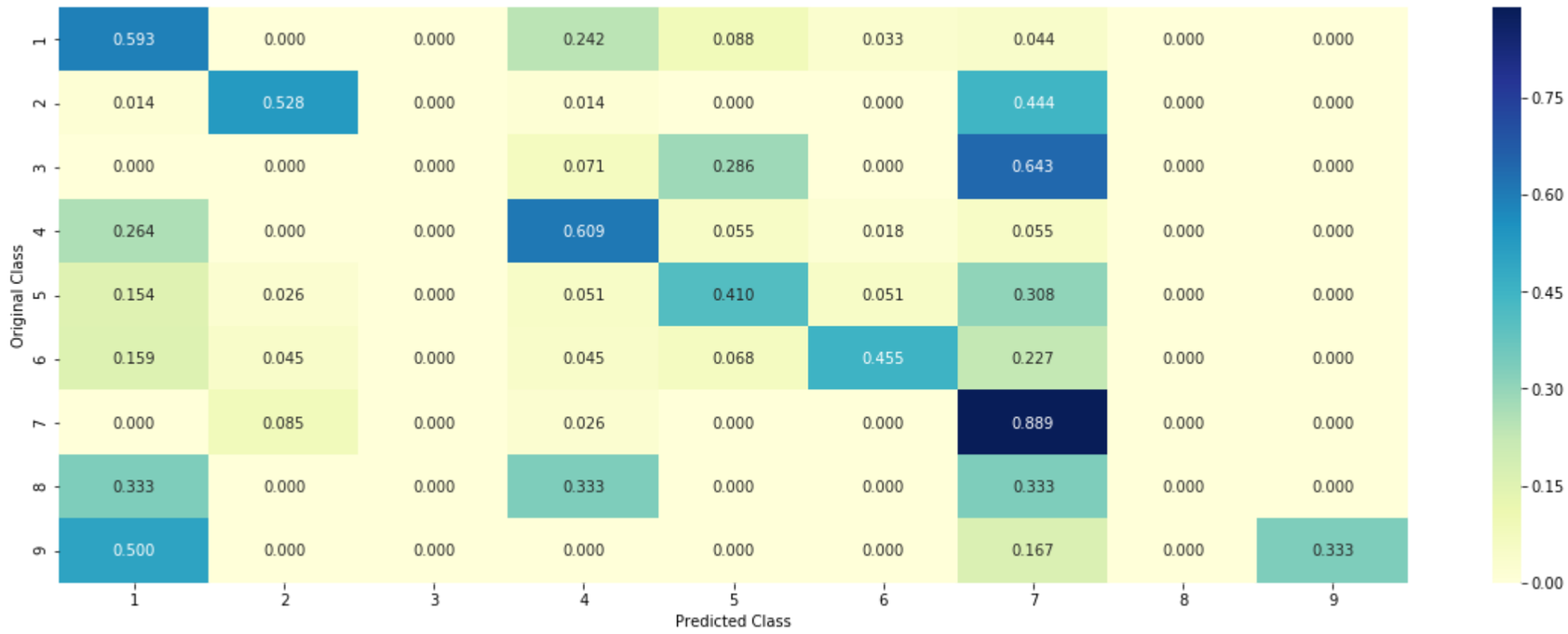
In [238]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.1915164605307575
Number of missclassified point : 0.37406015037593987
-------------------- Confusion matrix --------------------
```


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



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



4.1.1.3. Feature Importance, Correctly classified point

In [239]:
```
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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.05   0.0515 0.0173 0.0519 0.035  0.0356 0.7493 0.0049 0.0045]]
Actual Class : 7
--------------------------------------------------
15 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 [downstream] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [signaling] present in test data point [True]
26 Text feature [growth] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [contrast] present in test data point [True]
29 Text feature [independent] present in test data point [True]
30 Text feature [inhibitor] present in test data point [True]
33 Text feature [10] present in test data point [True]
34 Text feature [however] present in test data point [True]
35 Text feature [factor] present in test data point [True]
36 Text feature [addition] present in test data point [True]
37 Text feature [phosphorylation] present in test data point [True]
38 Text feature [constitutive] present in test data point [True]
39 Text feature [compared] present in test data point [True]
40 Text feature [similar] present in test data point [True]
41 Text feature [presence] present in test data point [True]
42 Text feature [cell] present in test data point [True]
43 Text feature [shown] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [well] present in test data point [True]
47 Text feature [inhibitors] present in test data point [True]
48 Text feature [treatment] present in test data point [True]
50 Text feature [activating] present in test data point [True]
51 Text feature [showed] present in test data point [True]
52 Text feature [previously] present in test data point [True]
53 Text feature [may] present in test data point [True]
54 Text feature [potential] present in test data point [True]
55 Text feature [oncogenic] present in test data point [True]
56 Text feature [found] present in test data point [True]
57 Text feature [suggest] present in test data point [True]
59 Text feature [proliferation] present in test data point [True]
60 Text feature [treated] present in test data point [True]
61 Text feature [inhibited] present in test data point [True]
62 Text feature [increased] present in test data point [True]
64 Text feature [pathways] present in test data point [True]
65 Text feature [mutant] present in test data point [True]
66 Text feature [although] present in test data point [True]
67 Text feature [results] present in test data point [True]
68 Text feature [observed] present in test data point [True]
69 Text feature [3b] present in test data point [True]
71 Text feature [absence] present in test data point [True]
73 Text feature [increase] present in test data point [True]
75 Text feature [different] present in test data point [True]
77 Text feature [mutation] present in test data point [True]
78 Text feature [described] present in test data point [True]
80 Text feature [activate] present in test data point [True]
82 Text feature [two] present in test data point [True]
83 Text feature [various] present in test data point [True]
84 Text feature [figure] present in test data point [True]
85 Text feature [recent] present in test data point [True]
86 Text feature [studies] present in test data point [True]
87 Text feature [inhibition] present in test data point [True]
88 Text feature [respectively] present in test data point [True]
89 Text feature [20] present in test data point [True]
91 Text feature [mechanism] present in test data point [True]
92 Text feature [discussion] present in test data point [True]
93 Text feature [using] present in test data point [True]
94 Text feature [either] present in test data point [True]
95 Text feature [serum] present in test data point [True]
97 Text feature [including] present in test data point [True]
98 Text feature [therapeutic] present in test data point [True]
99 Text feature [without] present in test data point [True]
Out of the top  100  features  67 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [240]:
```
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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0518 0.0544 0.0177 0.0561 0.0356 0.036  0.7388 0.005  0.0046]]
Actual Class : 7
--------------------------------------------------
15 Text feature [activation] present in test data point [True]
18 Text feature [activated] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [signaling] present in test data point [True]
26 Text feature [growth] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [contrast] present in test data point [True]
29 Text feature [independent] present in test data point [True]
30 Text feature [inhibitor] present in test data point [True]
33 Text feature [10] present in test data point [True]
34 Text feature [however] present in test data point [True]
35 Text feature [factor] present in test data point [True]
36 Text feature [addition] present in test data point [True]
37 Text feature [phosphorylation] present in test data point [True]
38 Text feature [constitutive] present in test data point [True]
39 Text feature [compared] present in test data point [True]
40 Text feature [similar] present in test data point [True]
41 Text feature [presence] present in test data point [True]
42 Text feature [cell] present in test data point [True]
43 Text feature [shown] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [well] present in test data point [True]
46 Text feature [sensitive] present in test data point [True]
47 Text feature [inhibitors] present in test data point [True]
48 Text feature [treatment] present in test data point [True]
49 Text feature [higher] present in test data point [True]
50 Text feature [activating] present in test data point [True]
51 Text feature [showed] present in test data point [True]
53 Text feature [may] present in test data point [True]
54 Text feature [potential] present in test data point [True]
55 Text feature [oncogenic] present in test data point [True]
56 Text feature [found] present in test data point [True]
57 Text feature [suggest] present in test data point [True]
58 Text feature [recently] present in test data point [True]
59 Text feature [proliferation] present in test data point [True]
60 Text feature [treated] present in test data point [True]
61 Text feature [inhibited] present in test data point [True]
62 Text feature [increased] present in test data point [True]
64 Text feature [pathways] present in test data point [True]
65 Text feature [mutant] present in test data point [True]
66 Text feature [although] present in test data point [True]
67 Text feature [results] present in test data point [True]
68 Text feature [observed] present in test data point [True]
69 Text feature [3b] present in test data point [True]
70 Text feature [12] present in test data point [True]
71 Text feature [absence] present in test data point [True]
72 Text feature [total] present in test data point [True]
73 Text feature [increase] present in test data point [True]
74 Text feature [receptor] present in test data point [True]
75 Text feature [different] present in test data point [True]
76 Text feature [enhanced] present in test data point [True]
77 Text feature [mutation] present in test data point [True]
78 Text feature [described] present in test data point [True]
79 Text feature [constitutively] present in test data point [True]
80 Text feature [activate] present in test data point [True]
81 Text feature [concentrations] present in test data point [True]
82 Text feature [two] present in test data point [True]
83 Text feature [various] present in test data point [True]
84 Text feature [figure] present in test data point [True]
85 Text feature [recent] present in test data point [True]
86 Text feature [studies] present in test data point [True]
87 Text feature [inhibition] present in test data point [True]
88 Text feature [respectively] present in test data point [True]
89 Text feature [20] present in test data point [True]
90 Text feature [interestingly] present in test data point [True]
91 Text feature [mechanism] present in test data point [True]
92 Text feature [discussion] present in test data point [True]
93 Text feature [using] present in test data point [True]
94 Text feature [either] present in test data point [True]
96 Text feature [13] present in test data point [True]
97 Text feature [including] present in test data point [True]
98 Text feature [therapeutic] present in test data point [True]
99 Text feature [without] present in test data point [True]
Out of the top  100  features  75 are present in query point
```

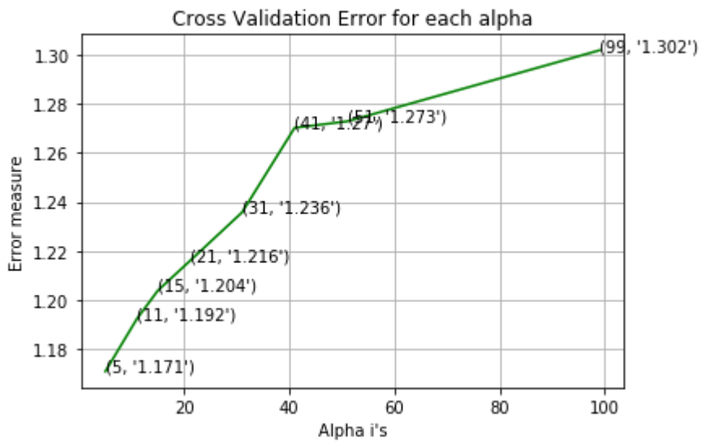
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [241]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=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)
knnLoss_train=log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",knnLoss_train)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
knnLoss_cv=log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",knnLoss_cv)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
knnLoss_test=log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",knnLoss_test)
```

```
for alpha = 5
Log Loss : 1.170760260907166
for alpha = 11
Log Loss : 1.1922217551857972
for alpha = 15
Log Loss : 1.203816242227566
for alpha = 21
Log Loss : 1.2157179805210214
for alpha = 31
Log Loss : 1.2357048446697156
for alpha = 41
Log Loss : 1.2702568840454405
for alpha = 51
Log Loss : 1.2728899169079537
for alpha = 99
Log Loss : 1.3019164020078928
```

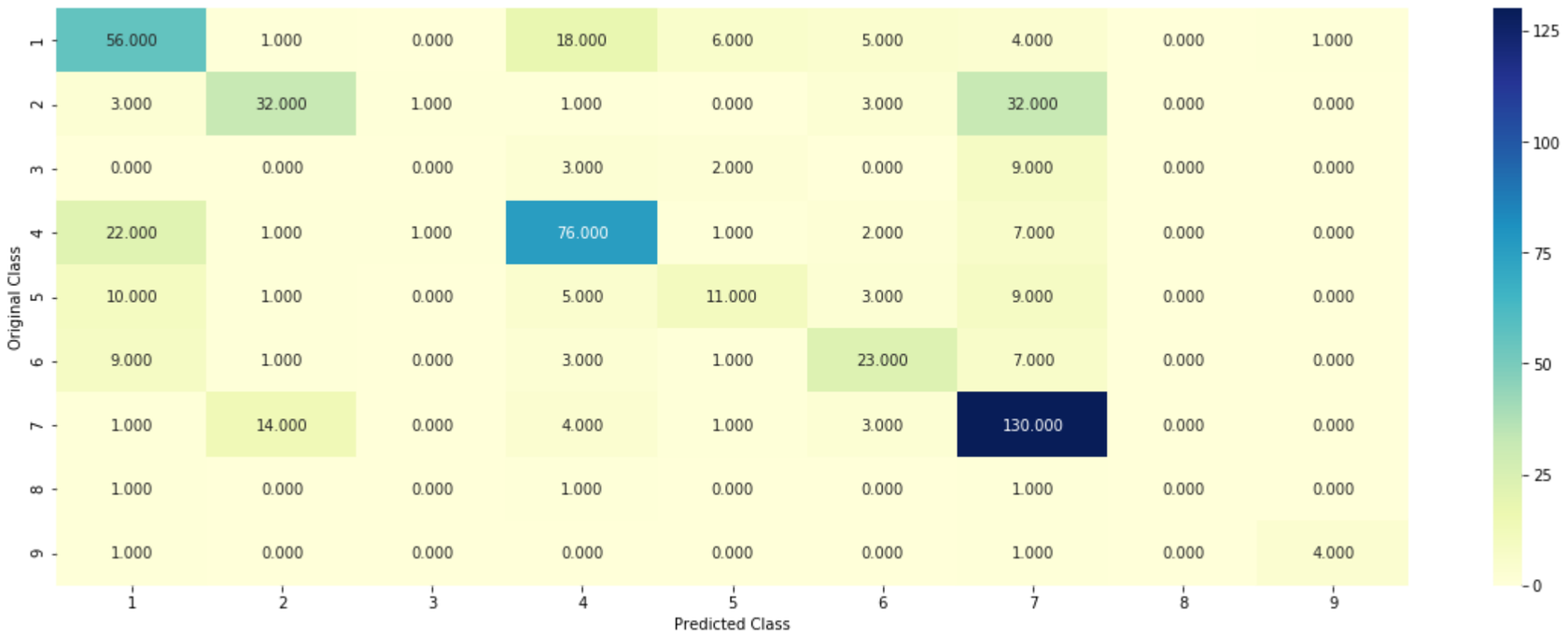



```
For values of best alpha =  5 The train log loss is: 0.9509309660615815
For values of best alpha =  5 The cross validation log loss is: 1.170760260907166
For values of best alpha =  5 The test log loss is: 1.1755732471983174
```

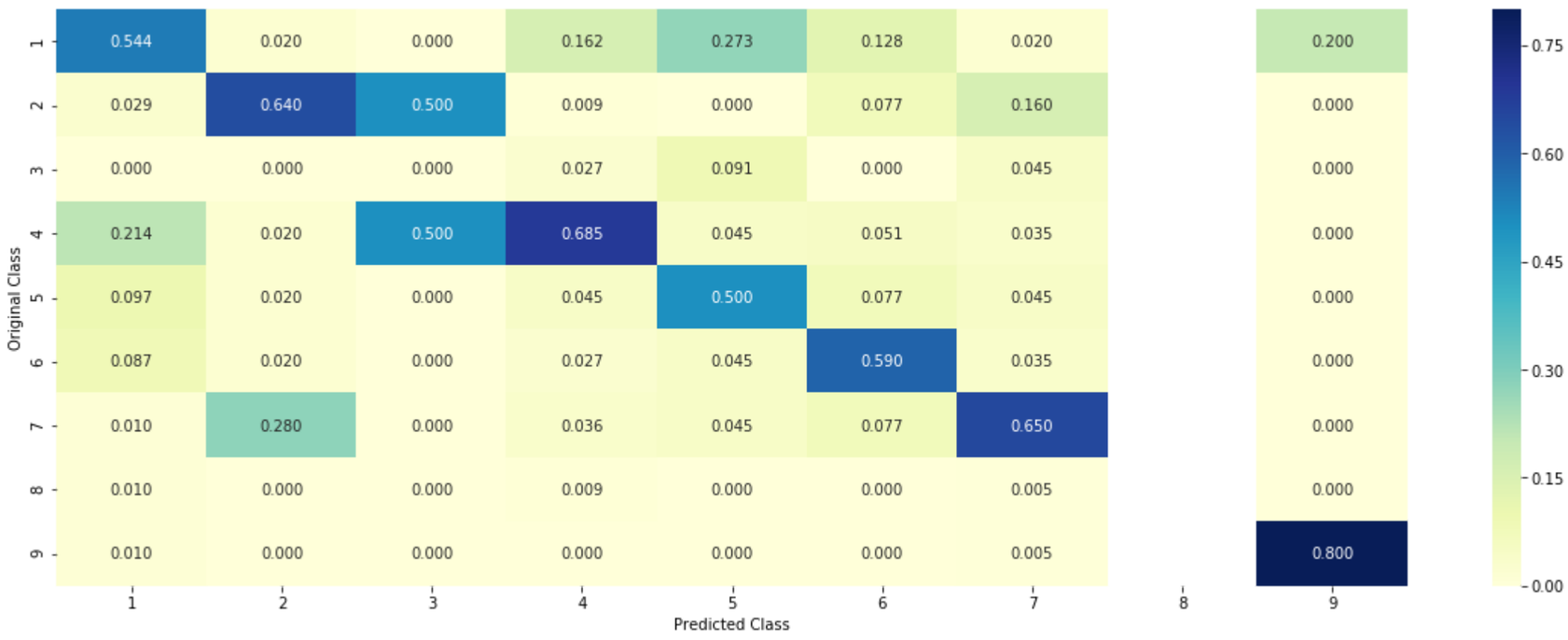
4.2.2. Testing the model with best hyper paramters

In [242]:
```
# 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])
knnmp=predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

