Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

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

from mlxtend.classifier import StackingClassifier

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

```
C:\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWar
ning: The sklearn.metrics.classification module is  deprecated in version
0.22 and will be removed in version 0.24. The corresponding classes / func
tions should instead be imported from sklearn.metrics. Anything that canno
t be imported from sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
C:\Anaconda3\lib\site-packages\sklearn\externals\six.py:31: FutureWarning:
The module is deprecated in version 0.21 and will be removed in version 0.
23 since we've dropped support for Python 2.7. Please rely on the official
version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
C:\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWar
ning: The sklearn.neighbors.base module is  deprecated in version 0.22 and
will be removed in version 0.24. The corresponding classes / functions sho
uld instead be imported from sklearn.neighbors. Anything that cannot be im
ported from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
C:\Anaconda3\lib\site-packages\sklearn\externals\joblib\__init__.py:15: Fu
tureWarning: sklearn.externals.joblib is deprecated in 0.21 and will be re
moved in 0.23. Please import this functionality directly from joblib, whic
h can be installed with: pip install joblib. If this warning is raised whe
n loading pickled models, you may need to re-serialize those models with s
cikit-learn 0.21+.
  warnings.warn(msg, category=FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

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

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

```
# loading stop words from nltk library
import nltk
nltk.download('stopwords')
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
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Venki\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 67.5082789 seconds
```

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

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

In [8]:

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

In [9]:

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

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

In [11]:

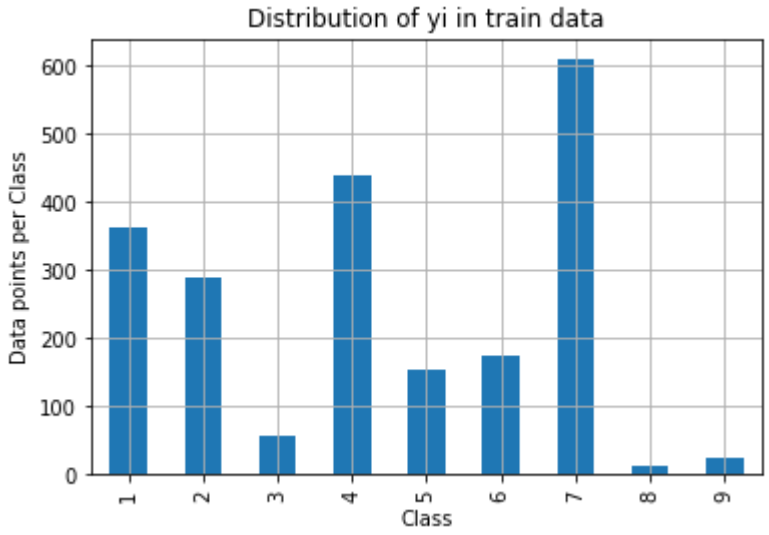
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

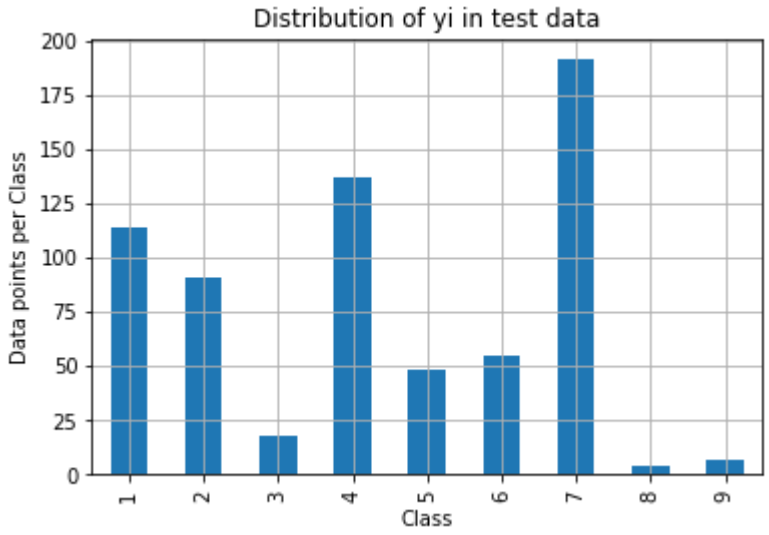
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:

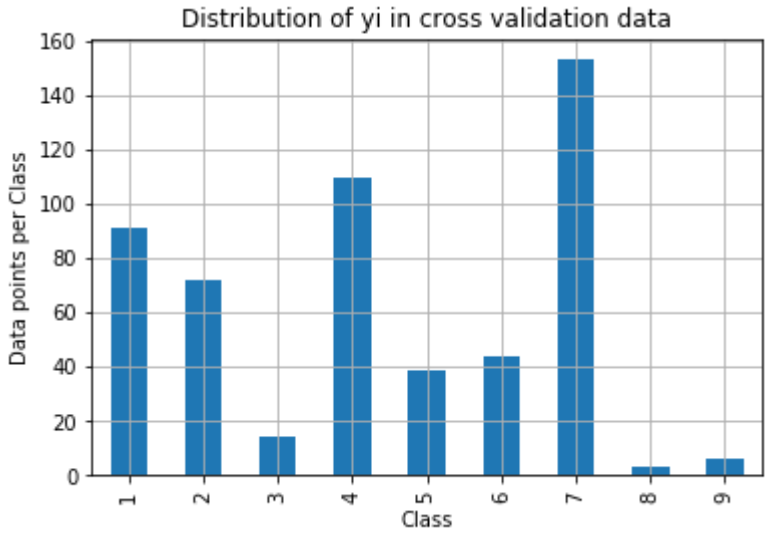
```
# it returns a dict, keys as class labels and values as the number of data points in th
at class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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.

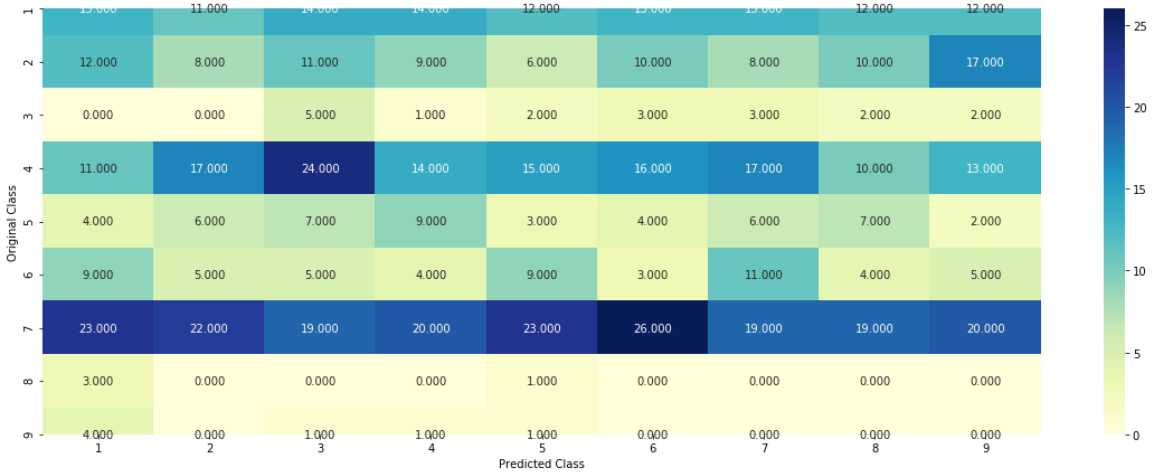
In [13]:

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

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

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

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

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

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

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

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

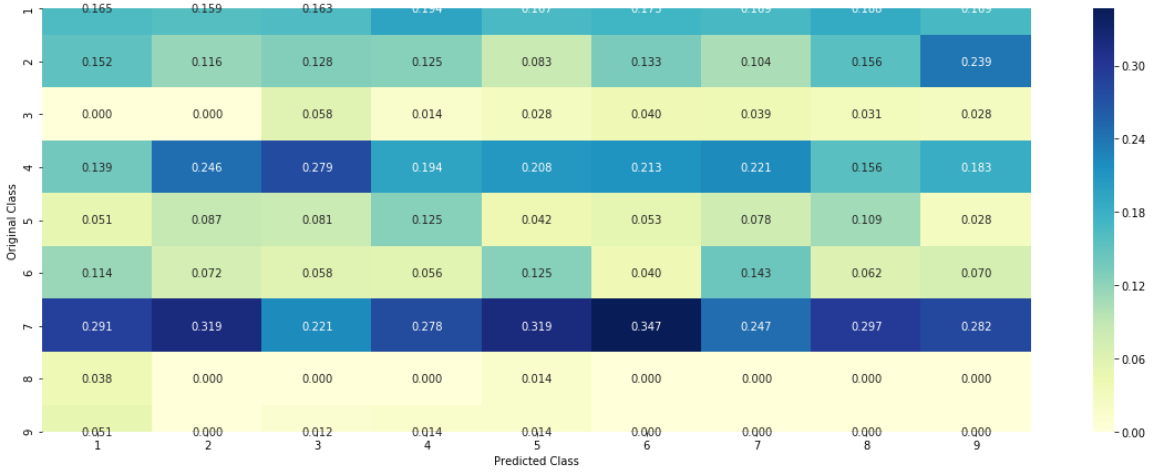
In [14]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by 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)
```

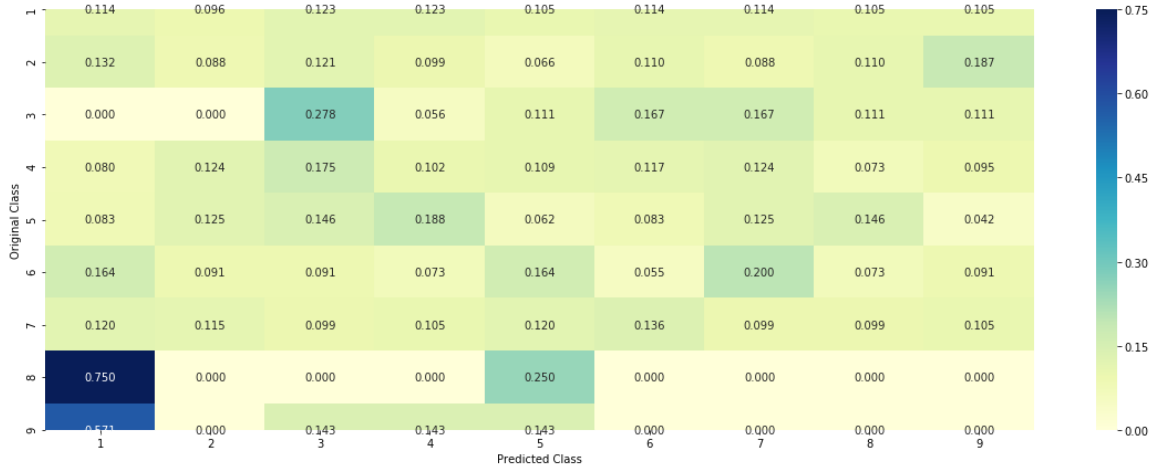
```
Log loss on Cross Validation Data using Random Model 2.3930675069040936
Log loss on Test Data using Random Model 2.4427245416367676
-------------------- Confusion matrix --------------------
```




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




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



3.3 Univariate Analysis

In [15]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train dat
a dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10
*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

    # denominator will contain the number of time that particular feature occured in wh
ole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pertic
ular class
```

```
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.
13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.
27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.
051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181
8177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917,
0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.0
72847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.0
66225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.
073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.
066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