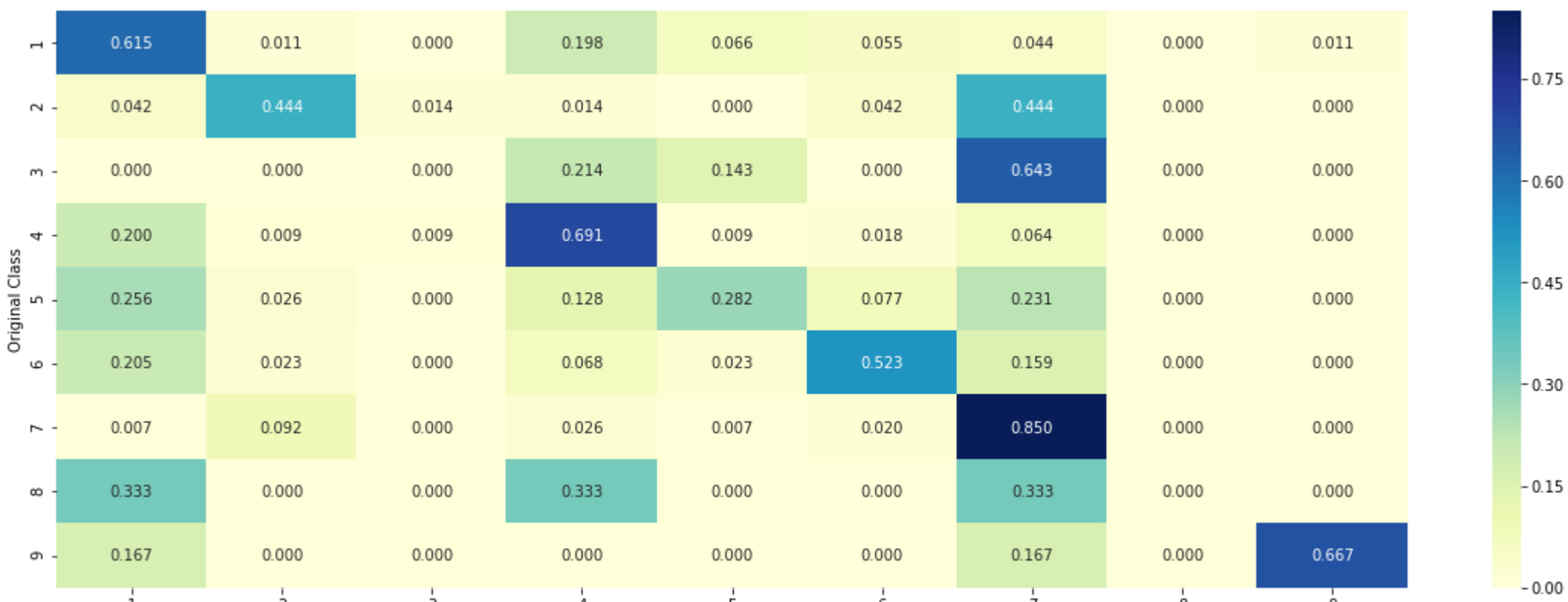
```
Log loss : 1.170760260907166
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [243]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
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
predicted_cls = sig_clf.predict(test_x_onehotCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_onehotCoding[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 : 4
Actual Class : 7
The  5  nearest neighbours of the test points belongs to classes [7 5 6 7 7]
Fequency of nearest points : Counter({7: 3, 5: 1, 6: 1})
```

4.2.4. Sample Query Point-2

In [244]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_onehotCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the 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 : 7
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [7 7 7 7 7]
Fequency of nearest points : Counter({7: 5})
```

4.3. Logistic Regression (TF-IDF)

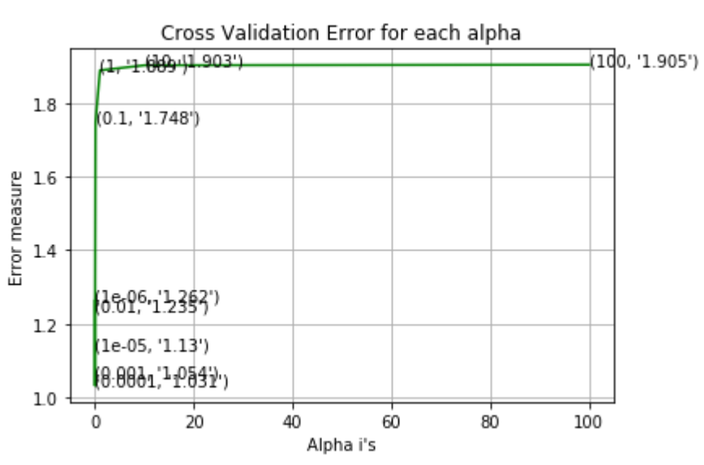
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [245]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.2622899291918217
for alpha = 1e-05
Log Loss : 1.129500707464076
for alpha = 0.0001
Log Loss : 1.030912509628833
for alpha = 0.001
Log Loss : 1.0540372989049644
for alpha = 0.01
Log Loss : 1.2349510959986052
for alpha = 0.1
Log Loss : 1.7481823347112109
for alpha = 1
Log Loss : 1.8894591915571086
for alpha = 10
Log Loss : 1.903195367679253
for alpha = 100
Log Loss : 1.9047940467274325
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5861527191925323
For values of best alpha =  0.0001 The cross validation log loss is: 1.030912509628833
For values of best alpha =  0.0001 The test log loss is: 1.020417833932263
```

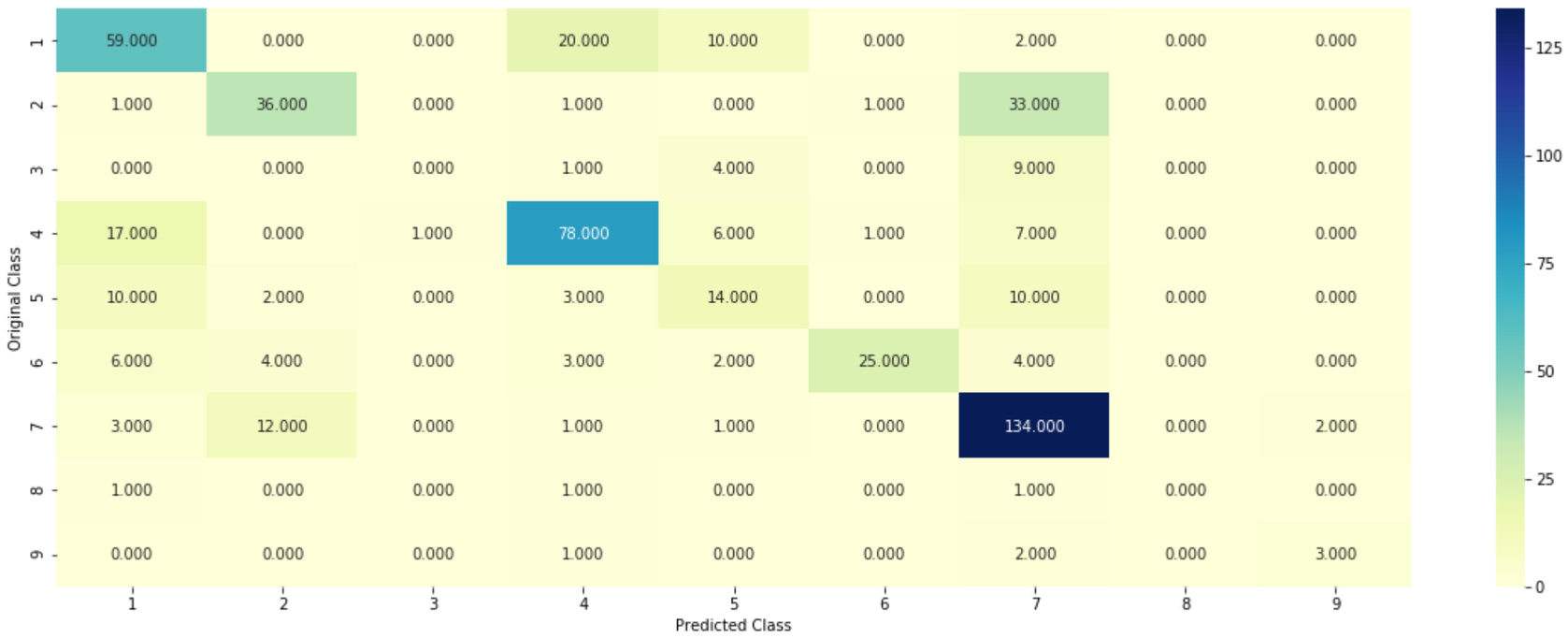
In [246]:
```
print("lrLossClassBalance_tfidf_train",lrLossClassBalance_tfidf_train)
print("lrLossClassBalance_tfidf_cv",lrLossClassBalance_tfidf_cv)
print("lrLossClassBalance_tfidf_test",lrLossClassBalance_tfidf_test)
```

```
lrLossClassBalance_tfidf_train 0.5861527191925323
lrLossClassBalance_tfidf_cv 1.030912509628833
lrLossClassBalance_tfidf_test 1.020417833932263
```

4.3.1.2. Testing the model with best hyper paramters

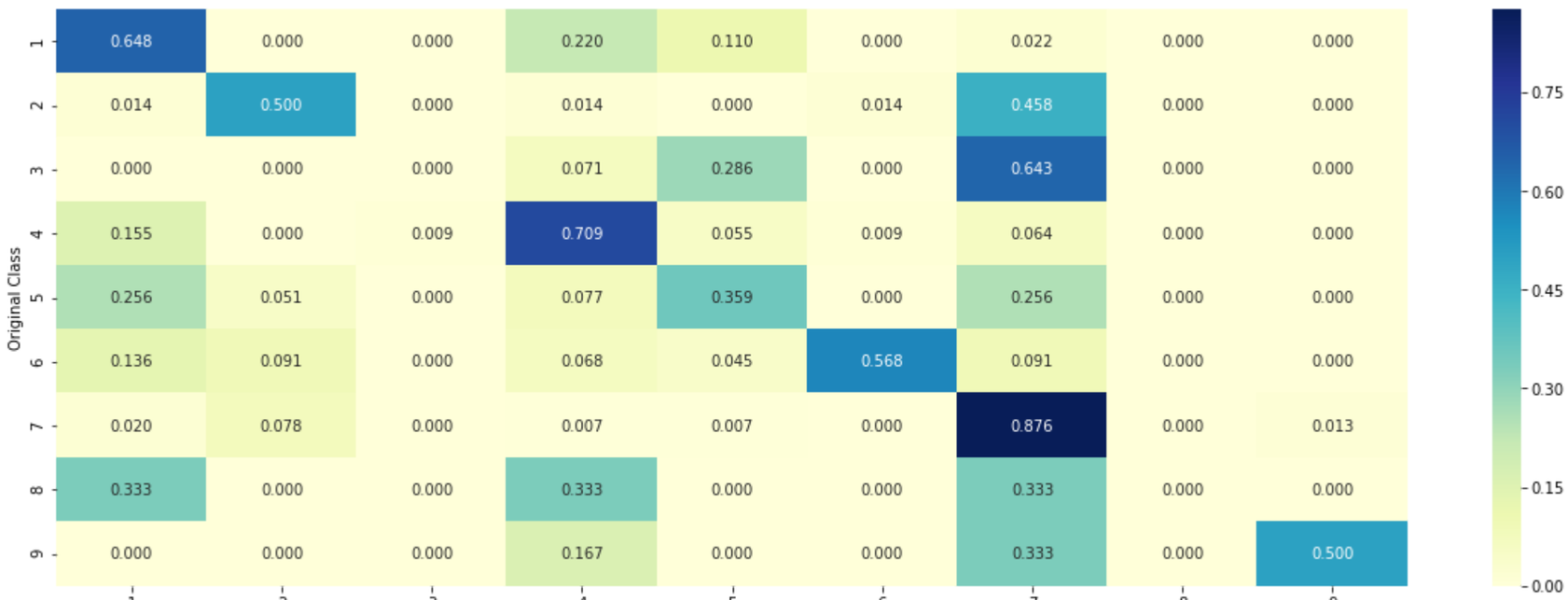
In [247]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.030912509628833
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```




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


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



4.3.1.3. Feature Importance

In [248]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most important features are present in our query point")
    print("-"*50)
    print("The features that are most important of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [249]:
```
# 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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0291 0.0502 0.0124 0.0175 0.0878 0.0174 0.7777 0.0048 0.003 ]]
Actual Class : 7
--------------------------------------------------
0 Text feature [constructs] present in test data point [True]
1 Text feature [washed] present in test data point [True]
3 Text feature [myeloid] present in test data point [True]
4 Text feature [individual] present in test data point [True]
9 Text feature [1a] present in test data point [True]
31 Text feature [allele] present in test data point [True]
46 Text feature [particular] present in test data point [True]
54 Text feature [target] present in test data point [True]
61 Text feature [blue] present in test data point [True]
66 Text feature [show] present in test data point [True]
98 Text feature [expression] present in test data point [True]
106 Text feature [treatment] present in test data point [True]
108 Text feature [next] present in test data point [True]
113 Text feature [mtor] present in test data point [True]
120 Text feature [assess] present in test data point [True]
127 Text feature [short] present in test data point [True]
150 Text feature [indicating] present in test data point [True]
192 Text feature [determine] present in test data point [True]
194 Text feature [similarly] present in test data point [True]
206 Text feature [measured] present in test data point [True]
214 Text feature [dependent] present in test data point [True]
220 Text feature [effects] present in test data point [True]
224 Text feature [indeed] present in test data point [True]
226 Text feature [incubated] present in test data point [True]
231 Text feature [exon] present in test data point [True]
232 Text feature [compared] present in test data point [True]
235 Text feature [involving] present in test data point [True]
238 Text feature [status] present in test data point [True]
240 Text feature [subset] present in test data point [True]
244 Text feature [possibility] present in test data point [True]
260 Text feature [regulation] present in test data point [True]
264 Text feature [non] present in test data point [True]
283 Text feature [levels] present in test data point [True]
288 Text feature [mutation] present in test data point [True]
293 Text feature [associated] present in test data point [True]
295 Text feature [directly] present in test data point [True]
298 Text feature [cell] present in test data point [True]
299 Text feature [vivo] present in test data point [True]
300 Text feature [test] present in test data point [True]
306 Text feature [endogenous] present in test data point [True]
312 Text feature [detected] present in test data point [True]
318 Text feature [recent] present in test data point [True]
319 Text feature [18] present in test data point [True]
332 Text feature [2000] present in test data point [True]
344 Text feature [genetic] present in test data point [True]
345 Text feature [comparison] present in test data point [True]
346 Text feature [identification] present in test data point [True]
350 Text feature [potential] present in test data point [True]
352 Text feature [six] present in test data point [True]
364 Text feature [phase] present in test data point [True]
373 Text feature [among] present in test data point [True]
379 Text feature [analyzed] present in test data point [True]
381 Text feature [48] present in test data point [True]
384 Text feature [formation] present in test data point [True]
400 Text feature [rna] present in test data point [True]
402 Text feature [linked] present in test data point [True]
416 Text feature [residue] present in test data point [True]
427 Text feature [early] present in test data point [True]
428 Text feature [spectrum] present in test data point [True]
431 Text feature [suppression] present in test data point [True]
432 Text feature [35] present in test data point [True]
444 Text feature [general] present in test data point [True]
445 Text feature [nih] present in test data point [True]
449 Text feature [encoding] present in test data point [True]
456 Text feature [s1] present in test data point [True]
459 Text feature [sensitivity] present in test data point [True]
460 Text feature [independent] present in test data point [True]
461 Text feature [hr] present in test data point [True]
477 Text feature [crystal] present in test data point [True]
479 Text feature [published] present in test data point [True]
490 Text feature [represent] present in test data point [True]
492 Text feature [include] present in test data point [True]
Out of the top  500  features  72 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [250]:
```
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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0313 0.0877 0.0104 0.0457 0.0178 0.0176 0.7758 0.0062 0.0076]]
Actual Class : 7
--------------------------------------------------
5 Text feature [association] present in test data point [True]
6 Text feature [ratio] present in test data point [True]
7 Text feature [representative] present in test data point [True]
9 Text feature [1a] present in test data point [True]
31 Text feature [allele] present in test data point [True]
42 Text feature [mg] present in test data point [True]
46 Text feature [particular] present in test data point [True]
54 Text feature [target] present in test data point [True]
59 Text feature [29] present in test data point [True]
60 Text feature [critical] present in test data point [True]
63 Text feature [basal] present in test data point [True]
65 Text feature [possible] present in test data point [True]
66 Text feature [show] present in test data point [True]
67 Text feature [product] present in test data point [True]
98 Text feature [expression] present in test data point [True]
106 Text feature [treatment] present in test data point [True]
108 Text feature [next] present in test data point [True]
120 Text feature [assess] present in test data point [True]
127 Text feature [short] present in test data point [True]
150 Text feature [indicating] present in test data point [True]
170 Text feature [11] present in test data point [True]
173 Text feature [positions] present in test data point [True]
175 Text feature [transcription] present in test data point [True]
194 Text feature [similarly] present in test data point [True]
201 Text feature [concentration] present in test data point [True]
204 Text feature [localization] present in test data point [True]
206 Text feature [measured] present in test data point [True]
208 Text feature [skin] present in test data point [True]
209 Text feature [displayed] present in test data point [True]
214 Text feature [dependent] present in test data point [True]
216 Text feature [2013] present in test data point [True]
220 Text feature [effects] present in test data point [True]
224 Text feature [indeed] present in test data point [True]
231 Text feature [exon] present in test data point [True]
232 Text feature [compared] present in test data point [True]
238 Text feature [status] present in test data point [True]
260 Text feature [regulation] present in test data point [True]
262 Text feature [forms] present in test data point [True]
264 Text feature [non] present in test data point [True]
274 Text feature [case] present in test data point [True]
283 Text feature [levels] present in test data point [True]
288 Text feature [mutation] present in test data point [True]
293 Text feature [associated] present in test data point [True]
295 Text feature [directly] present in test data point [True]
298 Text feature [cell] present in test data point [True]
299 Text feature [vivo] present in test data point [True]
300 Text feature [test] present in test data point [True]
306 Text feature [endogenous] present in test data point [True]
307 Text feature [enzyme] present in test data point [True]
312 Text feature [detected] present in test data point [True]
318 Text feature [recent] present in test data point [True]
319 Text feature [18] present in test data point [True]
332 Text feature [2000] present in test data point [True]
344 Text feature [genetic] present in test data point [True]
346 Text feature [identification] present in test data point [True]
350 Text feature [potential] present in test data point [True]
352 Text feature [six] present in test data point [True]
357 Text feature [less] present in test data point [True]
364 Text feature [phase] present in test data point [True]
366 Text feature [carrying] present in test data point [True]
371 Text feature [cohort] present in test data point [True]
373 Text feature [among] present in test data point [True]
379 Text feature [analyzed] present in test data point [True]
381 Text feature [48] present in test data point [True]
384 Text feature [formation] present in test data point [True]
400 Text feature [rna] present in test data point [True]
405 Text feature [domains] present in test data point [True]
408 Text feature [nrf2] present in test data point [True]
409 Text feature [cause] present in test data point [True]
427 Text feature [early] present in test data point [True]
428 Text feature [spectrum] present in test data point [True]
429 Text feature [single] present in test data point [True]
431 Text feature [suppression] present in test data point [True]
432 Text feature [35] present in test data point [True]
433 Text feature [e2] present in test data point [True]
447 Text feature [method] present in test data point [True]
456 Text feature [s1] present in test data point [True]
457 Text feature [myc] present in test data point [True]
458 Text feature [subunit] present in test data point [True]
459 Text feature [sensitivity] present in test data point [True]
460 Text feature [independent] present in test data point [True]
466 Text feature [sequencing] present in test data point [True]
467 Text feature [39] present in test data point [True]
477 Text feature [crystal] present in test data point [True]
479 Text feature [published] present in test data point [True]
490 Text feature [represent] present in test data point [True]
492 Text feature [include] present in test data point [True]
495 Text feature [structure] present in test data point [True]
Out of the top  500  features  88 are present in query point
```

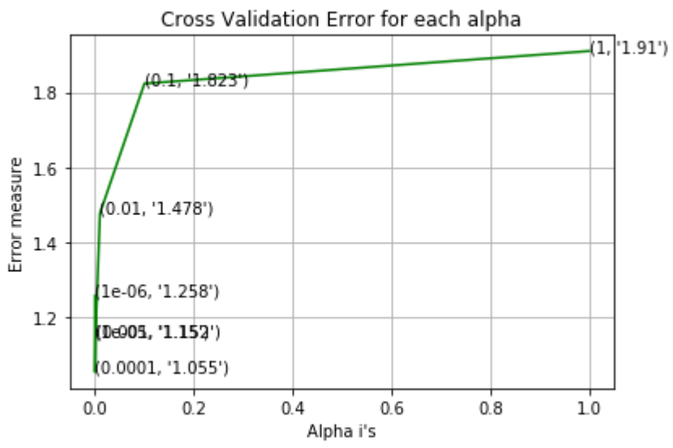
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [251]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.2580627618991984
for alpha = 1e-05
Log Loss : 1.1498620945241531
for alpha = 0.0001
Log Loss : 1.055245847824896
for alpha = 0.001
Log Loss : 1.1517398709169782
for alpha = 0.01
Log Loss : 1.4780197235315382
for alpha = 0.1
Log Loss : 1.8231464347451902
for alpha = 1
Log Loss : 1.909613629117402
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5816837139754962
For values of best alpha =  0.0001 The cross validation log loss is: 1.055245847824896
For values of best alpha =  0.0001 The test log loss is: 1.0368583306830037
```

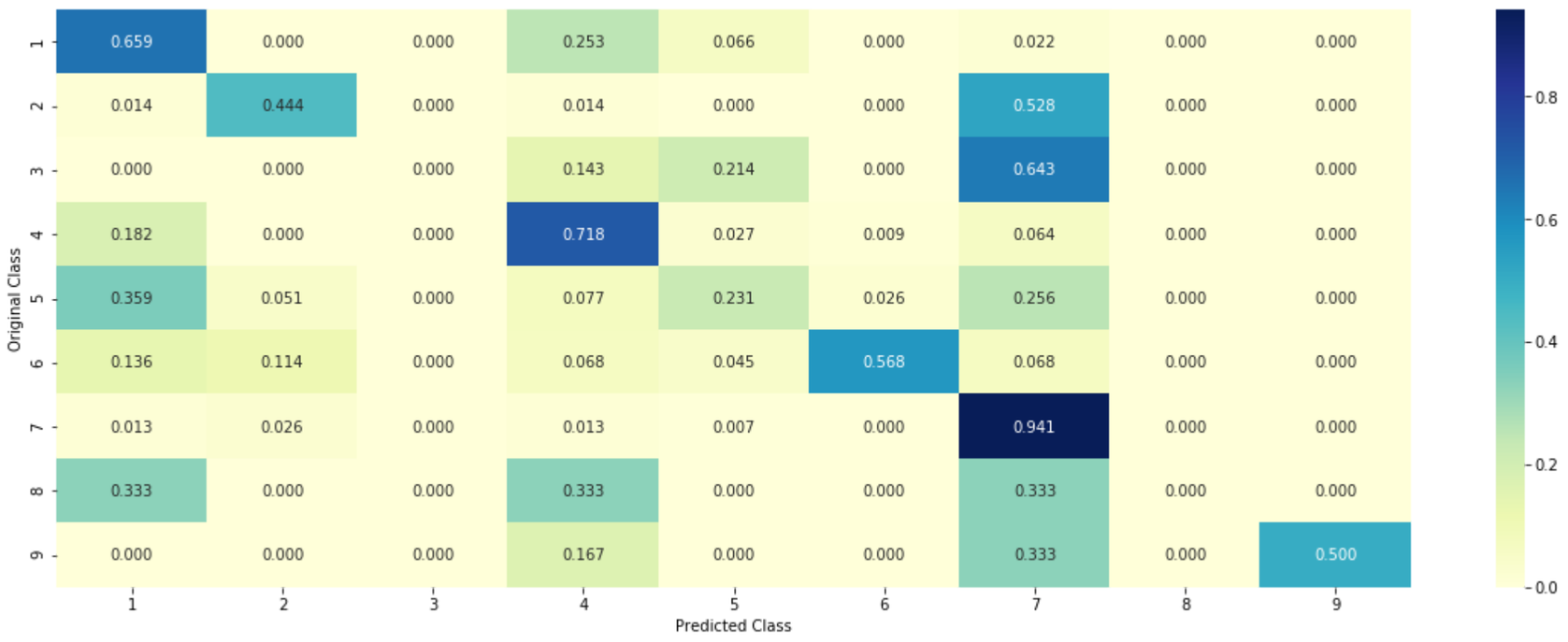
4.3.2.2. Testing model with best hyper parameters

In [252]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.055245847824896
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```


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


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



In [253]:
```
lrWithoutClassBalancemp
```

Out[253]:
```
0.3383458646616541
```

4.3.2.3. Feature Importance, Correctly Classified point

In [254]:
```
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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0319 0.0544 0.0111 0.0167 0.0676 0.0183 0.7926 0.0055 0.002 ]]
Actual Class : 7
--------------------------------------------------
0 Text feature [1a] present in test data point [True]
2 Text feature [according] present in test data point [True]
3 Text feature [assays] present in test data point [True]
4 Text feature [2008] present in test data point [True]
5 Text feature [experiments] present in test data point [True]
6 Text feature [demonstrate] present in test data point [True]
7 Text feature [expression] present in test data point [True]
11 Text feature [cell] present in test data point [True]
25 Text feature [48] present in test data point [True]
43 Text feature [target] present in test data point [True]
61 Text feature [rna] present in test data point [True]
62 Text feature [primers] present in test data point [True]
64 Text feature [non] present in test data point [True]
66 Text feature [indicating] present in test data point [True]
76 Text feature [s1] present in test data point [True]
103 Text feature [associated] present in test data point [True]
110 Text feature [report] present in test data point [True]
129 Text feature [recent] present in test data point [True]
143 Text feature [hr] present in test data point [True]
149 Text feature [comparison] present in test data point [True]
150 Text feature [endogenous] present in test data point [True]
157 Text feature [members] present in test data point [True]
178 Text feature [one] present in test data point [True]
207 Text feature [data] present in test data point [True]
208 Text feature [early] present in test data point [True]
224 Text feature [analysis] present in test data point [True]
227 Text feature [incubated] present in test data point [True]
247 Text feature [include] present in test data point [True]
251 Text feature [identification] present in test data point [True]
252 Text feature [methods] present in test data point [True]
260 Text feature [phosphatase] present in test data point [True]
264 Text feature [compared] present in test data point [True]
266 Text feature [18] present in test data point [True]
269 Text feature [independent] present in test data point [True]
285 Text feature [myeloid] present in test data point [True]
302 Text feature [published] present in test data point [True]
303 Text feature [general] present in test data point [True]
319 Text feature [present] present in test data point [True]
322 Text feature [similarly] present in test data point [True]
331 Text feature [1998] present in test data point [True]
333 Text feature [s2] present in test data point [True]
340 Text feature [introduction] present in test data point [True]
342 Text feature [status] present in test data point [True]
366 Text feature [isolated] present in test data point [True]
371 Text feature [alleles] present in test data point [True]
372 Text feature [significantly] present in test data point [True]
387 Text feature [nih] present in test data point [True]
395 Text feature [vivo] present in test data point [True]
401 Text feature [transcriptional] present in test data point [True]
403 Text feature [suppression] present in test data point [True]
422 Text feature [1996] present in test data point [True]
431 Text feature [short] present in test data point [True]
436 Text feature [finding] present in test data point [True]
438 Text feature [dependent] present in test data point [True]
444 Text feature [without] present in test data point [True]
448 Text feature [blue] present in test data point [True]
459 Text feature [subset] present in test data point [True]
460 Text feature [inhibit] present in test data point [True]
462 Text feature [show] present in test data point [True]
470 Text feature [possibility] present in test data point [True]
471 Text feature [induced] present in test data point [True]
477 Text feature [primary] present in test data point [True]
486 Text feature [despite] present in test data point [True]
490 Text feature [cancer] present in test data point [True]
494 Text feature [particular] present in test data point [True]
495 Text feature [among] present in test data point [True]
496 Text feature [phenotype] present in test data point [True]
Out of the top  500  features  67 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [255]:
```
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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0299 0.0956 0.0117 0.0394 0.0179 0.0184 0.7742 0.006  0.0071]]
Actual Class : 7
--------------------------------------------------
0 Text feature [1a] present in test data point [True]
1 Text feature [full] present in test data point [True]
2 Text feature [according] present in test data point [True]
4 Text feature [2008] present in test data point [True]
5 Text feature [experiments] present in test data point [True]
6 Text feature [demonstrate] present in test data point [True]
7 Text feature [expression] present in test data point [True]
11 Text feature [cell] present in test data point [True]
25 Text feature [48] present in test data point [True]
33 Text feature [domains] present in test data point [True]
34 Text feature [critical] present in test data point [True]
43 Text feature [target] present in test data point [True]
58 Text feature [sequencing] present in test data point [True]
61 Text feature [rna] present in test data point [True]
62 Text feature [primers] present in test data point [True]
64 Text feature [non] present in test data point [True]
65 Text feature [subunit] present in test data point [True]
66 Text feature [indicating] present in test data point [True]
76 Text feature [s1] present in test data point [True]
86 Text feature [positions] present in test data point [True]
87 Text feature [ratio] present in test data point [True]
103 Text feature [associated] present in test data point [True]
105 Text feature [product] present in test data point [True]
110 Text feature [report] present in test data point [True]
129 Text feature [recent] present in test data point [True]
150 Text feature [endogenous] present in test data point [True]
157 Text feature [members] present in test data point [True]
178 Text feature [one] present in test data point [True]
200 Text feature [29] present in test data point [True]
207 Text feature [data] present in test data point [True]
208 Text feature [early] present in test data point [True]
224 Text feature [analysis] present in test data point [True]
233 Text feature [forms] present in test data point [True]
235 Text feature [skin] present in test data point [True]
237 Text feature [e2] present in test data point [True]
240 Text feature [association] present in test data point [True]
243 Text feature [14] present in test data point [True]
247 Text feature [include] present in test data point [True]
248 Text feature [displayed] present in test data point [True]
251 Text feature [identification] present in test data point [True]
252 Text feature [methods] present in test data point [True]
264 Text feature [compared] present in test data point [True]
266 Text feature [18] present in test data point [True]
269 Text feature [independent] present in test data point [True]
273 Text feature [method] present in test data point [True]
283 Text feature [side] present in test data point [True]
302 Text feature [published] present in test data point [True]
311 Text feature [transfected] present in test data point [True]
319 Text feature [present] present in test data point [True]
321 Text feature [myc] present in test data point [True]
322 Text feature [similarly] present in test data point [True]
331 Text feature [1998] present in test data point [True]
333 Text feature [s2] present in test data point [True]
340 Text feature [introduction] present in test data point [True]
341 Text feature [p53] present in test data point [True]
342 Text feature [status] present in test data point [True]
343 Text feature [basal] present in test data point [True]
371 Text feature [alleles] present in test data point [True]
372 Text feature [significantly] present in test data point [True]
382 Text feature [39] present in test data point [True]
384 Text feature [least] present in test data point [True]
395 Text feature [vivo] present in test data point [True]
396 Text feature [carrying] present in test data point [True]
397 Text feature [iii] present in test data point [True]
401 Text feature [transcriptional] present in test data point [True]
403 Text feature [suppression] present in test data point [True]
410 Text feature [single] present in test data point [True]
414 Text feature [length] present in test data point [True]
417 Text feature [tested] present in test data point [True]
422 Text feature [1996] present in test data point [True]
423 Text feature [luciferase] present in test data point [True]
431 Text feature [short] present in test data point [True]
438 Text feature [dependent] present in test data point [True]
442 Text feature [possible] present in test data point [True]
444 Text feature [without] present in test data point [True]
460 Text feature [inhibit] present in test data point [True]
461 Text feature [representative] present in test data point [True]
462 Text feature [show] present in test data point [True]
468 Text feature [subjected] present in test data point [True]
471 Text feature [induced] present in test data point [True]
472 Text feature [cause] present in test data point [True]
477 Text feature [primary] present in test data point [True]
486 Text feature [despite] present in test data point [True]
490 Text feature [cancer] present in test data point [True]
494 Text feature [particular] present in test data point [True]
495 Text feature [among] present in test data point [True]
Out of the top  500  features  86 are present in query point
```

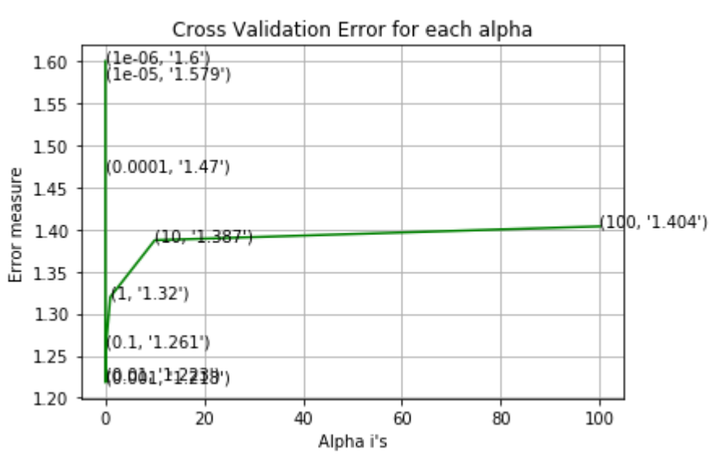
4.3.B Logistic Regression (COUNT VECTORIZER)

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [256]:
```
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_count, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_count, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_count)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.5996698881045526
for alpha = 1e-05
Log Loss : 1.5794690221362369
for alpha = 0.0001
Log Loss : 1.4701040733797988
for alpha = 0.001
Log Loss : 1.218471182288538
for alpha = 0.01
Log Loss : 1.2228162767992852
for alpha = 0.1
Log Loss : 1.2612237767977659
for alpha = 1
Log Loss : 1.31981011498038
for alpha = 10
Log Loss : 1.3872212211307773
for alpha = 100
Log Loss : 1.4036206847143773
```