```
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:

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

```
Number of Unique Genes : 229
BRCA1     175
TP53      112
EGFR       86
PTEN       86
BRCA2      85
BRAF       62
KIT        59
ERBB2      44
PDGFRA     41
PIK3CA     40
Name: Gene, dtype: int64
```

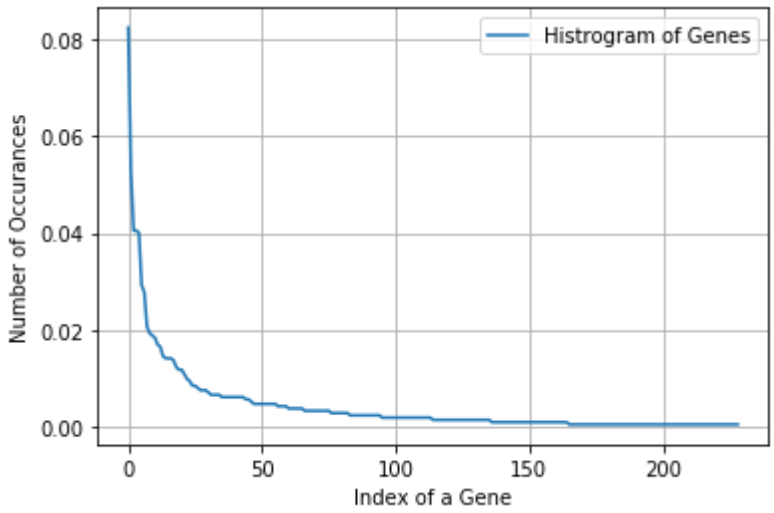
In [17]:

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

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

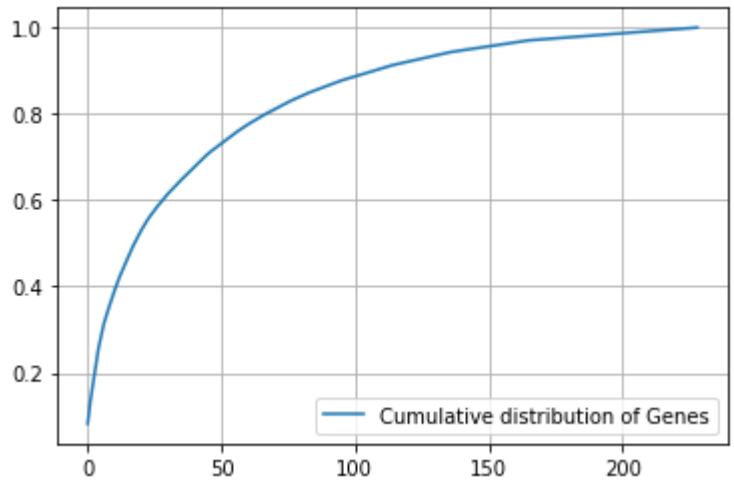
In [18]:

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



In [19]:

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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", 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 [21]:

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

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

In [22]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:

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

Out[23]:

```
1141       MET
651     CDKN2A
2824     BRCA2
775      ERBB3
2635     BRCA1
Name: Gene, dtype: object
```

In [24]:

```
gene_vectorizer.get_feature_names()
```

Out[24]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3b',
```

```
    'dusp4',
    'egfr',
    'eif1ax',
    'elf3',
    'ep300',
    'epas1',
    'erbb2',
    'erbb3',
    'erbb4',
    'ercc2',
    'ercc3',
    'ercc4',
    'erg',
    'errfi1',
    'esr1',
    'etv1',
    'etv6',
    'ewsr1',
    'ezh2',
    'fam58a',
    'fanca',
    'fat1',
    'fbxw7',
    'fgf3',
    'fgf4',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'fgfr4',
    'flt1',
    'flt3',
    'foxa1',
    'foxl2',
    'foxo1',
    'foxp1',
    'gata3',
    'gna11',
    'gnas',
    'h3f3a',
    'hnf1a',
    'hras',
    'idh1',
    'idh2',
    'ikbke',
    'ikzf1',
    'jak1',
    'jak2',
    'jun',
    'kdm5a',
    'kdm5c',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'kmt2a',
    'kmt2b',
    'kmt2c',
    'kmt2d',
    'knstrn',
    'kras',
    'lats2',
```

```
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mapk1',
    'mdm2',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mga',
    'mlh1',
    'mpl',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myd88',
    'myod1',
    'nf1',
    'nf2',
    'nfe2l2',
    'nfkbia',
    'nkx2',
    'notch1',
    'notch2',
    'npm1',
    'nras',
    'nsd1',
    'ntrk1',
    'ntrk2',
    'ntrk3',
    'pak1',
    'pbrm1',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
    'pik3r3',
    'pim1',
    'pms1',
    'pms2',
    'pole',
    'ppm1d',
    'ppp2r1a',
    'ppp6c',
    'prdm1',
    'ptch1',
    'pten',
    'ptpn11',
    'ptprd',
    'ptprt',
    'rab35',
    'rac1',
    'rad21',
    'rad50',
    'rad51c',
```

```
 'rad51d',
 'rad54l',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:

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

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

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

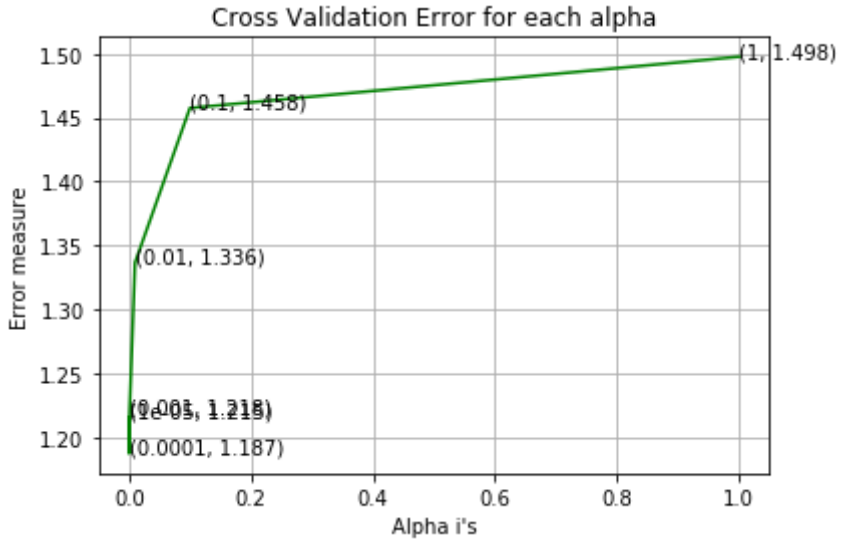
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/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_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2149854308882255
For values of alpha =  0.0001 The log loss is: 1.187051844068848
For values of alpha =  0.001 The log loss is: 1.2183894180015384
For values of alpha =  0.01 The log loss is: 1.3364204135330024
For values of alpha =  0.1 The log loss is: 1.4575403512917326
For values of alpha =  1 The log loss is: 1.4977201424493225
```




```
For values of best alpha =  0.0001 The train log loss is: 0.98684113575890
78
For values of best alpha =  0.0001 The cross validation log loss is: 1.187
051844068848
For values of best alpha =  0.0001 The test log loss is: 1.194397918059185
```

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

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

In [27]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  229
genes in train dataset?
Ans
1. In test data 634 out of 665 : 95.33834586466166
2. In cross validation data 519 out of  532 : 97.55639097744361
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [28]:

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

```
Number of Unique Variations : 1914
Truncating_Mutations    61
Amplification           54
Deletion                49
Fusions                 28
Overexpression           5
G12V                     3
Q61R                     3
P34R                     2
R841K                    2
Q61K                     2
Name: Variation, dtype: int64
```

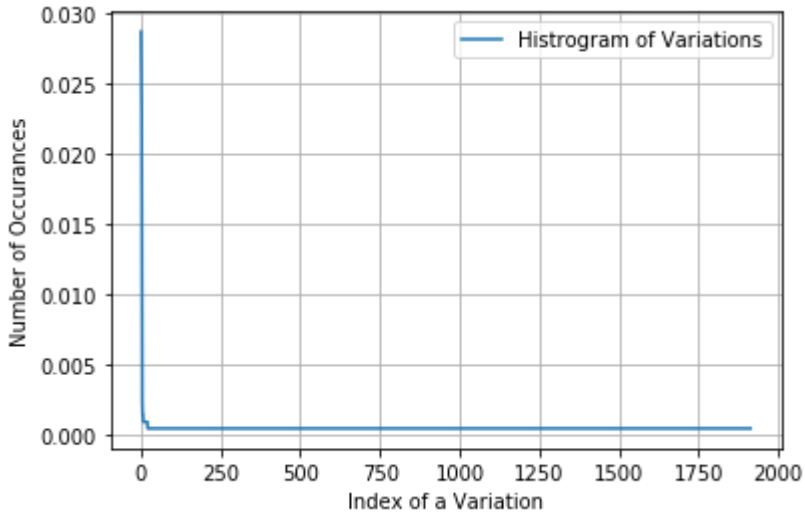
In [29]:

```
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 1914 different categories of variations in the train data,
and they are distibuted as follows
```

In [30]:

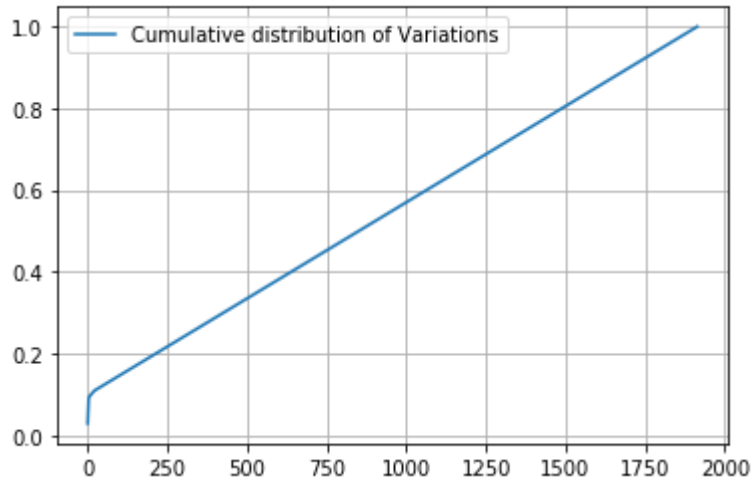
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:

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

```
[0.0287194  0.05414313 0.07721281 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:

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

In [33]:

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

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

In [34]:

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

In [35]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-
hot encoding method. The shape of Variation feature: (2124, 1941)
```

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

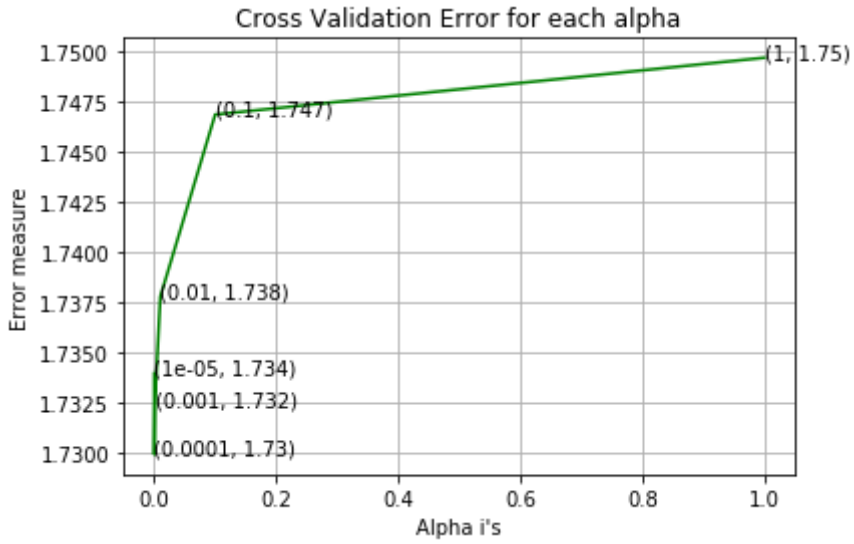
Let's build a model just like the earlier!

In [36]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/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_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7339675642983863
For values of alpha =  0.0001 The log loss is: 1.729926996338199
For values of alpha =  0.001 The log loss is: 1.7323947457805056
For values of alpha =  0.01 The log loss is: 1.737789306561023
For values of alpha =  0.1 The log loss is: 1.746859883848056
For values of alpha =  1 The log loss is: 1.7496890344200948
```




```
For values of best alpha =  0.0001 The train log loss is: 0.68868449206986
87
For values of best alpha =  0.0001 The cross validation log loss is: 1.729
926996338199
For values of best alpha =  0.0001 The test log loss is: 1.717970603825089
```

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

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

In [37]:

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

```
Q12. How many data points are covered by total  1914  genes in test and cr
oss validation data sets?
Ans
1. In test data 66 out of 665 : 9.924812030075188
2. In cross validation data 40 out of  532 : 7.518796992481203
```

3.2.3 Univariate Analysis on Text Feature

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

In [38]:

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

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

In [39]:

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

In [40]:

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

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

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


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

```
Total number of unique words in train data : 53909
```

In [41]:

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

In [42]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [43]:

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

In [44]:

```
# 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 [45]:

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

In [46]:

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

```
Counter({3: 5566, 4: 3754, 6: 3035, 5: 2989, 7: 2208, 9: 1758, 8: 1737, 1
2: 1545, 10: 1371, 11: 1306, 13: 991, 16: 831, 14: 830, 15: 779, 18: 735,
20: 609, 17: 602, 22: 573, 24: 515, 21: 489, 19: 461, 26: 412, 28: 390, 4
5: 388, 23: 355, 25: 352, 27: 348, 32: 337, 30: 333, 48: 318, 29: 309, 36:
272, 33: 258, 35: 251, 34: 248, 40: 238, 31: 228, 39: 194, 44: 192, 38: 18
9, 37: 185, 46: 183, 42: 181, 41: 178, 50: 175, 47: 173, 60: 170, 54: 163,
49: 155, 51: 150, 58: 147, 55: 142, 43: 141, 57: 132, 52: 130, 72: 128, 5
6: 121, 63: 120, 53: 115, 64: 106, 66: 105, 73: 103, 59: 102, 65: 101, 70:
100, 61: 100, 69: 99, 68: 97, 62: 97, 78: 94, 77: 94, 90: 93, 74: 90, 76:
84, 67: 83, 87: 80, 71: 79, 91: 76, 75: 74, 92: 73, 84: 73, 81: 72, 96: 7
0, 88: 69, 82: 69, 93: 67, 108: 66, 99: 66, 80: 66, 97: 64, 95: 64, 98: 6
3, 94: 62, 85: 61, 104: 59, 102: 58, 79: 58, 86: 53, 83: 53, 117: 52, 105:
52, 101: 52, 135: 50, 144: 49, 103: 48, 89: 48, 130: 46, 120: 46, 111: 45,
106: 45, 100: 45, 142: 44, 116: 44, 112: 44, 125: 43, 115: 43, 138: 42, 13
2: 42, 124: 42, 110: 42, 121: 41, 114: 41, 149: 39, 127: 38, 147: 37, 141:
37, 131: 37, 170: 36, 119: 36, 113: 36, 153: 35, 140: 35, 133: 35, 123: 3
5, 122: 35, 128: 34, 109: 34, 154: 33, 134: 33, 129: 33, 137: 32, 136: 32,
107: 32, 160: 31, 150: 31, 139: 31, 126: 31, 118: 31, 159: 30, 157: 30, 16
8: 29, 176: 28, 173: 28, 169: 28, 155: 28, 151: 27, 228: 26, 201: 26, 200:
26, 158: 26, 152: 26, 195: 25, 180: 25, 179: 25, 166: 25, 165: 25, 156: 2
5, 145: 25, 143: 25, 302: 24, 193: 24, 167: 24, 234: 23, 212: 23, 210: 23,
198: 23, 196: 23, 192: 23, 177: 23, 163: 23, 148: 23, 213: 22, 209: 22, 20
2: 22, 146: 22, 215: 21, 205: 21, 204: 21, 191: 21, 189: 21, 185: 21, 181:
21, 172: 21, 162: 21, 270: 20, 232: 20, 225: 20, 186: 20, 178: 20, 175: 2
0, 164: 20, 161: 20, 235: 19, 227: 19, 226: 19, 224: 19, 222: 19, 211: 19,
183: 19, 260: 18, 243: 18, 240: 18, 236: 18, 230: 18, 217: 18, 199: 18, 18
8: 18, 184: 18, 174: 18, 171: 18, 257: 17, 245: 17, 241: 17, 229: 17, 223:
17, 208: 17, 206: 17, 203: 17, 295: 16, 283: 16, 267: 16, 259: 16, 238: 1
6, 221: 16, 214: 16, 194: 16, 366: 15, 324: 15, 317: 15, 299: 15, 290: 15,
286: 15, 255: 15, 254: 15, 253: 15, 246: 15, 242: 15, 239: 15, 220: 15, 20
7: 15, 190: 15, 333: 14, 322: 14, 272: 14, 268: 14, 266: 14, 261: 14, 256:
14, 218: 14, 216: 14, 197: 14, 182: 14, 453: 13, 403: 13, 334: 13, 275: 1
3, 273: 13, 269: 13, 263: 13, 258: 13, 252: 13, 495: 12, 384: 12, 382: 12,
361: 12, 360: 12, 335: 12, 315: 12, 314: 12, 297: 12, 288: 12, 279: 12, 27
8: 12, 271: 12, 264: 12, 435: 11, 365: 11, 350: 11, 345: 11, 338: 11, 328:
11, 321: 11, 313: 11, 306: 11, 300: 11, 296: 11, 289: 11, 285: 11, 277: 1
1, 274: 11, 251: 11, 231: 11, 219: 11, 187: 11, 543: 10, 450: 10, 422: 10,
418: 10, 417: 10, 410: 10, 389: 10, 378: 10, 373: 10, 368: 10, 357: 10, 35
6: 10, 344: 10, 343: 10, 332: 10, 320: 10, 319: 10, 305: 10, 291: 10, 284:
10, 282: 10, 280: 10, 250: 10, 248: 10, 247: 10, 587: 9, 522: 9, 514: 9, 5
12: 9, 509: 9, 440: 9, 409: 9, 408: 9, 394: 9, 374: 9, 372: 9, 339: 9, 32
6: 9, 325: 9, 309: 9, 308: 9, 301: 9, 298: 9, 294: 9, 293: 9, 292: 9, 287:
9, 265: 9, 262: 9, 237: 9, 927: 8, 787: 8, 658: 8, 589: 8, 584: 8, 554: 8,
540: 8, 525: 8, 521: 8, 478: 8, 461: 8, 458: 8, 447: 8, 426: 8, 397: 8, 38
8: 8, 371: 8, 354: 8, 353: 8, 340: 8, 310: 8, 307: 8, 304: 8, 281: 8, 249:
8, 244: 8, 233: 8, 811: 7, 716: 7, 702: 7, 624: 7, 623: 7, 579: 7, 571: 7,
560: 7, 559: 7, 538: 7, 528: 7, 508: 7, 473: 7, 463: 7, 460: 7, 456: 7, 44
8: 7, 445: 7, 432: 7, 406: 7, 404: 7, 398: 7, 392: 7, 383: 7, 375: 7, 362:
7, 359: 7, 348: 7, 342: 7, 337: 7, 336: 7, 323: 7, 318: 7, 316: 7, 311: 7,
1274: 6, 909: 6, 871: 6, 844: 6, 817: 6, 786: 6, 780: 6, 681: 6, 660: 6, 6
55: 6, 653: 6, 632: 6, 626: 6, 573: 6, 572: 6, 550: 6, 545: 6, 539: 6, 52
6: 6, 524: 6, 516: 6, 510: 6, 503: 6, 502: 6, 500: 6, 497: 6, 494: 6, 487:
6, 484: 6, 474: 6, 471: 6, 468: 6, 466: 6, 457: 6, 454: 6, 449: 6, 446: 6,
442: 6, 441: 6, 437: 6, 433: 6, 431: 6, 429: 6, 425: 6, 416: 6, 413: 6, 40
2: 6, 391: 6, 380: 6, 376: 6, 364: 6, 352: 6, 349: 6, 347: 6, 346: 6, 341:
6, 330: 6, 327: 6, 303: 6, 1224: 5, 1114: 5, 937: 5, 853: 5, 799: 5, 781:
5, 776: 5, 774: 5, 769: 5, 760: 5, 748: 5, 747: 5, 740: 5, 705: 5, 651: 5,
650: 5, 643: 5, 619: 5, 618: 5, 613: 5, 604: 5, 581: 5, 578: 5, 553: 5, 54
8: 5, 534: 5, 531: 5, 511: 5, 507: 5, 492: 5, 480: 5, 477: 5, 475: 5, 470:
5, 464: 5, 462: 5, 452: 5, 443: 5, 428: 5, 415: 5, 414: 5, 412: 5, 405: 5,
401: 5, 396: 5, 395: 5, 390: 5, 387: 5, 386: 5, 381: 5, 379: 5, 370: 5, 36
```