```
For values of best alpha =  0.001 The train log loss is: 0.7456427016998882
For values of best alpha =  0.001 The cross validation log loss is: 1.218471182288538
For values of best alpha =  0.001 The test log loss is: 1.1809193277964423
```

4.3.1.2. Testing the model with best hyper paramters

In [257]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
lrClassBalanceCountmp=predict_and_plot_confusion_matrix(train_x_onehotCoding_count, train_y, cv_x_onehotCoding_count, cv_y, clf)
```

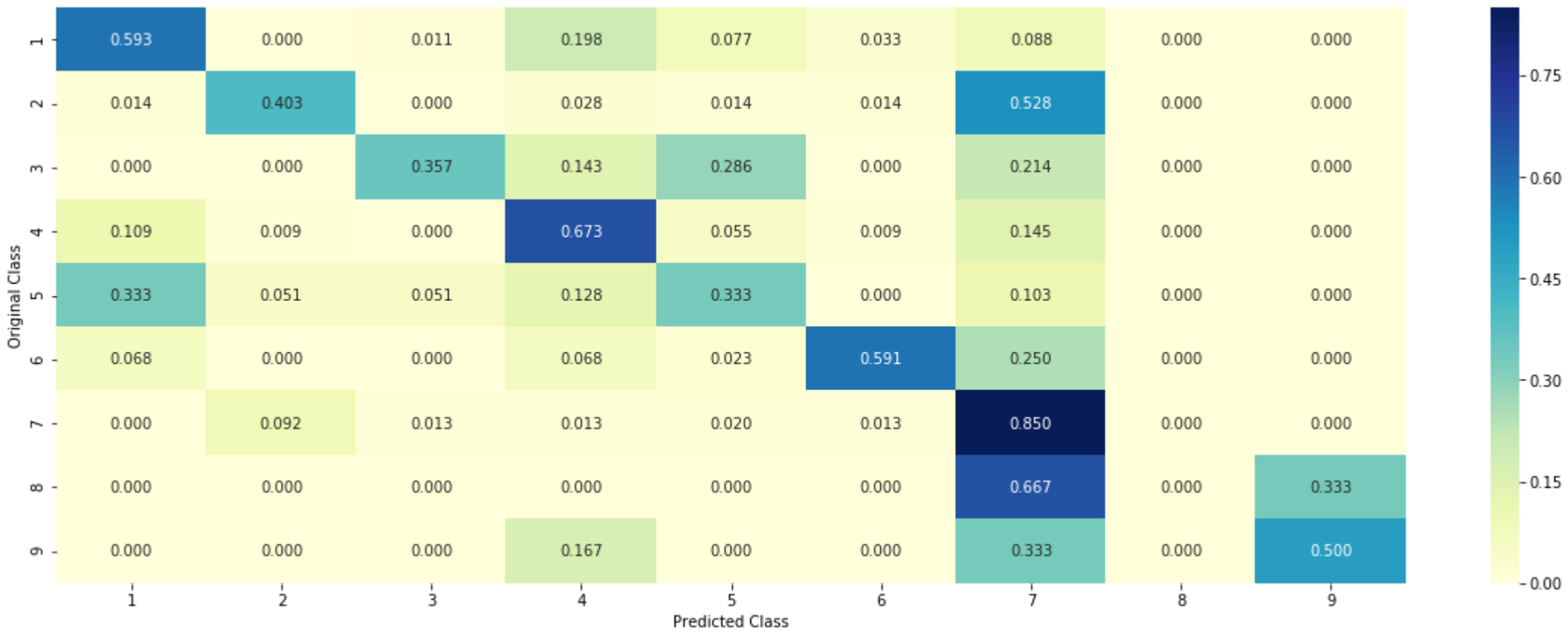
```
Log loss : 1.218471182288538
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [258]:
```
def get_imp_feature_names_countVec(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding_onehot.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [259]:
```
# 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_count,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_count[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_count[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
# get_impfeature_names_countVec(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.149  0.1257 0.0212 0.146  0.0559 0.0333 0.4516 0.0056 0.0119]]
Actual Class : 7
--------------------------------------------------
```

4.3.1.3.2. Incorrectly Classified point

In [260]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_count[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_count[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
# get_impfeature_names_countVec(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

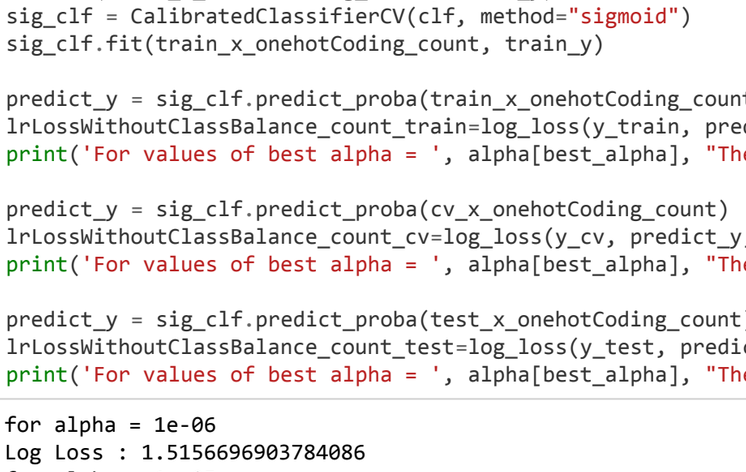
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0217 0.0263 0.0061 0.0067 0.0135 0.0019 0.9102 0.006  0.0076]]
Actual Class : 7
--------------------------------------------------
```

4.3.1. WITHOUT Class balancing

4.3.1.1. Hyper paramter tuning

In [261]:
```
alpha = [10 ** x for x in range(-6, 3)]
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_count, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_count, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_count)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.5156696903784086
for alpha = 1e-05
Log Loss : 1.492389735695097
for alpha = 0.0001
Log Loss : 1.4473916587324844
for alpha = 0.001
Log Loss : 1.227599516079416
for alpha = 0.01
Log Loss : 1.222456868986534
for alpha = 0.1
Log Loss : 1.2507518273317457
for alpha = 1
Log Loss : 1.303814973838129
for alpha = 10
Log Loss : 1.3537346146253044
for alpha = 100
Log Loss : 1.366279314705432
```

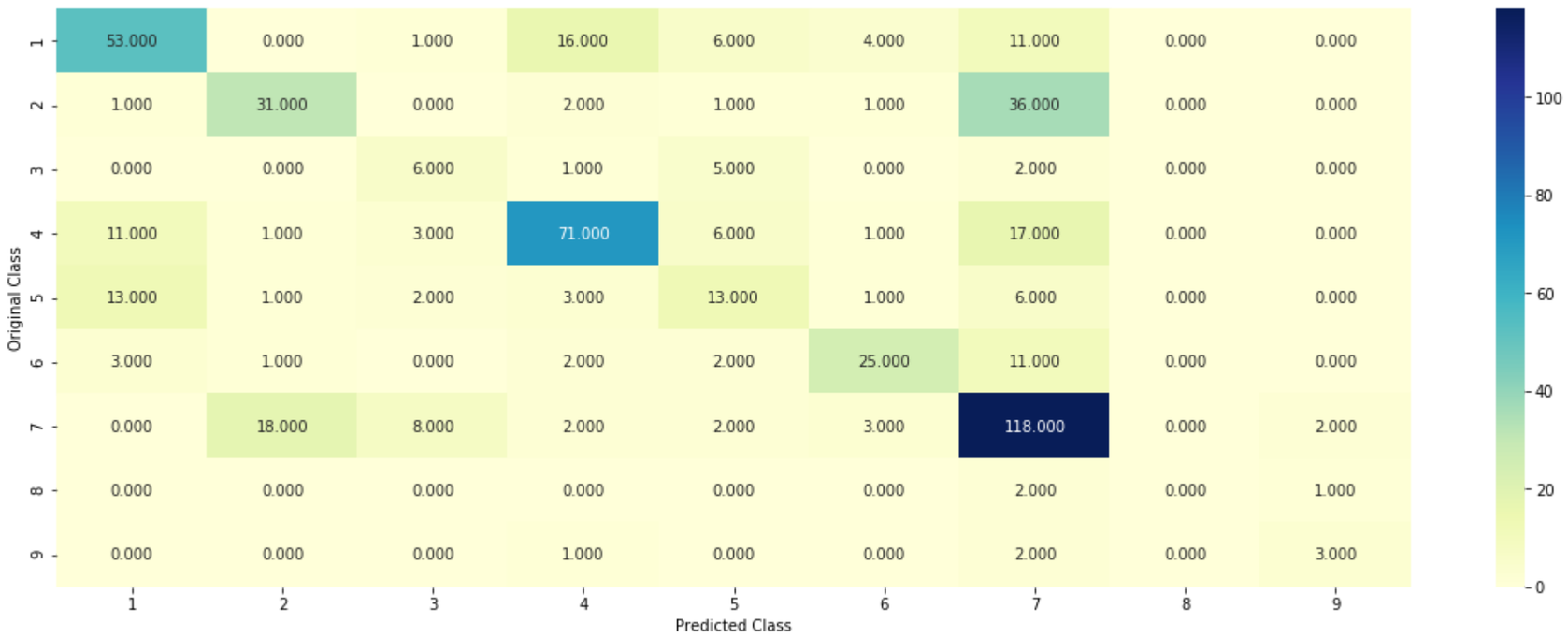



```
For values of best alpha =  0.01 The train log loss is: 0.7177315720526737
For values of best alpha =  0.01 The cross validation log loss is: 1.2228162767992852
For values of best alpha =  0.01 The test log loss is: 1.165539115332345
```

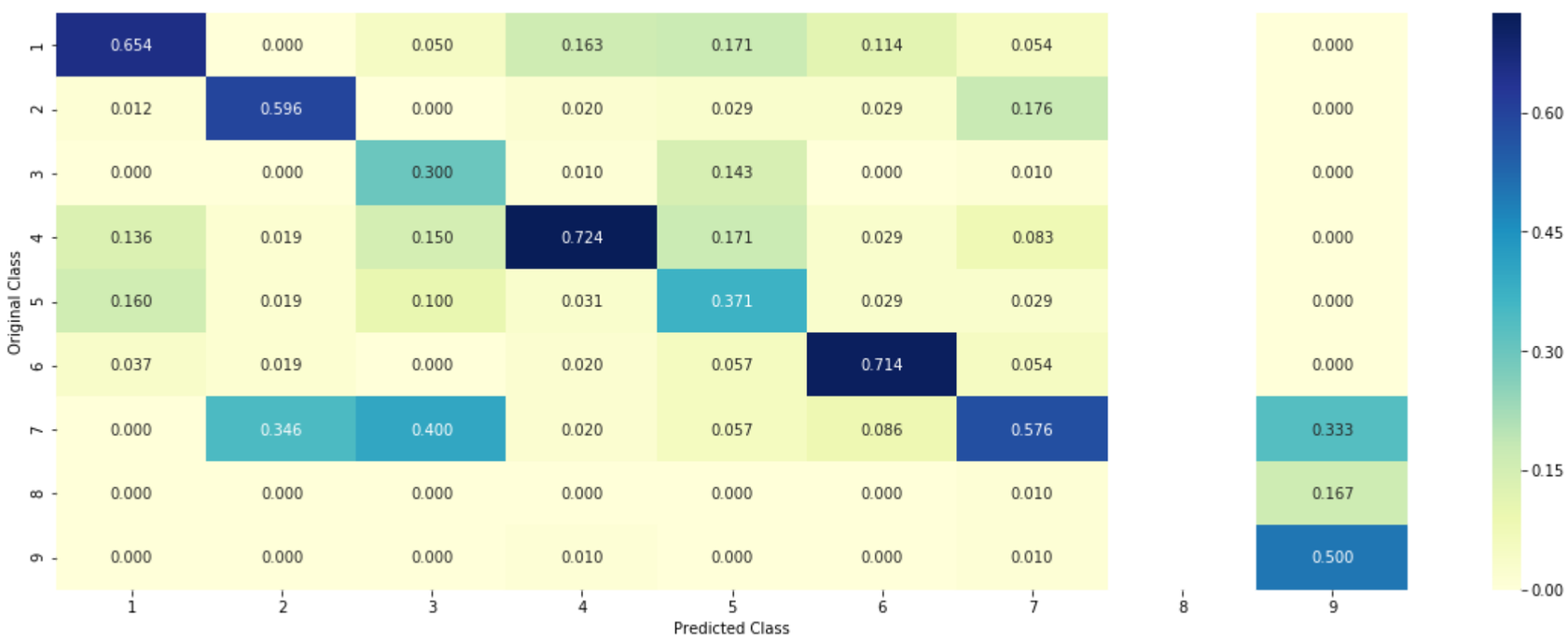
4.3.1.2. Testing the model with best hyper paramters

In [262]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
lrWithoutClassBalanceCountmp=predict_and_plot_confusion_matrix(train_x_onehotCoding_count, train_y, cv_x_onehotCoding_count, cv_y, clf)
```