```
9: 5, 358: 5, 355: 5, 331: 5, 312: 5, 276: 5, 2115: 4, 1926: 4, 1495: 4, 1
355: 4, 1319: 4, 1263: 4, 1261: 4, 1244: 4, 1234: 4, 1233: 4, 1180: 4, 113
3: 4, 1046: 4, 1036: 4, 1026: 4, 1009: 4, 1001: 4, 993: 4, 981: 4, 972: 4,
964: 4, 952: 4, 945: 4, 941: 4, 936: 4, 918: 4, 904: 4, 902: 4, 880: 4, 87
7: 4, 829: 4, 824: 4, 822: 4, 815: 4, 814: 4, 809: 4, 808: 4, 807: 4, 800:
4, 789: 4, 779: 4, 772: 4, 768: 4, 767: 4, 761: 4, 750: 4, 742: 4, 741: 4,
734: 4, 729: 4, 726: 4, 713: 4, 711: 4, 710: 4, 699: 4, 696: 4, 695: 4, 69
1: 4, 690: 4, 680: 4, 677: 4, 676: 4, 673: 4, 662: 4, 659: 4, 654: 4, 649:
4, 647: 4, 644: 4, 642: 4, 635: 4, 633: 4, 628: 4, 625: 4, 610: 4, 598: 4,
594: 4, 592: 4, 591: 4, 583: 4, 580: 4, 570: 4, 568: 4, 567: 4, 562: 4, 56
1: 4, 558: 4, 557: 4, 544: 4, 542: 4, 535: 4, 532: 4, 520: 4, 517: 4, 505:
4, 504: 4, 501: 4, 496: 4, 489: 4, 488: 4, 483: 4, 482: 4, 472: 4, 469: 4,
467: 4, 455: 4, 451: 4, 444: 4, 438: 4, 427: 4, 423: 4, 407: 4, 400: 4, 37
7: 4, 367: 4, 363: 4, 351: 4, 329: 4, 3578: 3, 3449: 3, 3161: 3, 2570: 3,
2542: 3, 2538: 3, 2533: 3, 2400: 3, 2378: 3, 2256: 3, 2135: 3, 2132: 3, 20
82: 3, 1857: 3, 1809: 3, 1800: 3, 1788: 3, 1752: 3, 1699: 3, 1667: 3, 165
7: 3, 1655: 3, 1651: 3, 1639: 3, 1605: 3, 1599: 3, 1595: 3, 1589: 3, 1552:
3, 1541: 3, 1523: 3, 1521: 3, 1516: 3, 1503: 3, 1502: 3, 1492: 3, 1482: 3,
1463: 3, 1448: 3, 1384: 3, 1375: 3, 1365: 3, 1360: 3, 1338: 3, 1305: 3, 12
97: 3, 1291: 3, 1286: 3, 1284: 3, 1282: 3, 1267: 3, 1266: 3, 1258: 3, 123
1: 3, 1214: 3, 1203: 3, 1166: 3, 1160: 3, 1159: 3, 1142: 3, 1135: 3, 1132:
3, 1131: 3, 1121: 3, 1115: 3, 1113: 3, 1110: 3, 1109: 3, 1099: 3, 1092: 3,
1091: 3, 1087: 3, 1072: 3, 1066: 3, 1063: 3, 1031: 3, 1025: 3, 1002: 3, 99
7: 3, 992: 3, 991: 3, 987: 3, 971: 3, 968: 3, 959: 3, 955: 3, 942: 3, 939:
3, 938: 3, 934: 3, 930: 3, 920: 3, 919: 3, 895: 3, 894: 3, 892: 3, 888: 3,
886: 3, 881: 3, 869: 3, 858: 3, 840: 3, 838: 3, 837: 3, 836: 3, 832: 3, 83
1: 3, 825: 3, 812: 3, 796: 3, 782: 3, 777: 3, 755: 3, 752: 3, 749: 3, 745:
3, 744: 3, 743: 3, 733: 3, 731: 3, 724: 3, 721: 3, 717: 3, 712: 3, 708: 3,
707: 3, 698: 3, 687: 3, 684: 3, 679: 3, 675: 3, 674: 3, 669: 3, 666: 3, 66
4: 3, 657: 3, 645: 3, 638: 3, 631: 3, 630: 3, 621: 3, 617: 3, 615: 3, 612:
3, 609: 3, 607: 3, 605: 3, 603: 3, 602: 3, 599: 3, 596: 3, 590: 3, 582: 3,
577: 3, 576: 3, 574: 3, 569: 3, 563: 3, 555: 3, 546: 3, 541: 3, 537: 3, 53
0: 3, 527: 3, 518: 3, 513: 3, 499: 3, 498: 3, 490: 3, 481: 3, 479: 3, 465:
3, 439: 3, 436: 3, 434: 3, 424: 3, 421: 3, 420: 3, 419: 3, 411: 3, 385: 3,
15456: 2, 12636: 2, 12460: 2, 7004: 2, 6972: 2, 6666: 2, 6302: 2, 6275: 2,
5146: 2, 4979: 2, 4892: 2, 4656: 2, 4247: 2, 4232: 2, 4223: 2, 4200: 2, 39
77: 2, 3780: 2, 3759: 2, 3663: 2, 3626: 2, 3618: 2, 3570: 2, 3560: 2, 352
2: 2, 3505: 2, 3479: 2, 3477: 2, 3459: 2, 3422: 2, 3399: 2, 3361: 2, 3348:
2, 3318: 2, 3315: 2, 3299: 2, 3282: 2, 3192: 2, 3165: 2, 3119: 2, 3102: 2,
3092: 2, 3080: 2, 3014: 2, 3011: 2, 2886: 2, 2883: 2, 2868: 2, 2822: 2, 28
04: 2, 2791: 2, 2698: 2, 2691: 2, 2639: 2, 2620: 2, 2572: 2, 2561: 2, 255
0: 2, 2541: 2, 2506: 2, 2496: 2, 2484: 2, 2440: 2, 2416: 2, 2408: 2, 2277:
2, 2266: 2, 2260: 2, 2258: 2, 2257: 2, 2188: 2, 2145: 2, 2129: 2, 2100: 2,
2098: 2, 2063: 2, 2059: 2, 2014: 2, 2006: 2, 1988: 2, 1974: 2, 1966: 2, 19
54: 2, 1952: 2, 1948: 2, 1944: 2, 1943: 2, 1941: 2, 1931: 2, 1921: 2, 191
3: 2, 1912: 2, 1901: 2, 1900: 2, 1896: 2, 1872: 2, 1864: 2, 1847: 2, 1845:
2, 1833: 2, 1831: 2, 1812: 2, 1806: 2, 1799: 2, 1796: 2, 1787: 2, 1775: 2,
1770: 2, 1742: 2, 1741: 2, 1740: 2, 1735: 2, 1729: 2, 1707: 2, 1704: 2, 16
95: 2, 1690: 2, 1687: 2, 1677: 2, 1658: 2, 1656: 2, 1650: 2, 1649: 2, 164
3: 2, 1637: 2, 1629: 2, 1614: 2, 1612: 2, 1610: 2, 1602: 2, 1598: 2, 1597:
2, 1561: 2, 1560: 2, 1558: 2, 1557: 2, 1555: 2, 1549: 2, 1542: 2, 1537: 2,
1530: 2, 1526: 2, 1525: 2, 1514: 2, 1513: 2, 1511: 2, 1496: 2, 1466: 2, 14
58: 2, 1446: 2, 1431: 2, 1415: 2, 1409: 2, 1408: 2, 1401: 2, 1395: 2, 139
0: 2, 1378: 2, 1373: 2, 1372: 2, 1371: 2, 1366: 2, 1364: 2, 1362: 2, 1352:
2, 1349: 2, 1345: 2, 1343: 2, 1342: 2, 1336: 2, 1334: 2, 1333: 2, 1330: 2,
1325: 2, 1321: 2, 1316: 2, 1314: 2, 1312: 2, 1296: 2, 1293: 2, 1288: 2, 12
56: 2, 1254: 2, 1245: 2, 1243: 2, 1242: 2, 1240: 2, 1237: 2, 1236: 2, 123
2: 2, 1228: 2, 1227: 2, 1223: 2, 1216: 2, 1209: 2, 1206: 2, 1201: 2, 1198:
2, 1197: 2, 1189: 2, 1185: 2, 1178: 2, 1177: 2, 1173: 2, 1170: 2, 1168: 2,
1157: 2, 1155: 2, 1153: 2, 1149: 2, 1148: 2, 1146: 2, 1140: 2, 1136: 2, 11
28: 2, 1127: 2, 1126: 2, 1125: 2, 1124: 2, 1122: 2, 1120: 2, 1111: 2, 110
```