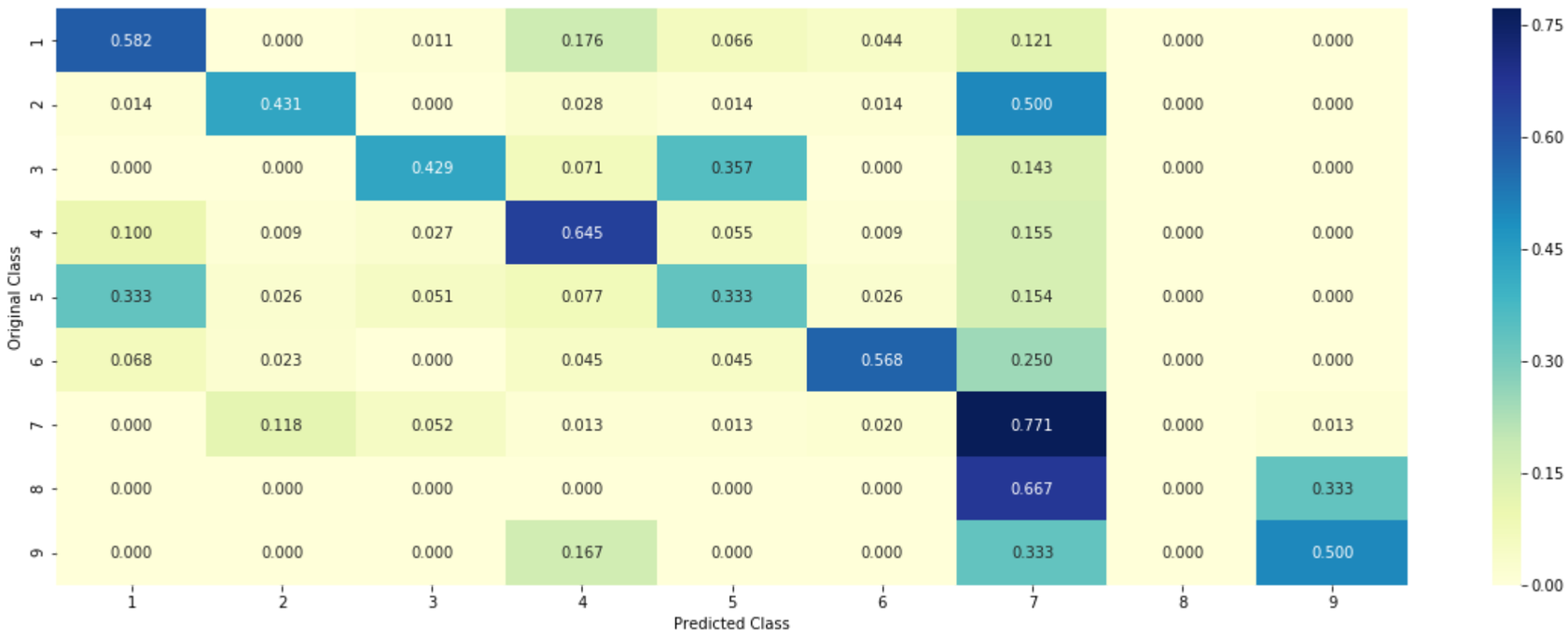
```
Log loss : 1.2228162767992852
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3.1. Correctly Classified point

In [263]:
```
# from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_count,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_count[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_count[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
# get_impfeature_names_countVec(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1534 0.1326 0.0113 0.1477 0.053  0.0308 0.4574 0.0065 0.0073]]
Actual Class : 7
--------------------------------------------------
```

4.3.1.3.2. Incorrectly Classified point

In [264]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_count[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_count[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
# get_impfeature_names_countVec(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.220e-02 1.230e-02 3.000e-04 4.300e-03 5.400e-03 5.000e-04 9.588e-01
  4.900e-03 1.300e-03]]
Actual Class : 7
--------------------------------------------------
```

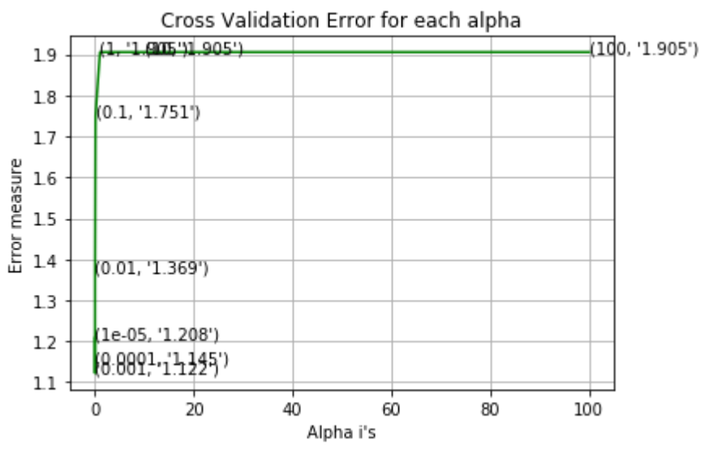
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [265]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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



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

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

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

```
for C = 1e-05
Log Loss : 1.2081034331839902
for C = 0.0001
Log Loss : 1.1451954340085735
for C = 0.001
Log Loss : 1.122120605502989
for C = 0.01
Log Loss : 1.3692937745109974
for C = 0.1
Log Loss : 1.75054935095415
for C = 1
Log Loss : 1.9052515780887211
for C = 10
Log Loss : 1.9052515600548747
for C = 100
Log Loss : 1.9052516961543047
```

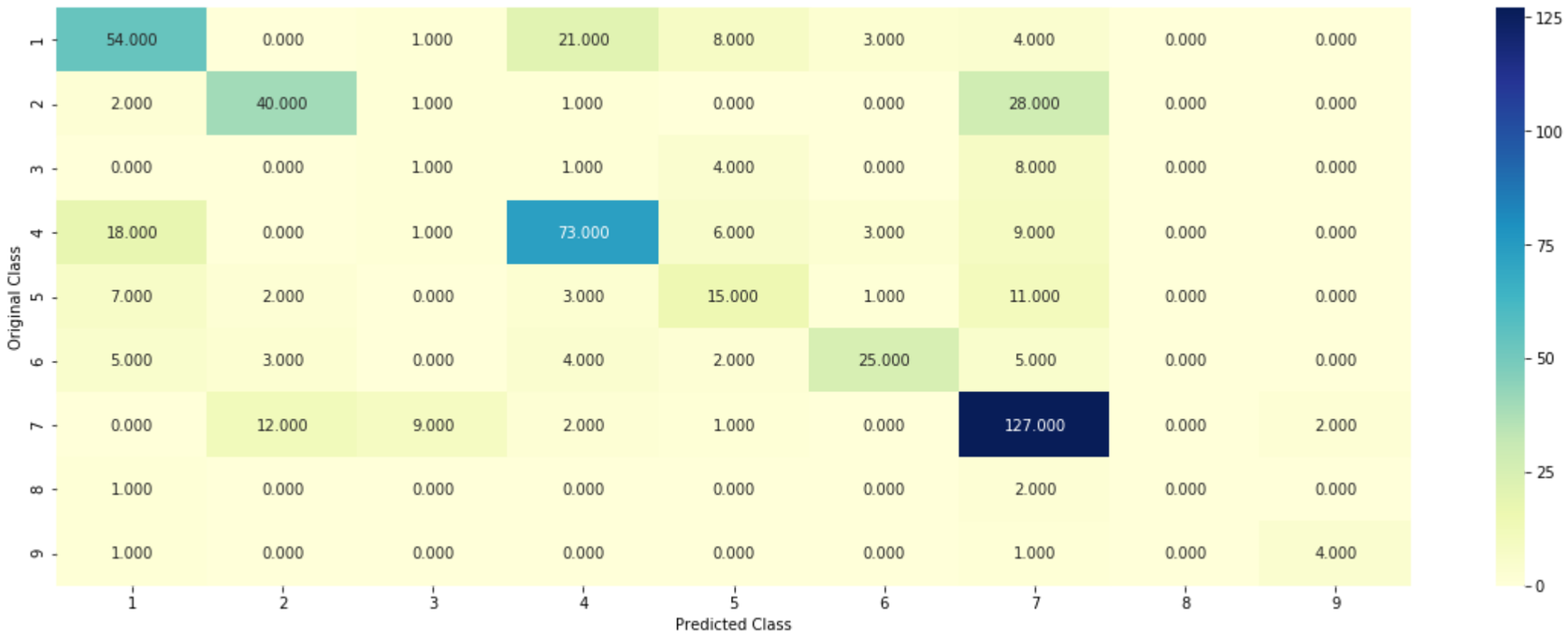



```
For values of best alpha =  0.001 The train log loss is: 0.819170194428988
For values of best alpha =  0.001 The cross validation log loss is: 1.122120605502989
For values of best alpha =  0.001 The test log loss is: 1.1444629127143888
```

4.4.2. Testing model with best hyper parameters

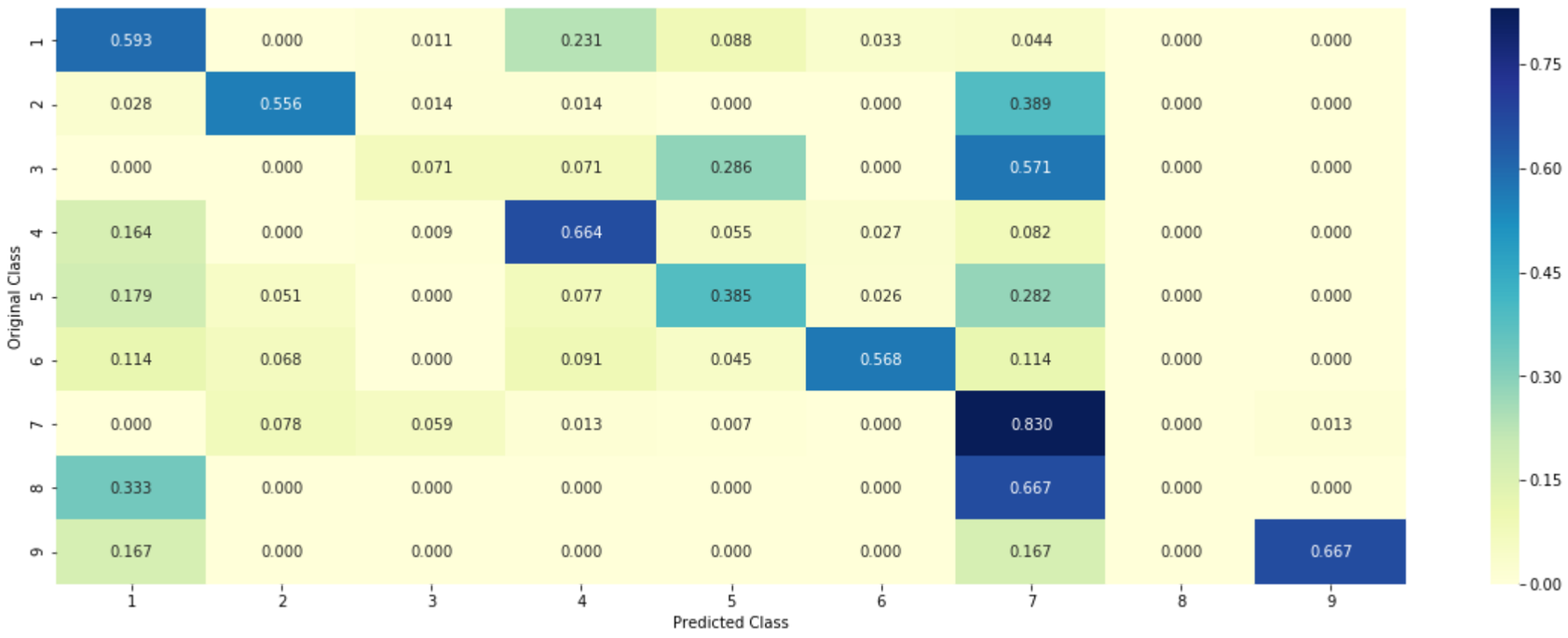
In [266]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 1.122120605502989
Number of mis-classified points : 0.36278195488721804
-------------------- 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 [267]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0807 0.0374 0.0174 0.0926 0.0649 0.0201 0.6762 0.0053 0.0054]]
Actual Class : 7
--------------------------------------------------
363 Text feature [manner] present in test data point [True]
365 Text feature [also] present in test data point [True]
366 Text feature [demonstrate] present in test data point [True]
367 Text feature [vivo] present in test data point [True]
368 Text feature [directly] present in test data point [True]
370 Text feature [2000] present in test data point [True]
372 Text feature [targets] present in test data point [True]
374 Text feature [contact] present in test data point [True]
375 Text feature [pcr] present in test data point [True]
376 Text feature [myeloid] present in test data point [True]
378 Text feature [indicating] present in test data point [True]
381 Text feature [among] present in test data point [True]
386 Text feature [compared] present in test data point [True]
388 Text feature [35] present in test data point [True]
389 Text feature [western] present in test data point [True]
390 Text feature [tumorigenesis] present in test data point [True]
395 Text feature [assays] present in test data point [True]
396 Text feature [represent] present in test data point [True]
398 Text feature [early] present in test data point [True]
402 Text feature [indicate] present in test data point [True]
403 Text feature [results] present in test data point [True]
404 Text feature [figures] present in test data point [True]
405 Text feature [active] present in test data point [True]
406 Text feature [sequences] present in test data point [True]
410 Text feature [members] present in test data point [True]
411 Text feature [although] present in test data point [True]
415 Text feature [experiments] present in test data point [True]
417 Text feature [drug] present in test data point [True]
418 Text feature [yet] present in test data point [True]
419 Text feature [six] present in test data point [True]
421 Text feature [potential] present in test data point [True]
423 Text feature [cell] present in test data point [True]
424 Text feature [cases] present in test data point [True]
426 Text feature [therapy] present in test data point [True]
428 Text feature [sufficient] present in test data point [True]
429 Text feature [alternative] present in test data point [True]
Out of the top  500  features  36 are present in query point
```

4.4.3.2. For Incorrectly classified point

In [268]:
```
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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0697 0.0649 0.0138 0.0827 0.0178 0.0299 0.7125 0.0041 0.0046]]
Actual Class : 7
--------------------------------------------------
363 Text feature [manner] present in test data point [True]
365 Text feature [also] present in test data point [True]
366 Text feature [demonstrate] present in test data point [True]
367 Text feature [vivo] present in test data point [True]
368 Text feature [directly] present in test data point [True]
370 Text feature [2000] present in test data point [True]
371 Text feature [established] present in test data point [True]
372 Text feature [targets] present in test data point [True]
374 Text feature [contact] present in test data point [True]
375 Text feature [pcr] present in test data point [True]
377 Text feature [whole] present in test data point [True]
378 Text feature [indicating] present in test data point [True]
381 Text feature [among] present in test data point [True]
383 Text feature [three] present in test data point [True]
385 Text feature [skin] present in test data point [True]
386 Text feature [compared] present in test data point [True]
388 Text feature [35] present in test data point [True]
390 Text feature [tumorigenesis] present in test data point [True]
391 Text feature [promoter] present in test data point [True]
396 Text feature [represent] present in test data point [True]
398 Text feature [early] present in test data point [True]
399 Text feature [progression] present in test data point [True]
400 Text feature [displayed] present in test data point [True]
401 Text feature [resistant] present in test data point [True]
402 Text feature [indicate] present in test data point [True]
403 Text feature [results] present in test data point [True]
404 Text feature [figures] present in test data point [True]
405 Text feature [active] present in test data point [True]
406 Text feature [sequences] present in test data point [True]
407 Text feature [history] present in test data point [True]
410 Text feature [members] present in test data point [True]
411 Text feature [although] present in test data point [True]
412 Text feature [39] present in test data point [True]
415 Text feature [experiments] present in test data point [True]
417 Text feature [drug] present in test data point [True]
418 Text feature [yet] present in test data point [True]
419 Text feature [six] present in test data point [True]
421 Text feature [potential] present in test data point [True]
423 Text feature [cell] present in test data point [True]
424 Text feature [cases] present in test data point [True]
426 Text feature [therapy] present in test data point [True]
427 Text feature [antibody] present in test data point [True]
430 Text feature [2013] present in test data point [True]
432 Text feature [13] present in test data point [True]
Out of the top  500  features  44 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