```
0: 2, 1098: 2, 1090: 2, 1088: 2, 1086: 2, 1083: 2, 1075: 2, 1074: 2, 1073:
2, 1062: 2, 1060: 2, 1052: 2, 1051: 2, 1038: 2, 1037: 2, 1033: 2, 1028: 2,
1027: 2, 1023: 2, 1021: 2, 1019: 2, 1017: 2, 1015: 2, 1006: 2, 1005: 2, 10
04: 2, 1003: 2, 994: 2, 990: 2, 984: 2, 978: 2, 977: 2, 975: 2, 974: 2, 96
6: 2, 963: 2, 962: 2, 961: 2, 958: 2, 956: 2, 951: 2, 949: 2, 948: 2, 947:
2, 944: 2, 940: 2, 935: 2, 933: 2, 932: 2, 926: 2, 921: 2, 916: 2, 906: 2,
903: 2, 901: 2, 899: 2, 893: 2, 891: 2, 890: 2, 889: 2, 887: 2, 883: 2, 87
9: 2, 878: 2, 875: 2, 872: 2, 868: 2, 866: 2, 864: 2, 861: 2, 856: 2, 855:
2, 854: 2, 852: 2, 851: 2, 850: 2, 849: 2, 841: 2, 834: 2, 833: 2, 830: 2,
827: 2, 826: 2, 821: 2, 818: 2, 805: 2, 803: 2, 801: 2, 797: 2, 793: 2, 78
8: 2, 783: 2, 775: 2, 771: 2, 766: 2, 765: 2, 764: 2, 762: 2, 756: 2, 754:
2, 753: 2, 739: 2, 737: 2, 735: 2, 732: 2, 727: 2, 722: 2, 719: 2, 718: 2,
715: 2, 709: 2, 706: 2, 704: 2, 701: 2, 700: 2, 689: 2, 688: 2, 683: 2, 68
2: 2, 672: 2, 670: 2, 661: 2, 656: 2, 648: 2, 646: 2, 641: 2, 639: 2, 634:
2, 629: 2, 627: 2, 620: 2, 608: 2, 601: 2, 600: 2, 597: 2, 595: 2, 588: 2,
586: 2, 585: 2, 575: 2, 566: 2, 565: 2, 556: 2, 549: 2, 536: 2, 533: 2, 52
9: 2, 519: 2, 493: 2, 485: 2, 476: 2, 459: 2, 430: 2, 399: 2, 393: 2, 1511
98: 1, 120077: 1, 80659: 1, 66595: 1, 66142: 1, 65482: 1, 65308: 1, 63914:
1, 63239: 1, 54736: 1, 54049: 1, 50764: 1, 49681: 1, 46849: 1, 46322: 1, 4
4442: 1, 43873: 1, 42297: 1, 42208: 1, 41976: 1, 40824: 1, 40767: 1, 4031
4: 1, 39473: 1, 38908: 1, 37942: 1, 36681: 1, 36231: 1, 36051: 1, 34361:
1, 34245: 1, 33741: 1, 33115: 1, 32779: 1, 31748: 1, 31357: 1, 29622: 1, 2
8166: 1, 27550: 1, 26128: 1, 25965: 1, 25882: 1, 25820: 1, 25507: 1, 2523
9: 1, 24924: 1, 24647: 1, 24254: 1, 24157: 1, 24076: 1, 23822: 1, 23790:
1, 23112: 1, 22393: 1, 22155: 1, 22151: 1, 21953: 1, 21590: 1, 21451: 1, 2
1065: 1, 20786: 1, 20490: 1, 20434: 1, 20077: 1, 20064: 1, 19612: 1, 1906
5: 1, 19059: 1, 18993: 1, 18719: 1, 18589: 1, 18570: 1, 18568: 1, 18545:
1, 18478: 1, 18254: 1, 18244: 1, 18152: 1, 18005: 1, 17950: 1, 17802: 1, 1
7747: 1, 17605: 1, 17518: 1, 17475: 1, 17468: 1, 17287: 1, 17263: 1, 1696
0: 1, 16943: 1, 16914: 1, 16898: 1, 16782: 1, 16699: 1, 16487: 1, 16244:
1, 16115: 1, 16071: 1, 16038: 1, 15967: 1, 15686: 1, 15684: 1, 15545: 1, 1
5538: 1, 15318: 1, 15257: 1, 15206: 1, 15167: 1, 15140: 1, 14988: 1, 1474
9: 1, 14745: 1, 14620: 1, 14604: 1, 14442: 1, 14158: 1, 14107: 1, 14059:
1, 14046: 1, 14027: 1, 13935: 1, 13841: 1, 13766: 1, 13582: 1, 13444: 1, 1
3359: 1, 13252: 1, 13236: 1, 13231: 1, 13176: 1, 13023: 1, 12985: 1, 1291
2: 1, 12861: 1, 12827: 1, 12790: 1, 12712: 1, 12660: 1, 12652: 1, 12641:
1, 12610: 1, 12568: 1, 12437: 1, 12387: 1, 12366: 1, 12308: 1, 12306: 1, 1
2301: 1, 12221: 1, 12197: 1, 12158: 1, 12114: 1, 12096: 1, 12061: 1, 1202
7: 1, 12025: 1, 11993: 1, 11955: 1, 11950: 1, 11912: 1, 11889: 1, 11878:
1, 11786: 1, 11783: 1, 11753: 1, 11721: 1, 11630: 1, 11616: 1, 11559: 1, 1
1544: 1, 11540: 1, 11436: 1, 11404: 1, 11315: 1, 11117: 1, 11102: 1, 1108
0: 1, 10901: 1, 10833: 1, 10779: 1, 10744: 1, 10683: 1, 10579: 1, 10546:
1, 10523: 1, 10442: 1, 10314: 1, 10274: 1, 10246: 1, 10115: 1, 10047: 1, 1
0021: 1, 10015: 1, 9991: 1, 9985: 1, 9971: 1, 9959: 1, 9951: 1, 9916: 1, 9
861: 1, 9829: 1, 9807: 1, 9731: 1, 9690: 1, 9653: 1, 9645: 1, 9607: 1, 960
5: 1, 9588: 1, 9585: 1, 9540: 1, 9534: 1, 9529: 1, 9435: 1, 9410: 1, 9402:
1, 9376: 1, 9274: 1, 9263: 1, 9241: 1, 9217: 1, 9210: 1, 9186: 1, 9132: 1,
9116: 1, 9057: 1, 9032: 1, 9026: 1, 9025: 1, 9011: 1, 9001: 1, 8985: 1, 89
45: 1, 8889: 1, 8888: 1, 8870: 1, 8869: 1, 8787: 1, 8775: 1, 8747: 1, 869
2: 1, 8573: 1, 8543: 1, 8503: 1, 8479: 1, 8434: 1, 8402: 1, 8348: 1, 8320:
1, 8300: 1, 8295: 1, 8294: 1, 8268: 1, 8234: 1, 8230: 1, 8218: 1, 8196: 1,
8166: 1, 8160: 1, 8142: 1, 8128: 1, 8122: 1, 8097: 1, 8084: 1, 8082: 1, 80
79: 1, 8004: 1, 7978: 1, 7974: 1, 7964: 1, 7963: 1, 7962: 1, 7945: 1, 792
0: 1, 7894: 1, 7877: 1, 7855: 1, 7852: 1, 7806: 1, 7803: 1, 7796: 1, 7768:
1, 7727: 1, 7628: 1, 7597: 1, 7592: 1, 7572: 1, 7544: 1, 7541: 1, 7520: 1,
7519: 1, 7510: 1, 7495: 1, 7421: 1, 7384: 1, 7363: 1, 7318: 1, 7312: 1, 72
70: 1, 7259: 1, 7258: 1, 7248: 1, 7246: 1, 7236: 1, 7224: 1, 7221: 1, 721
4: 1, 7199: 1, 7185: 1, 7173: 1, 7168: 1, 7159: 1, 7137: 1, 7124: 1, 7109:
1, 7081: 1, 7073: 1, 7070: 1, 7056: 1, 7048: 1, 7042: 1, 7001: 1, 7000: 1,
6976: 1, 6956: 1, 6947: 1, 6935: 1, 6899: 1, 6877: 1, 6853: 1, 6825: 1, 68
24: 1, 6817: 1, 6813: 1, 6777: 1, 6728: 1, 6724: 1, 6717: 1, 6700: 1, 669
```

```
6: 1, 6670: 1, 6661: 1, 6620: 1, 6611: 1, 6610: 1, 6589: 1, 6578: 1, 6572:
1, 6560: 1, 6546: 1, 6529: 1, 6526: 1, 6521: 1, 6489: 1, 6437: 1, 6408: 1,
6394: 1, 6373: 1, 6360: 1, 6333: 1, 6323: 1, 6317: 1, 6304: 1, 6301: 1, 62
95: 1, 6291: 1, 6264: 1, 6255: 1, 6250: 1, 6249: 1, 6241: 1, 6172: 1, 616
4: 1, 6154: 1, 6119: 1, 6116: 1, 6100: 1, 6083: 1, 6073: 1, 6067: 1, 6011:
1, 6010: 1, 6008: 1, 6004: 1, 5992: 1, 5979: 1, 5968: 1, 5958: 1, 5950: 1,
5939: 1, 5928: 1, 5918: 1, 5901: 1, 5882: 1, 5881: 1, 5862: 1, 5859: 1, 58
13: 1, 5806: 1, 5796: 1, 5779: 1, 5777: 1, 5772: 1, 5767: 1, 5762: 1, 575
8: 1, 5748: 1, 5747: 1, 5745: 1, 5740: 1, 5732: 1, 5730: 1, 5722: 1, 5670:
1, 5656: 1, 5650: 1, 5649: 1, 5641: 1, 5628: 1, 5592: 1, 5590: 1, 5551: 1,
5533: 1, 5529: 1, 5525: 1, 5522: 1, 5501: 1, 5494: 1, 5485: 1, 5481: 1, 54
73: 1, 5463: 1, 5455: 1, 5450: 1, 5437: 1, 5431: 1, 5423: 1, 5420: 1, 541
4: 1, 5408: 1, 5406: 1, 5403: 1, 5344: 1, 5332: 1, 5275: 1, 5259: 1, 5241:
1, 5213: 1, 5193: 1, 5169: 1, 5165: 1, 5157: 1, 5145: 1, 5143: 1, 5140: 1,
5138: 1, 5135: 1, 5127: 1, 5106: 1, 5105: 1, 5100: 1, 5069: 1, 5068: 1, 50
64: 1, 5061: 1, 5022: 1, 5012: 1, 5010: 1, 4981: 1, 4971: 1, 4968: 1, 495
2: 1, 4949: 1, 4935: 1, 4926: 1, 4918: 1, 4917: 1, 4913: 1, 4903: 1, 4898:
1, 4897: 1, 4895: 1, 4875: 1, 4870: 1, 4868: 1, 4856: 1, 4850: 1, 4841: 1,
4840: 1, 4837: 1, 4835: 1, 4822: 1, 4816: 1, 4803: 1, 4797: 1, 4794: 1, 47
88: 1, 4785: 1, 4779: 1, 4771: 1, 4769: 1, 4765: 1, 4760: 1, 4746: 1, 474
3: 1, 4741: 1, 4734: 1, 4718: 1, 4711: 1, 4683: 1, 4647: 1, 4642: 1, 4609:
1, 4587: 1, 4585: 1, 4582: 1, 4575: 1, 4573: 1, 4572: 1, 4562: 1, 4560: 1,
4553: 1, 4530: 1, 4517: 1, 4494: 1, 4483: 1, 4441: 1, 4436: 1, 4431: 1, 44
18: 1, 4417: 1, 4416: 1, 4415: 1, 4401: 1, 4400: 1, 4399: 1, 4392: 1, 438
5: 1, 4371: 1, 4370: 1, 4364: 1, 4362: 1, 4345: 1, 4344: 1, 4337: 1, 4328:
1, 4327: 1, 4322: 1, 4321: 1, 4301: 1, 4284: 1, 4278: 1, 4275: 1, 4273: 1,
4261: 1, 4258: 1, 4233: 1, 4224: 1, 4220: 1, 4207: 1, 4205: 1, 4199: 1, 41
93: 1, 4190: 1, 4189: 1, 4181: 1, 4178: 1, 4177: 1, 4172: 1, 4170: 1, 415
4: 1, 4152: 1, 4151: 1, 4146: 1, 4145: 1, 4140: 1, 4132: 1, 4131: 1, 4128:
1, 4125: 1, 4109: 1, 4106: 1, 4099: 1, 4092: 1, 4089: 1, 4086: 1, 4076: 1,
4074: 1, 4069: 1, 4038: 1, 4035: 1, 4025: 1, 4024: 1, 4021: 1, 4018: 1, 40
16: 1, 4013: 1, 4003: 1, 4000: 1, 3996: 1, 3988: 1, 3987: 1, 3986: 1, 398
5: 1, 3970: 1, 3964: 1, 3961: 1, 3959: 1, 3958: 1, 3957: 1, 3949: 1, 3923:
1, 3921: 1, 3919: 1, 3918: 1, 3911: 1, 3905: 1, 3904: 1, 3894: 1, 3891: 1,
3878: 1, 3873: 1, 3869: 1, 3867: 1, 3866: 1, 3865: 1, 3862: 1, 3858: 1, 38
45: 1, 3839: 1, 3835: 1, 3829: 1, 3823: 1, 3822: 1, 3807: 1, 3796: 1, 379
1: 1, 3779: 1, 3773: 1, 3766: 1, 3763: 1, 3760: 1, 3756: 1, 3744: 1, 3734:
1, 3731: 1, 3718: 1, 3713: 1, 3712: 1, 3711: 1, 3703: 1, 3701: 1, 3681: 1,
3680: 1, 3675: 1, 3669: 1, 3653: 1, 3646: 1, 3641: 1, 3637: 1, 3635: 1, 36
33: 1, 3627: 1, 3617: 1, 3610: 1, 3609: 1, 3606: 1, 3595: 1, 3590: 1, 357
7: 1, 3559: 1, 3558: 1, 3554: 1, 3552: 1, 3550: 1, 3549: 1, 3542: 1, 3541:
1, 3539: 1, 3534: 1, 3531: 1, 3528: 1, 3503: 1, 3501: 1, 3499: 1, 3496: 1,
3493: 1, 3484: 1, 3471: 1, 3469: 1, 3466: 1, 3465: 1, 3455: 1, 3446: 1, 34
45: 1, 3444: 1, 3443: 1, 3437: 1, 3427: 1, 3406: 1, 3392: 1, 3390: 1, 338
9: 1, 3388: 1, 3384: 1, 3375: 1, 3371: 1, 3366: 1, 3364: 1, 3360: 1, 3351:
1, 3349: 1, 3347: 1, 3346: 1, 3337: 1, 3335: 1, 3324: 1, 3323: 1, 3321: 1,
3316: 1, 3313: 1, 3298: 1, 3295: 1, 3291: 1, 3288: 1, 3285: 1, 3283: 1, 32
80: 1, 3279: 1, 3277: 1, 3273: 1, 3269: 1, 3249: 1, 3248: 1, 3246: 1, 323
7: 1, 3236: 1, 3233: 1, 3231: 1, 3229: 1, 3225: 1, 3224: 1, 3218: 1, 3211:
1, 3203: 1, 3190: 1, 3176: 1, 3174: 1, 3162: 1, 3160: 1, 3152: 1, 3146: 1,
3145: 1, 3143: 1, 3139: 1, 3132: 1, 3129: 1, 3118: 1, 3108: 1, 3107: 1, 31
06: 1, 3100: 1, 3096: 1, 3088: 1, 3079: 1, 3078: 1, 3055: 1, 3053: 1, 305
0: 1, 3042: 1, 3034: 1, 3032: 1, 3026: 1, 3024: 1, 3021: 1, 3020: 1, 3012:
1, 3009: 1, 3008: 1, 3007: 1, 3006: 1, 3003: 1, 3001: 1, 3000: 1, 2997: 1,
2990: 1, 2988: 1, 2987: 1, 2986: 1, 2981: 1, 2978: 1, 2975: 1, 2972: 1, 29
71: 1, 2969: 1, 2967: 1, 2959: 1, 2955: 1, 2949: 1, 2943: 1, 2939: 1, 293
5: 1, 2931: 1, 2925: 1, 2921: 1, 2918: 1, 2912: 1, 2910: 1, 2900: 1, 2899:
1, 2882: 1, 2879: 1, 2873: 1, 2872: 1, 2862: 1, 2860: 1, 2857: 1, 2849: 1,
2847: 1, 2845: 1, 2838: 1, 2831: 1, 2830: 1, 2829: 1, 2827: 1, 2818: 1, 28
10: 1, 2807: 1, 2797: 1, 2790: 1, 2789: 1, 2781: 1, 2775: 1, 2772: 1, 276
7: 1, 2756: 1, 2751: 1, 2745: 1, 2735: 1, 2722: 1, 2715: 1, 2713: 1, 2712:
```