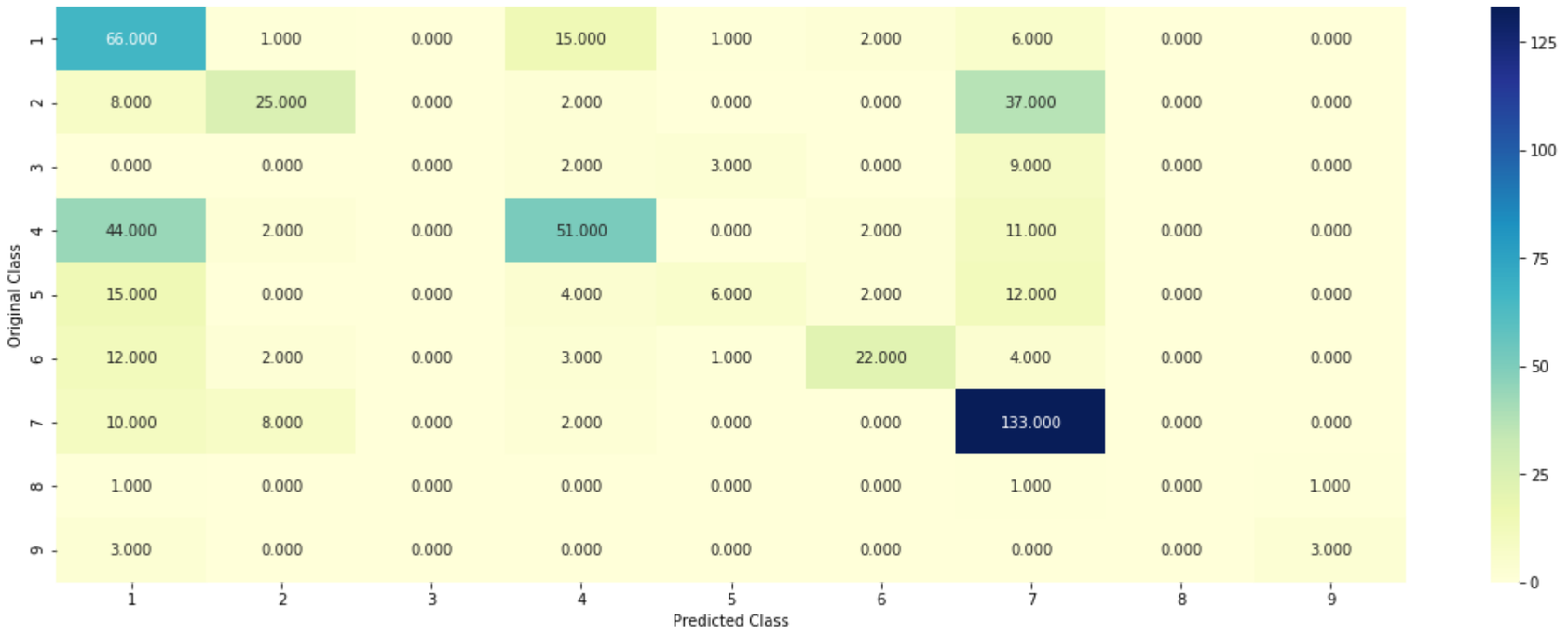
In [269]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2160121060947022
for n_estimators = 100 and max depth =  10
Log Loss : 1.2472442309168115
for n_estimators = 200 and max depth =  5
Log Loss : 1.2040943622113887
for n_estimators = 200 and max depth =  10
Log Loss : 1.2368460709698479
for n_estimators = 500 and max depth =  5
Log Loss : 1.198888627192858
for n_estimators = 500 and max depth =  10
Log Loss : 1.2304574191323114
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1922970805549726
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2280261435665731
for n_estimators = 2000 and max depth =  5
Log Loss : 1.193157282492909
for n_estimators = 2000 and max depth =  10
Log Loss : 1.228716997830508
For values of best estimator =  1000 The train log loss is: 0.8451884253934411
For values of best estimator =  1000 The cross validation log loss is: 1.1922970805549726
For values of best estimator =  1000 The test log loss is: 1.2083965634494773
```

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

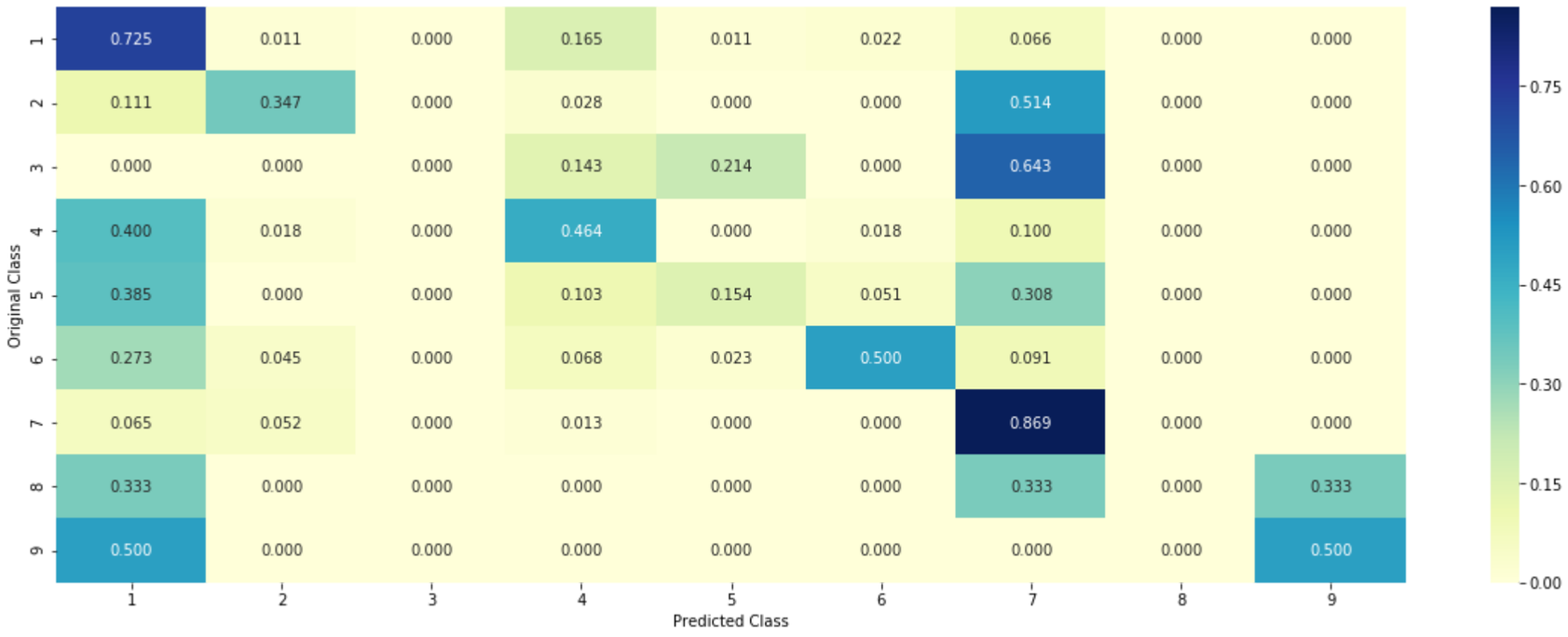
In [270]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.1922970805549726
Number of mis-classified points : 0.424812030075188
-------------------- 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 [271]:
```
# 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_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0923 0.1392 0.0291 0.1119 0.0549 0.043  0.4938 0.0102 0.0256]]
Actual Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
3 Text feature [suppressor] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [constitutive] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [missense] present in test data point [True]
12 Text feature [activated] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
16 Text feature [signaling] present in test data point [True]
18 Text feature [therapy] present in test data point [True]
19 Text feature [inhibitor] present in test data point [True]
21 Text feature [akt] present in test data point [True]
22 Text feature [protein] present in test data point [True]
23 Text feature [cells] present in test data point [True]
25 Text feature [variants] present in test data point [True]
26 Text feature [activate] present in test data point [True]
29 Text feature [erk] present in test data point [True]
30 Text feature [therapeutic] present in test data point [True]
35 Text feature [kinases] present in test data point [True]
36 Text feature [inactivation] present in test data point [True]
38 Text feature [transforming] present in test data point [True]
40 Text feature [trials] present in test data point [True]
41 Text feature [drug] present in test data point [True]
42 Text feature [functional] present in test data point [True]
43 Text feature [3t3] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
47 Text feature [phosphatase] present in test data point [True]
49 Text feature [cell] present in test data point [True]
51 Text feature [growth] present in test data point [True]
53 Text feature [expression] present in test data point [True]
56 Text feature [patients] present in test data point [True]
57 Text feature [resistance] present in test data point [True]
59 Text feature [proteins] present in test data point [True]
61 Text feature [functions] present in test data point [True]
62 Text feature [inhibited] present in test data point [True]
67 Text feature [treated] present in test data point [True]
71 Text feature [predicted] present in test data point [True]
74 Text feature [amplification] present in test data point [True]
78 Text feature [oncogene] present in test data point [True]
79 Text feature [clinical] present in test data point [True]
80 Text feature [dna] present in test data point [True]
81 Text feature [inhibition] present in test data point [True]
85 Text feature [mek] present in test data point [True]
87 Text feature [affected] present in test data point [True]
88 Text feature [activity] present in test data point [True]
89 Text feature [use] present in test data point [True]
90 Text feature [potential] present in test data point [True]
91 Text feature [survival] present in test data point [True]
94 Text feature [proliferation] present in test data point [True]
96 Text feature [sensitivity] present in test data point [True]
97 Text feature [response] present in test data point [True]
98 Text feature [mapk] present in test data point [True]
99 Text feature [binding] present in test data point [True]
Out of the top  100  features  58 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [272]:
```
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: [[0.1151 0.1128 0.0214 0.1567 0.0526 0.0493 0.4787 0.0061 0.0072]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
3 Text feature [suppressor] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [constitutive] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [missense] present in test data point [True]
12 Text feature [activated] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
14 Text feature [receptor] present in test data point [True]
15 Text feature [stability] present in test data point [True]
16 Text feature [signaling] present in test data point [True]
18 Text feature [therapy] present in test data point [True]
19 Text feature [inhibitor] present in test data point [True]
22 Text feature [protein] present in test data point [True]
23 Text feature [cells] present in test data point [True]
26 Text feature [activate] present in test data point [True]
30 Text feature [therapeutic] present in test data point [True]
31 Text feature [constitutively] present in test data point [True]
40 Text feature [trials] present in test data point [True]
41 Text feature [drug] present in test data point [True]
42 Text feature [functional] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
49 Text feature [cell] present in test data point [True]
51 Text feature [growth] present in test data point [True]
53 Text feature [expression] present in test data point [True]
56 Text feature [patients] present in test data point [True]
57 Text feature [resistance] present in test data point [True]
58 Text feature [repair] present in test data point [True]
59 Text feature [proteins] present in test data point [True]
61 Text feature [functions] present in test data point [True]
62 Text feature [inhibited] present in test data point [True]
63 Text feature [ovarian] present in test data point [True]
65 Text feature [advanced] present in test data point [True]
67 Text feature [treated] present in test data point [True]
68 Text feature [null] present in test data point [True]
69 Text feature [p53] present in test data point [True]
71 Text feature [predicted] present in test data point [True]
74 Text feature [amplification] present in test data point [True]
76 Text feature [information] present in test data point [True]
78 Text feature [oncogene] present in test data point [True]
79 Text feature [clinical] present in test data point [True]
80 Text feature [dna] present in test data point [True]
81 Text feature [inhibition] present in test data point [True]
82 Text feature [efficacy] present in test data point [True]
86 Text feature [history] present in test data point [True]
87 Text feature [affected] present in test data point [True]
88 Text feature [activity] present in test data point [True]
90 Text feature [potential] present in test data point [True]
91 Text feature [survival] present in test data point [True]
92 Text feature [ring] present in test data point [True]
94 Text feature [proliferation] present in test data point [True]
96 Text feature [sensitivity] present in test data point [True]
97 Text feature [response] present in test data point [True]
99 Text feature [binding] present in test data point [True]
Out of the top  100  features  58 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [273]:
```
# # --------------------------------
# # default parameters
# # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# # class_weight=None)

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

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

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


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

# alpha = [10,50,100,200,500,1000]
# max_depth = [2,3,5,10]
# cv_log_error_array = []
# for i in alpha:
#     for j in max_depth:
#         print("for n_estimators =", i,"and max depth = ", j)
#         clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
#         clf.fit(train_x_responseCoding, train_y)
#         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
#         sig_clf.fit(train_x_responseCoding, train_y)
#         sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
#         cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
#         print("Log Loss :",log_loss(cv_y, sig_clf_probs))
# '''
# fig, ax = plt.subplots()
# features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
# ax.plot(features, cv_log_error_array,c='g')
# for i, txt in enumerate(np.round(cv_log_error_array,3)):
#     ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_array[i]))
# plt.grid()
# plt.title("Cross Validation Error for each alpha")
# plt.xlabel("Alpha i's")
# plt.ylabel("Error measure")
# plt.show()
# '''

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

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

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

In [274]:
```
# # --------------------------------
# # default parameters
# # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# # class_weight=None)

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

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

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

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

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [275]:
```
# 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].reshape(1,-1))
# print("Predicted Class :", predicted_cls[0])
# print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
# print("Actual Class :", test_y[test_point_index])
# indices = np.argsort(-clf.feature_importances_)
# print("-"*50)
# for i in indices:
#     if i<9:
#         print("Gene is important feature")
#     elif i<18:
#         print("Variation is important feature")
#     else:
#         print("Text is important feature")
```

4.5.5.2. Incorrectly Classified point

In [276]:
```
# test_point_index = 100
# predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
# print("Predicted Class :", predicted_cls[0])
# print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
# print("Actual Class :", test_y[test_point_index])
# indices = np.argsort(-clf.feature_importances_)
# print("-"*50)
# for i in indices:
#     if i<9:
#         print("Gene is important feature")
#     elif i<18:
#         print("Variation is important feature")
#     else:
#         print("Text is important feature")
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

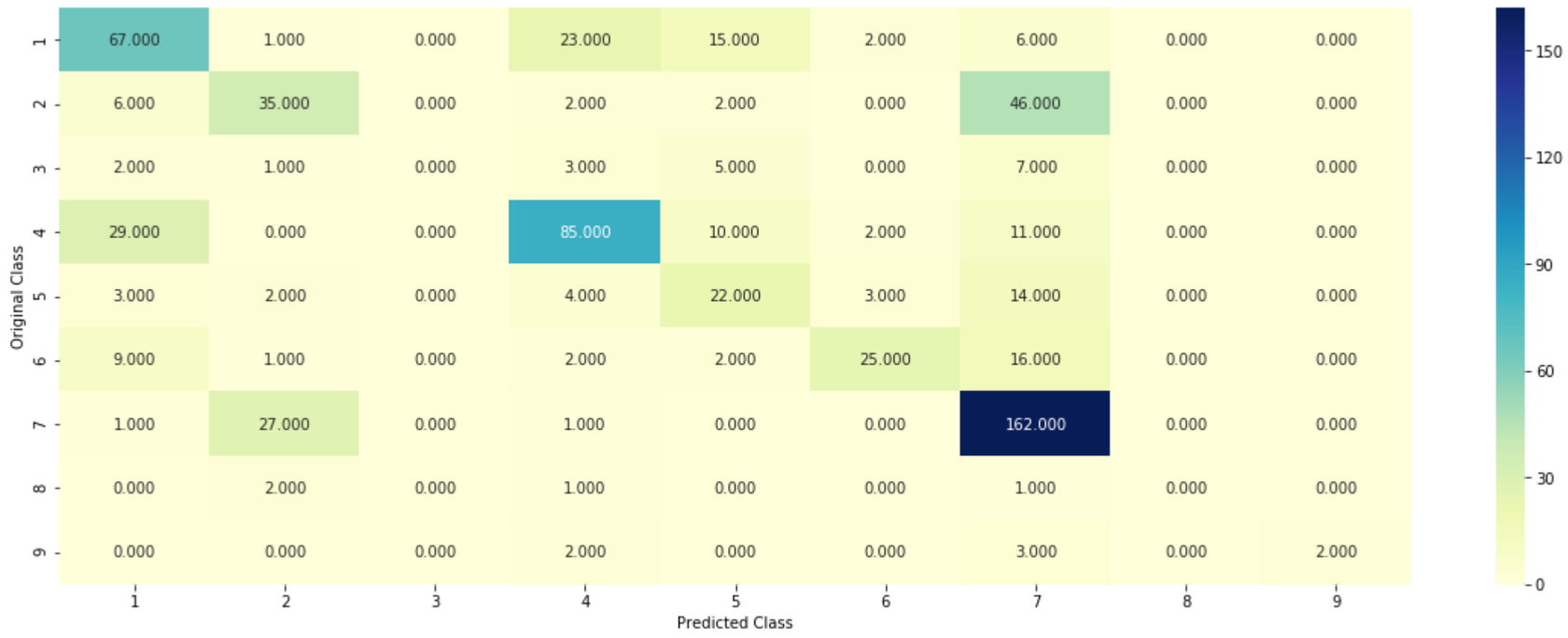
In [277]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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