```
1, 2711: 1, 2708: 1, 2700: 1, 2699: 1, 2695: 1, 2693: 1, 2687: 1, 2683: 1,
2682: 1, 2675: 1, 2671: 1, 2669: 1, 2666: 1, 2665: 1, 2664: 1, 2663: 1, 26
58: 1, 2656: 1, 2655: 1, 2654: 1, 2653: 1, 2648: 1, 2645: 1, 2644: 1, 264
1: 1, 2637: 1, 2636: 1, 2635: 1, 2624: 1, 2616: 1, 2615: 1, 2608: 1, 2599:
1, 2594: 1, 2593: 1, 2591: 1, 2590: 1, 2589: 1, 2584: 1, 2577: 1, 2576: 1,
2573: 1, 2560: 1, 2559: 1, 2556: 1, 2554: 1, 2548: 1, 2544: 1, 2543: 1, 25
39: 1, 2534: 1, 2527: 1, 2519: 1, 2518: 1, 2517: 1, 2511: 1, 2508: 1, 250
3: 1, 2500: 1, 2499: 1, 2498: 1, 2493: 1, 2492: 1, 2489: 1, 2485: 1, 2480:
1, 2479: 1, 2477: 1, 2475: 1, 2464: 1, 2463: 1, 2462: 1, 2461: 1, 2459: 1,
2455: 1, 2451: 1, 2450: 1, 2449: 1, 2447: 1, 2445: 1, 2439: 1, 2429: 1, 24
25: 1, 2420: 1, 2419: 1, 2415: 1, 2414: 1, 2410: 1, 2409: 1, 2405: 1, 240
2: 1, 2396: 1, 2393: 1, 2392: 1, 2385: 1, 2383: 1, 2377: 1, 2373: 1, 2368:
1, 2366: 1, 2365: 1, 2362: 1, 2356: 1, 2350: 1, 2339: 1, 2334: 1, 2333: 1,
2332: 1, 2329: 1, 2328: 1, 2327: 1, 2325: 1, 2320: 1, 2319: 1, 2311: 1, 23
08: 1, 2303: 1, 2300: 1, 2299: 1, 2298: 1, 2289: 1, 2288: 1, 2287: 1, 228
6: 1, 2284: 1, 2283: 1, 2281: 1, 2280: 1, 2279: 1, 2276: 1, 2275: 1, 2274:
1, 2273: 1, 2271: 1, 2268: 1, 2261: 1, 2259: 1, 2253: 1, 2251: 1, 2250: 1,
2247: 1, 2243: 1, 2240: 1, 2238: 1, 2235: 1, 2234: 1, 2233: 1, 2227: 1, 22
25: 1, 2223: 1, 2216: 1, 2213: 1, 2207: 1, 2206: 1, 2205: 1, 2200: 1, 219
9: 1, 2196: 1, 2190: 1, 2187: 1, 2186: 1, 2185: 1, 2184: 1, 2180: 1, 2179:
1, 2177: 1, 2173: 1, 2168: 1, 2167: 1, 2166: 1, 2165: 1, 2164: 1, 2163: 1,
2161: 1, 2160: 1, 2157: 1, 2156: 1, 2155: 1, 2153: 1, 2151: 1, 2149: 1, 21
48: 1, 2144: 1, 2140: 1, 2139: 1, 2137: 1, 2133: 1, 2127: 1, 2124: 1, 212
3: 1, 2117: 1, 2114: 1, 2109: 1, 2108: 1, 2104: 1, 2103: 1, 2099: 1, 2095:
1, 2094: 1, 2091: 1, 2089: 1, 2087: 1, 2083: 1, 2076: 1, 2072: 1, 2071: 1,
2066: 1, 2062: 1, 2060: 1, 2058: 1, 2053: 1, 2052: 1, 2050: 1, 2048: 1, 20
47: 1, 2044: 1, 2041: 1, 2035: 1, 2033: 1, 2032: 1, 2031: 1, 2030: 1, 202
6: 1, 2024: 1, 2023: 1, 2015: 1, 2011: 1, 2010: 1, 2009: 1, 2007: 1, 2004:
1, 2001: 1, 1996: 1, 1992: 1, 1987: 1, 1982: 1, 1977: 1, 1976: 1, 1975: 1,
1973: 1, 1971: 1, 1963: 1, 1961: 1, 1960: 1, 1959: 1, 1956: 1, 1955: 1, 19
51: 1, 1946: 1, 1942: 1, 1930: 1, 1929: 1, 1927: 1, 1925: 1, 1922: 1, 192
0: 1, 1919: 1, 1918: 1, 1917: 1, 1914: 1, 1908: 1, 1907: 1, 1906: 1, 1903:
1, 1902: 1, 1898: 1, 1897: 1, 1893: 1, 1892: 1, 1890: 1, 1889: 1, 1885: 1,
1881: 1, 1878: 1, 1876: 1, 1874: 1, 1871: 1, 1870: 1, 1866: 1, 1863: 1, 18
62: 1, 1860: 1, 1859: 1, 1854: 1, 1852: 1, 1851: 1, 1850: 1, 1849: 1, 184
8: 1, 1846: 1, 1842: 1, 1840: 1, 1838: 1, 1837: 1, 1832: 1, 1829: 1, 1825:
1, 1821: 1, 1820: 1, 1819: 1, 1818: 1, 1817: 1, 1816: 1, 1813: 1, 1811: 1,
1803: 1, 1802: 1, 1801: 1, 1798: 1, 1793: 1, 1789: 1, 1785: 1, 1781: 1, 17
80: 1, 1779: 1, 1778: 1, 1776: 1, 1773: 1, 1767: 1, 1766: 1, 1765: 1, 176
3: 1, 1762: 1, 1760: 1, 1749: 1, 1748: 1, 1746: 1, 1745: 1, 1744: 1, 1743:
1, 1738: 1, 1730: 1, 1726: 1, 1718: 1, 1717: 1, 1716: 1, 1714: 1, 1712: 1,
1711: 1, 1708: 1, 1706: 1, 1705: 1, 1702: 1, 1700: 1, 1689: 1, 1685: 1, 16
84: 1, 1683: 1, 1682: 1, 1681: 1, 1679: 1, 1678: 1, 1675: 1, 1673: 1, 167
2: 1, 1670: 1, 1669: 1, 1668: 1, 1664: 1, 1654: 1, 1645: 1, 1640: 1, 1633:
1, 1628: 1, 1627: 1, 1624: 1, 1621: 1, 1619: 1, 1618: 1, 1617: 1, 1615: 1,
1613: 1, 1611: 1, 1609: 1, 1607: 1, 1606: 1, 1604: 1, 1601: 1, 1600: 1, 15
96: 1, 1594: 1, 1593: 1, 1592: 1, 1591: 1, 1588: 1, 1586: 1, 1585: 1, 158
4: 1, 1580: 1, 1579: 1, 1574: 1, 1573: 1, 1572: 1, 1571: 1, 1568: 1, 1566:
1, 1564: 1, 1559: 1, 1556: 1, 1548: 1, 1546: 1, 1543: 1, 1540: 1, 1539: 1,
1538: 1, 1536: 1, 1535: 1, 1533: 1, 1531: 1, 1529: 1, 1528: 1, 1524: 1, 15
22: 1, 1519: 1, 1518: 1, 1517: 1, 1508: 1, 1507: 1, 1506: 1, 1501: 1, 149
7: 1, 1485: 1, 1484: 1, 1481: 1, 1479: 1, 1478: 1, 1477: 1, 1476: 1, 1473:
1, 1472: 1, 1471: 1, 1470: 1, 1469: 1, 1467: 1, 1465: 1, 1461: 1, 1460: 1,
1459: 1, 1457: 1, 1454: 1, 1452: 1, 1451: 1, 1450: 1, 1447: 1, 1445: 1, 14
44: 1, 1443: 1, 1442: 1, 1440: 1, 1438: 1, 1437: 1, 1435: 1, 1434: 1, 143
2: 1, 1430: 1, 1428: 1, 1426: 1, 1422: 1, 1421: 1, 1419: 1, 1418: 1, 1417:
1, 1416: 1, 1414: 1, 1413: 1, 1412: 1, 1410: 1, 1402: 1, 1400: 1, 1398: 1,
1397: 1, 1394: 1, 1393: 1, 1392: 1, 1388: 1, 1387: 1, 1386: 1, 1383: 1, 13
74: 1, 1370: 1, 1368: 1, 1363: 1, 1361: 1, 1357: 1, 1356: 1, 1354: 1, 135
3: 1, 1347: 1, 1346: 1, 1340: 1, 1337: 1, 1332: 1, 1331: 1, 1329: 1, 1328:
1, 1324: 1, 1323: 1, 1322: 1, 1320: 1, 1318: 1, 1317: 1, 1315: 1, 1311: 1,
```

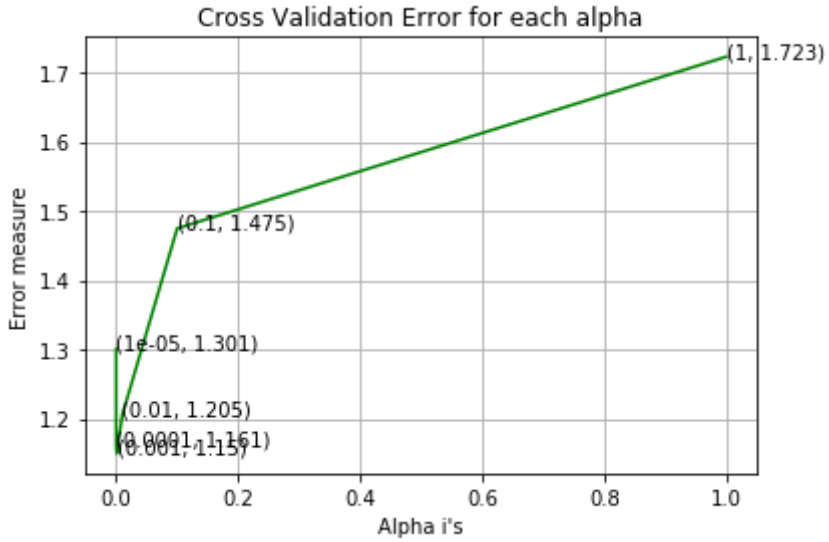
```
1310: 1, 1308: 1, 1307: 1, 1306: 1, 1304: 1, 1302: 1, 1300: 1, 1299: 1, 12
98: 1, 1295: 1, 1290: 1, 1285: 1, 1283: 1, 1281: 1, 1279: 1, 1278: 1, 127
3: 1, 1272: 1, 1264: 1, 1262: 1, 1260: 1, 1259: 1, 1257: 1, 1253: 1, 1252:
1, 1251: 1, 1250: 1, 1249: 1, 1247: 1, 1238: 1, 1235: 1, 1230: 1, 1226: 1,
1221: 1, 1220: 1, 1218: 1, 1217: 1, 1215: 1, 1213: 1, 1212: 1, 1211: 1, 12
10: 1, 1207: 1, 1205: 1, 1204: 1, 1202: 1, 1200: 1, 1199: 1, 1195: 1, 119
2: 1, 1188: 1, 1187: 1, 1186: 1, 1184: 1, 1183: 1, 1181: 1, 1179: 1, 1174:
1, 1171: 1, 1169: 1, 1165: 1, 1162: 1, 1158: 1, 1156: 1, 1154: 1, 1151: 1,
1147: 1, 1145: 1, 1137: 1, 1134: 1, 1129: 1, 1117: 1, 1116: 1, 1112: 1, 11
08: 1, 1107: 1, 1106: 1, 1102: 1, 1101: 1, 1096: 1, 1095: 1, 1094: 1, 108
5: 1, 1082: 1, 1078: 1, 1076: 1, 1064: 1, 1061: 1, 1058: 1, 1057: 1, 1055:
1, 1054: 1, 1053: 1, 1050: 1, 1048: 1, 1044: 1, 1043: 1, 1041: 1, 1040: 1,
1039: 1, 1035: 1, 1032: 1, 1030: 1, 1020: 1, 1018: 1, 1013: 1, 1012: 1, 10
11: 1, 999: 1, 998: 1, 996: 1, 989: 1, 986: 1, 985: 1, 980: 1, 979: 1, 97
6: 1, 970: 1, 969: 1, 967: 1, 965: 1, 960: 1, 954: 1, 953: 1, 950: 1, 946:
1, 943: 1, 929: 1, 928: 1, 925: 1, 924: 1, 923: 1, 922: 1, 917: 1, 915: 1,
912: 1, 911: 1, 910: 1, 905: 1, 900: 1, 898: 1, 896: 1, 885: 1, 884: 1, 88
2: 1, 876: 1, 873: 1, 865: 1, 862: 1, 860: 1, 857: 1, 848: 1, 847: 1, 846:
1, 839: 1, 828: 1, 823: 1, 820: 1, 816: 1, 813: 1, 810: 1, 806: 1, 802: 1,
798: 1, 794: 1, 792: 1, 791: 1, 790: 1, 785: 1, 784: 1, 773: 1, 770: 1, 76
3: 1, 759: 1, 758: 1, 757: 1, 751: 1, 746: 1, 736: 1, 725: 1, 723: 1, 714:
1, 703: 1, 697: 1, 694: 1, 693: 1, 692: 1, 686: 1, 685: 1, 671: 1, 667: 1,
665: 1, 652: 1, 640: 1, 637: 1, 636: 1, 622: 1, 616: 1, 611: 1, 606: 1, 59
3: 1, 564: 1, 552: 1, 551: 1, 547: 1, 523: 1, 515: 1, 491: 1, 486: 1})
```

In [47]:

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

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

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

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


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

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

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


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.3005289634031527
For values of alpha =  0.0001 The log loss is: 1.160967994200014
For values of alpha =  0.001 The log loss is: 1.1498368649869457
For values of alpha =  0.01 The log loss is: 1.2052272437064975
For values of alpha =  0.1 The log loss is: 1.4747961582286422
For values of alpha =  1 The log loss is: 1.7228412049518813
```




```
For values of best alpha =  0.001 The train log loss is: 0.639508040837042
3
For values of best alpha =  0.001 The cross validation log loss is: 1.1498
368649869457
For values of best alpha =  0.001 The test log loss is: 1.2016451435478406
```

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

Ans. Yes, it seems like!