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

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

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

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


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

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


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

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

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.91
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.523
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.134
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.145
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.245
```

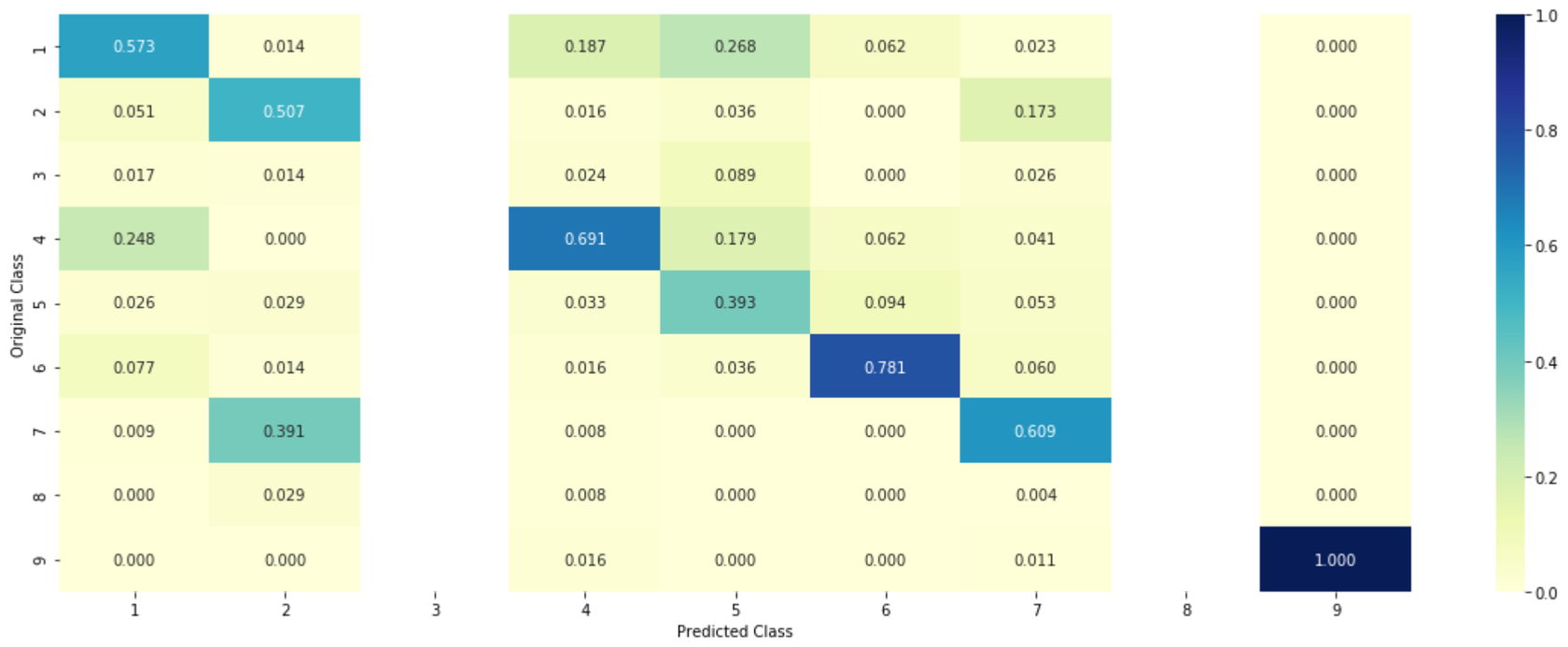
4.7.2 testing the model with the best hyper parameters

In [278]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

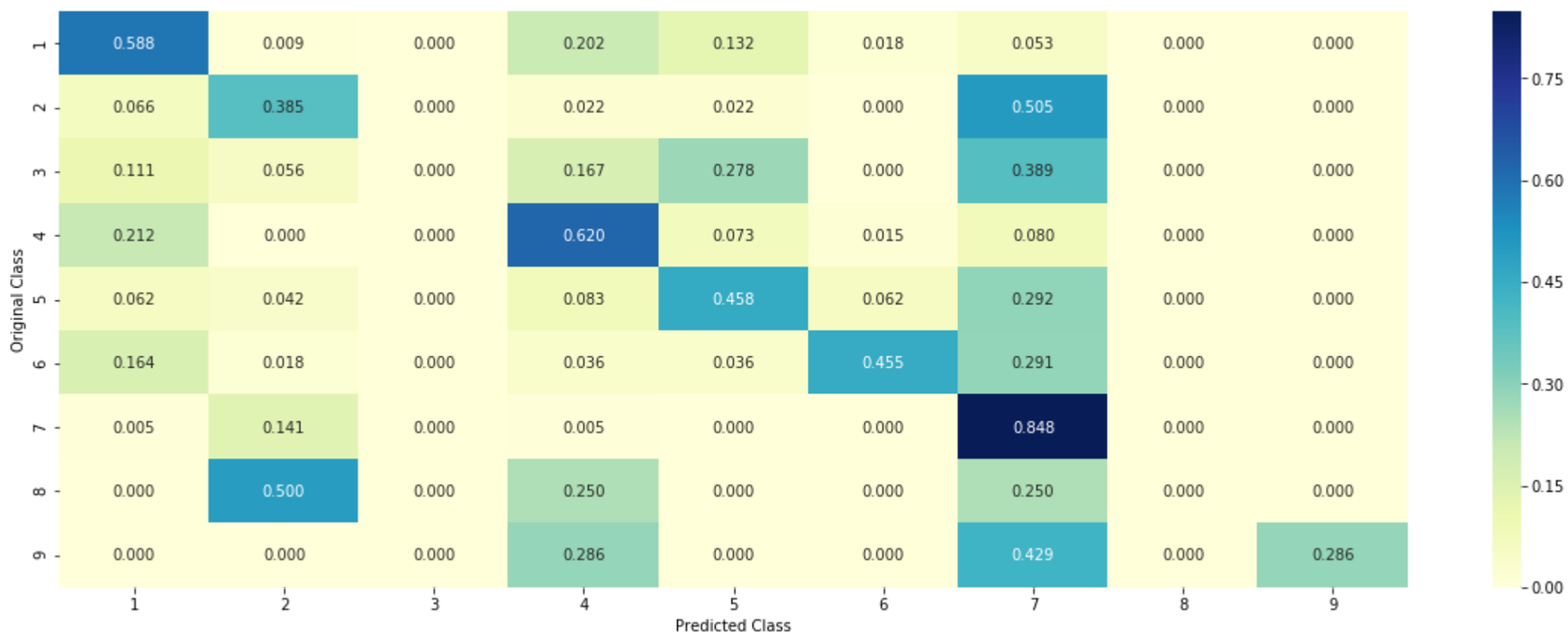
```
Log loss (train) on the stacking classifier : 0.8239533313876906
Log loss (CV) on the stacking classifier : 1.1335162683235565
Log loss (test) on the stacking classifier : 1.1615189888114121
Number of missclassified point : 0.40150375939849625
-------------------- Confusion matrix --------------------
```




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



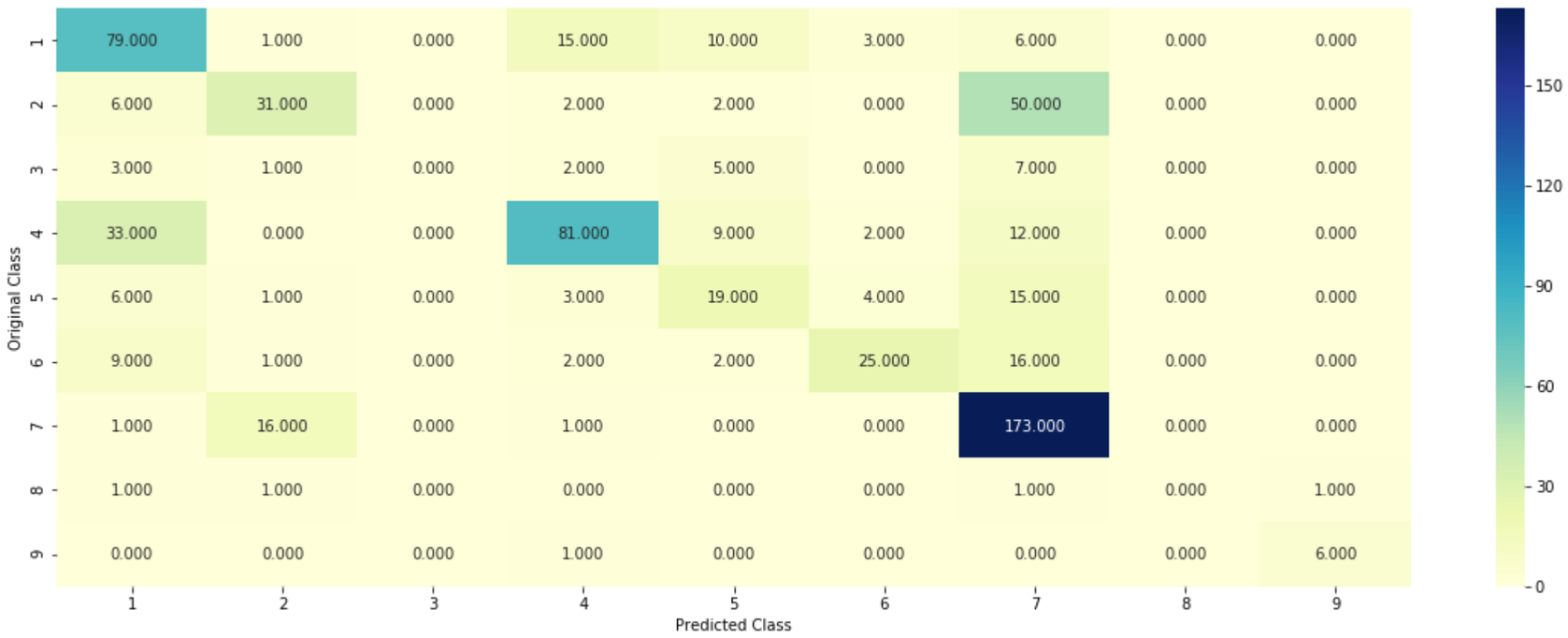

```
-------------------- Recall matrix (Row sum=1) --------------------
```



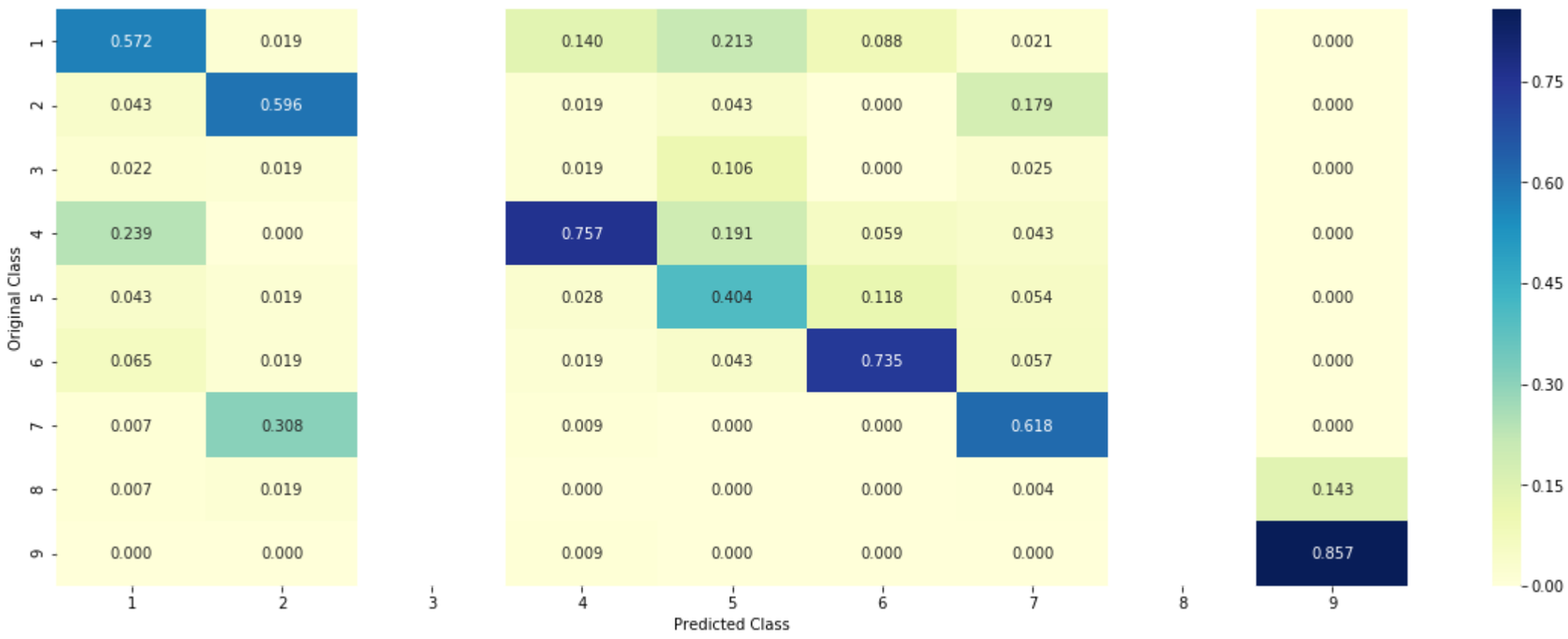
4.7.3 Maximum Voting classifier

In [279]:
```
#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)

max_log_error_train=log_loss(train_y, vclf.predict_proba(train_x_onehotCoding))
max_log_error_cv=log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
max_log_error_test=log_loss(test_y, vclf.predict_proba(test_x_onehotCoding))

print("Log loss (train) on the VotingClassifier :",max_log_error_train)
print("Log loss (CV) on the VotingClassifier :",max_log_error_train)
print("Log loss (test) on the VotingClassifier :",max_log_error_train)
maxmp=np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0]
print("Number of missclassified point :", maxmp)
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

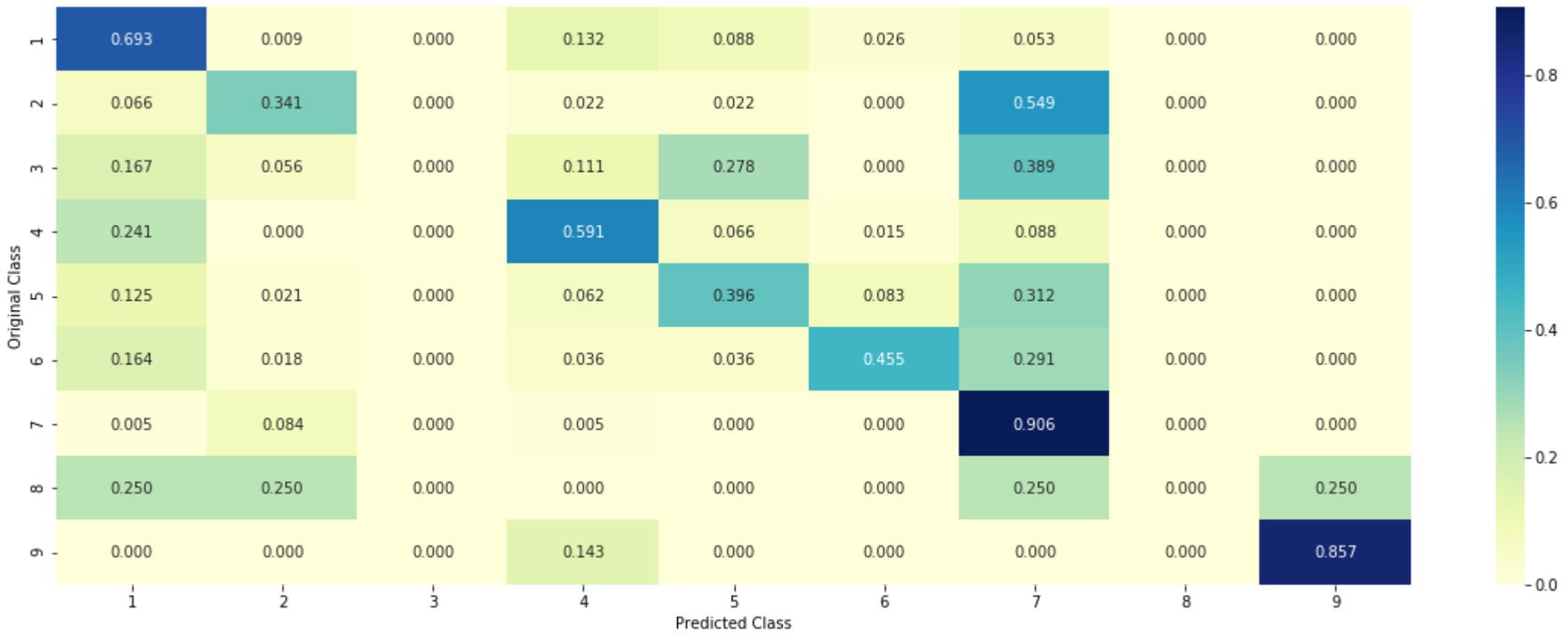
```
Log loss (train) on the VotingClassifier : 0.9633631291524695
Log loss (CV) on the VotingClassifier : 0.9633631291524695
Log loss (test) on the VotingClassifier : 0.9633631291524695
Number of missclassified point : 0.3774436090225564
-------------------- Confusion matrix --------------------
```




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




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



5. Assignments

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

5. APPLYING FEATURE ENGG -> ONE HOT on GENE & VARIATION and TF-IDF - BIGRAM (Top 20000) on TEXT

GENE - ONE HOT CODING

In [280]:
```
# one-hot encoding of Gene feature.
gene_vectorizer_fe = CountVectorizer()
train_gene_feature_onehotCoding_fe = gene_vectorizer_fe.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_fe = gene_vectorizer_fe.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_fe = gene_vectorizer_fe.transform(cv_df['Gene'])
print('train_gene_feature_onehotCoding',train_gene_feature_onehotCoding_fe.shape)
```

```
train_gene_feature_onehotCoding (2124, 229)
```

VARIATION - ONE HOT CODING

In [281]:
```
### one-hot encoding of variation feature.
variation_vectorizer_fe = CountVectorizer()
train_variation_feature_onehotCoding_fe = variation_vectorizer_fe.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_fe = variation_vectorizer_fe.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_fe = variation_vectorizer_fe.transform(cv_df['Variation'])
```

TEXT - TFIDF (TOP 20000) CODING

In [282]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
## text_vectorizer = CountVectorizer(min_df=3)
text_vectorizer_fe = TfidfVectorizer(ngram_range=(2,2),min_df=3,max_features=20000)
train_text_feature_onehotCoding_fe = text_vectorizer_fe.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer_fe.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
train_text_fea_counts = train_text_feature_onehotCoding_fe.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 : 20000
```

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

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

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

In [284]:
```
## for count vectorizer
train_x_onehotCoding_count_fe = hstack((train_gene_feature_onehotCoding_fe,train_variation_feature_onehotCoding_fe, train_text_feature_onehotCoding_fe)).tocsr()
test_x_onehotCoding_count_fe = hstack((test_gene_feature_onehotCoding_fe,test_variation_feature_onehotCoding_fe, test_text_feature_onehotCoding_fe)).tocsr()
cv_x_onehotCoding_count_fe = hstack((cv_gene_feature_onehotCoding_fe,cv_variation_feature_onehotCoding_fe, cv_text_feature_onehotCoding_fe)).tocsr()
```

In [285]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding_count_fe.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding_count_fe.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding_count_fe.shape)
```

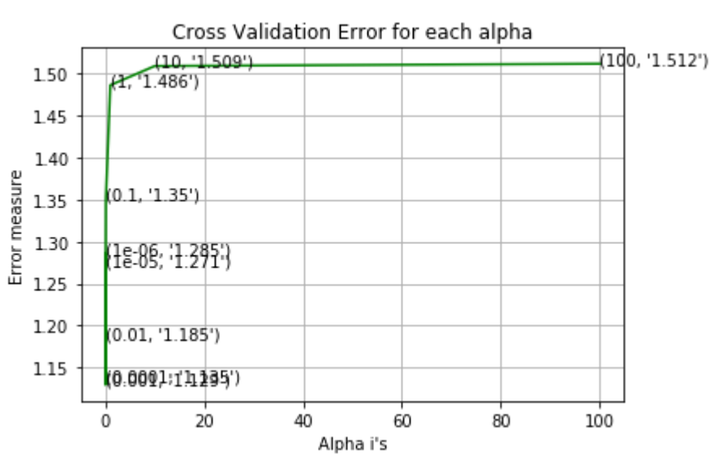
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 22183)
(number of data points * number of features) in test data =  (665, 22183)
(number of data points * number of features) in cross validation data = (532, 22183)
```

With Class balancing

Hyper paramter tuning

In [286]:
```
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_count_fe, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_count_fe, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_count_fe)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.2849979023287197
for alpha = 1e-05
Log Loss : 1.2712414218818369
for alpha = 0.0001
Log Loss : 1.1346454367296734
for alpha = 0.001
Log Loss : 1.1294924015284218
for alpha = 0.01
Log Loss : 1.1849629766515126
for alpha = 0.1
Log Loss : 1.350325952443423
for alpha = 1
Log Loss : 1.4864088418630865
for alpha = 10
Log Loss : 1.5092245753406153
for alpha = 100
Log Loss : 1.5119242455239816
```




```
For values of best alpha =  0.001 The train log loss is: 0.5289714669249315
For values of best alpha =  0.001 The cross validation log loss is: 1.1294924015284218
For values of best alpha =  0.001 The test log loss is: 1.0532609109841884
```

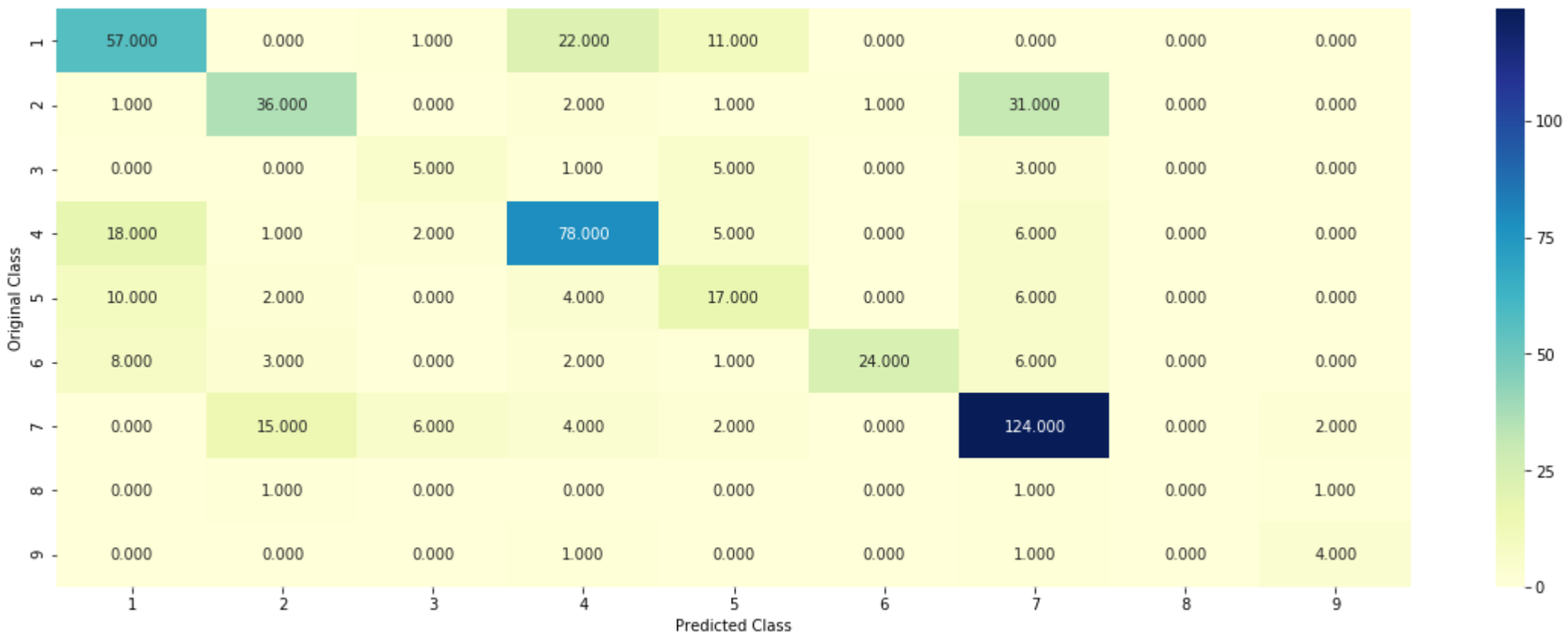
In [287]:
```
print("lrLossClassBalance_tfidf_train",lrLossClassBalance_tfidf_train_fe)
print("lrLossClassBalance_tfidf_cv",lrLossClassBalance_tfidf_cv_fe)
print("lrLossClassBalance_tfidf_test",lrLossClassBalance_tfidf_test_fe)
```

```
lrLossClassBalance_tfidf_train 0.5289714669249315
lrLossClassBalance_tfidf_cv 1.1294924015284218
lrLossClassBalance_tfidf_test 1.0532609109841884
```

4.3.1.2. Testing the model with best hyper paramters

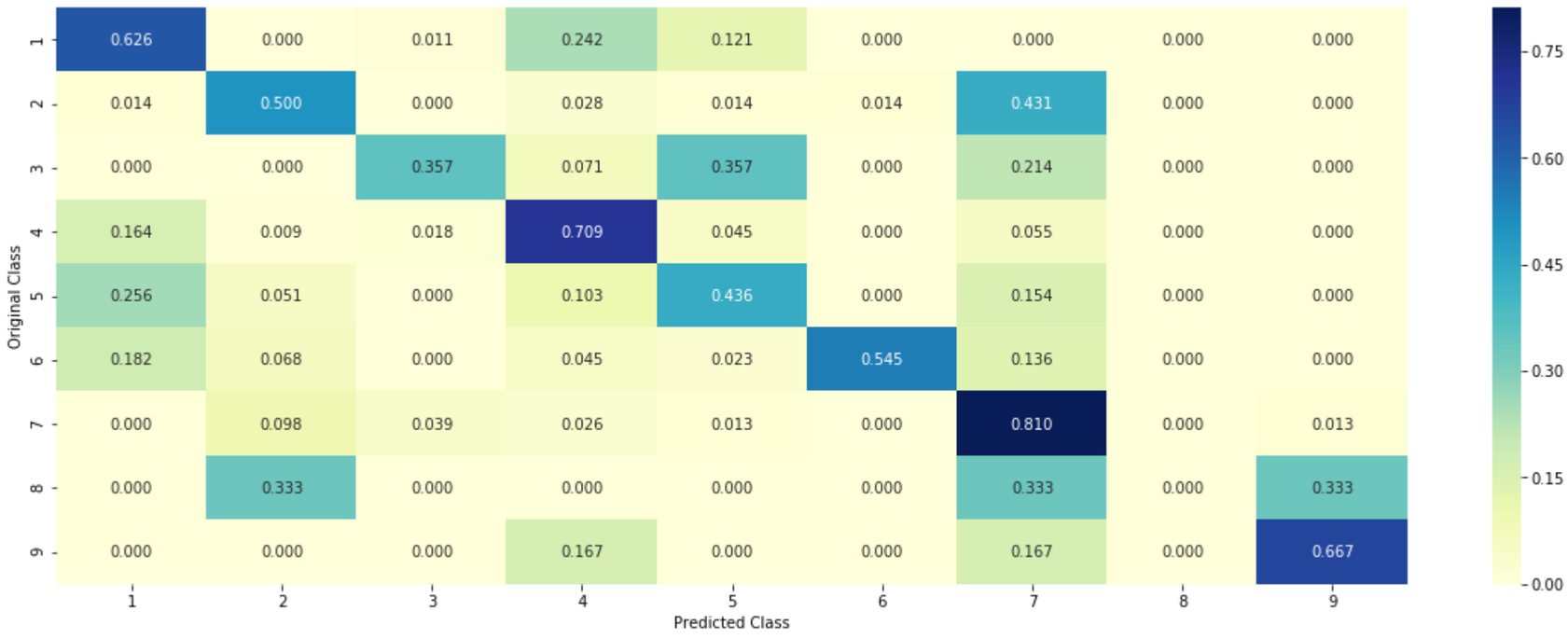
In [288]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
lrClassBalancemp_fe=predict_and_plot_confusion_matrix(train_x_onehotCoding_count_fe, train_y, cv_x_onehotCoding_count_fe, cv_y, clf)
```

```
Log loss : 1.1294924015284218
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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


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



In [290]:
```
from prettytable import PrettyTable
# Names of models
model=['Naive Bayes ','KNN','Logistic Regression With Class balancing '
       ,'LogisticRegression Without Class balancing','Linear SVM '
       ,'Random Forest Classifier With One hot Encoding'
       ,'Stack Models:LR+NB+SVM','Maximum Voting classifier'
       ,'LR(BALANCED): CountVectorizer Features, including both unigrams and bigrams','LR(UNBALANCED): CountVectorizer Features, including both unigrams and bigrams'
       ,'LR: after feature engineering']
train =[
    nbLoss_train,
    knnLoss_train,
    lrLossClassBalance_tfidf_train,
    lrLossWithoutClassBalance_tfidf_train,
    svmLoss_train,
    rfLoss_train,
    stack_log_error_train,
    max_log_error_train,
    lrLossClassBalance_count_train,
    lrLossWithoutClassBalance_count_train,
    lrLossClassBalance_tfidf_train_fe]
cv=[
    nbLoss_cv,
    knnLoss_cv,
    lrLossClassBalance_tfidf_cv,
    lrLossWithoutClassBalance_tfidf_cv,
    svmLoss_cv,
    rfLoss_cv,
    stack_log_error_cv,
    max_log_error_cv,
    lrLossClassBalance_count_cv,
    lrLossWithoutClassBalance_count_cv,
    lrLossClassBalance_tfidf_cv_fe
]
test = [
    nbLoss_test,
    knnLoss_test,
    lrLossClassBalance_tfidf_test,
    lrLossWithoutClassBalance_tfidf_test,
    svmLoss_test,
    rfLoss_test,
    stack_log_error_test,
    max_log_error_test,
    lrLossClassBalance_count_test,
    lrLossWithoutClassBalance_count_test,
    lrLossClassBalance_tfidf_test_fe,
]

mp=[
    nbmp,
    knnmp,
    lrClassBalancemp,
    lrWithoutClassBalancemp,
    svmmp,
    rfmp,
    stackmp,
    maxmp,
    lrClassBalanceCountmp,
    lrWithoutClassBalanceCountmp,
    lrClassBalancemp_fe,
   ]


train=[round(x,2) for x in train]
cv=[round(x,2) for x in cv]
test=[round(x,2) for x in test]
mp=[round(x,2) for x in mp]
numbering=[1,2,3,4,5,6,7,8,9,10,11]
# Initializing prettytable
ptable = PrettyTable()
# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("model",model)
ptable.add_column("train",train)
ptable.add_column("cv",cv)
ptable.add_column("test",test)
ptable.add_column("% Missclassified Points",mp)
# Printing the Table
print(ptable)
```

```
+-------+---------------------------------------------------------------------------+-------+------+------+------------------------+
| S.NO. |                                   model                                   | train |  cv  | test | % Misclassified Points |
+-------+---------------------------------------------------------------------------+-------+------+------+------------------------+
|   1   |                                Naive Bayes                                |  0.81 | 1.19 | 1.21 |          0.37          |
|   2   |                                    KNN                                    |  0.95 | 1.17 | 1.18 |          0.37          |
|   3   |                  Logistic Regression With Class balancing                 |  0.59 | 1.03 | 1.02 |          0.34          |
|   4   |                  LogisticRegression Without Class balancing               |  0.58 | 1.06 | 1.04 |          0.33          |
|   5   |                               Linear SVM                                  |  0.82 | 1.12 | 1.14 |          0.36          |
|   6   |               Random Forest Classifier With One hot Encoding              |  0.85 | 1.19 | 1.21 |          0.42          |
|   7   |                          Stack Models:LR+NB+SVM                           |  0.82 | 1.13 | 1.16 |          0.40          |
|   8   |                        Maximum Voting classifier                          |  0.96 | 1.21 | 1.21 |          0.37          |
|   9   |  LR(BALANCED): CountVectorizer Features, including both unigrams and bigrams  |  0.75 | 1.22 | 1.18 |          0.37          |
|   10  | LR(UNBALANCED): CountVectorizer Features, including both unigrams and bigrams |  0.72 | 1.22 | 1.17 |          0.39          |
|   11  |                     LR: after feature engineering                         |  0.53 | 1.13 | 1.05 |          0.35          |
+-------+---------------------------------------------------------------------------+-------+------+------+------------------------+
```

CONCLUSION

1. Thus we can see that Logistic Regression is the best compared to all the models.
2. Our model with BOW on GENE & VARIATION and with TFIDF (TOP 20000 BIGRAM) performs the second best with LogLoss of TEST and CV very very close to 1.