In [48]:

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

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

In [49]:

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

```
96.806 % of word of test data appeared in train data
98.803 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [51]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [52]:

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

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

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

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

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

Stacking the three types of features

In [53]:

```
# merging gene, variance and text features

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

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

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

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

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


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

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

In [54]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_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, 56079)
(number of data points * number of features) in test data =  (665, 56079)
(number of data points * number of features) in cross validation data = (532, 56079)
```

In [55]:

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

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

4.1. Base Line Model

4.1.1. Naive Bayes

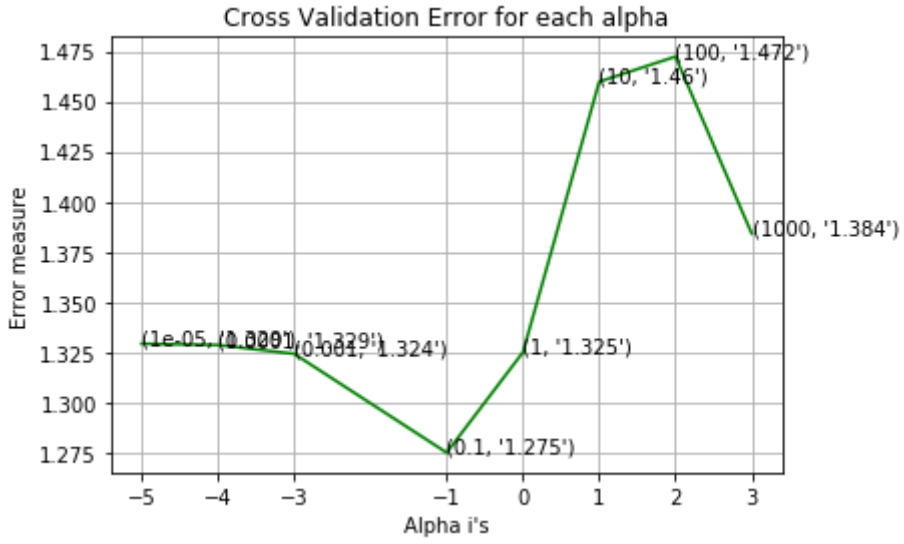
4.1.1.1. Hyper parameter tuning

In [56]:

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

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


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


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

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

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


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

```
for alpha = 1e-05
Log Loss : 1.3292016907339665
for alpha = 0.0001
Log Loss : 1.3285517634334163
for alpha = 0.001
Log Loss : 1.3241634953341932
for alpha = 0.1
Log Loss : 1.274876680492585
for alpha = 1
Log Loss : 1.3249136296441961
for alpha = 10
Log Loss : 1.4598382624368842
for alpha = 100
Log Loss : 1.4723450603714732
for alpha = 1000
Log Loss : 1.3841331326272117
```



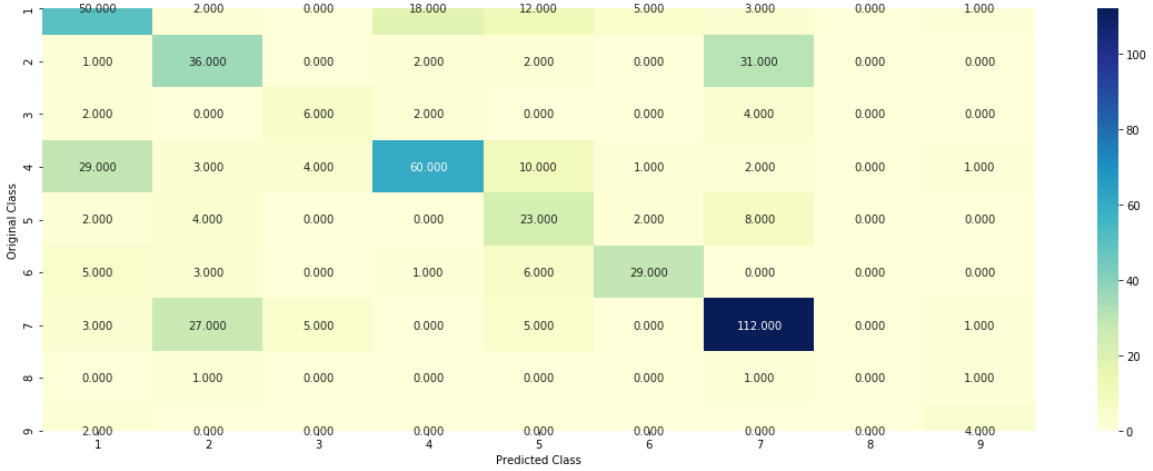

```
For values of best alpha =  0.1 The train log loss is: 0.8348146170362971
For values of best alpha =  0.1 The cross validation log loss is: 1.274876
680492585
For values of best alpha =  0.1 The test log loss is: 1.3302862385980612
```

4.1.1.2. Testing the model with best hyper paramters

In [57]:

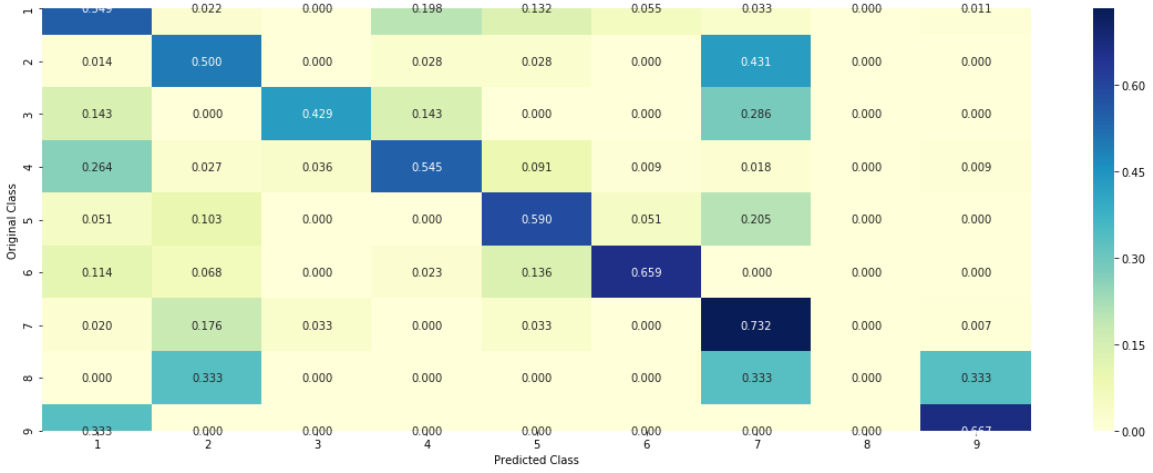
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
odules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.274876680492585
Number of missclassified point : 0.39849624060150374
-------------------- Confusion matrix --------------------
```




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


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



4.1.1.3. Feature Importance, Correctly classified point

In [115]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5941 0.0716 0.0132 0.1078 0.0379 0.0402
0.1263 0.0052 0.0038]]
Actual Class : 7
--------------------------------------------------
4 Text feature [jacobson] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0862 0.1875 0.013  0.1046 0.0339 0.0405
0.5248 0.0052 0.0043]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

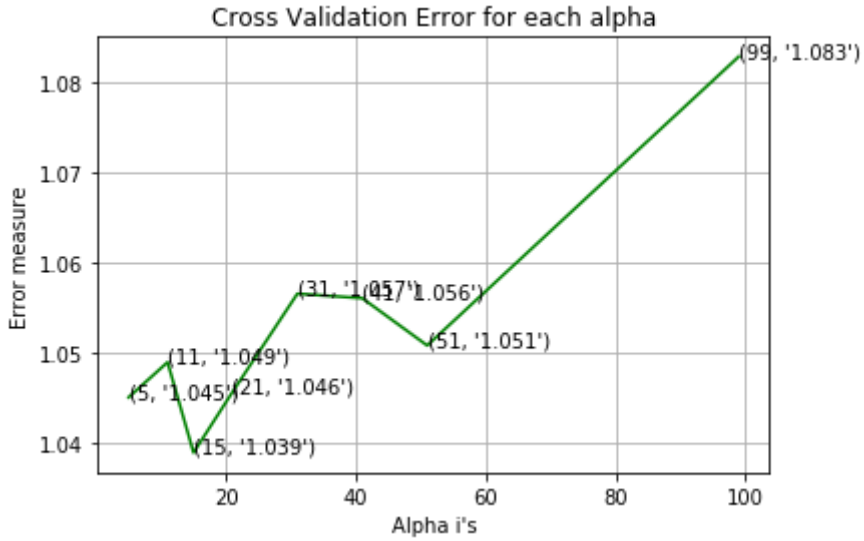
4.2.1. Hyper parameter tuning

In [60]:

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

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


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


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

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

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

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

```
for alpha = 5
Log Loss : 1.0450087682833507
for alpha = 11
Log Loss : 1.0489124509632353
for alpha = 15
Log Loss : 1.0388636940007498
for alpha = 21
Log Loss : 1.045598499599349
for alpha = 31
Log Loss : 1.056523228943271
for alpha = 41
Log Loss : 1.0560240271648624
for alpha = 51
Log Loss : 1.050769604751828
for alpha = 99
Log Loss : 1.0828683801151386
```



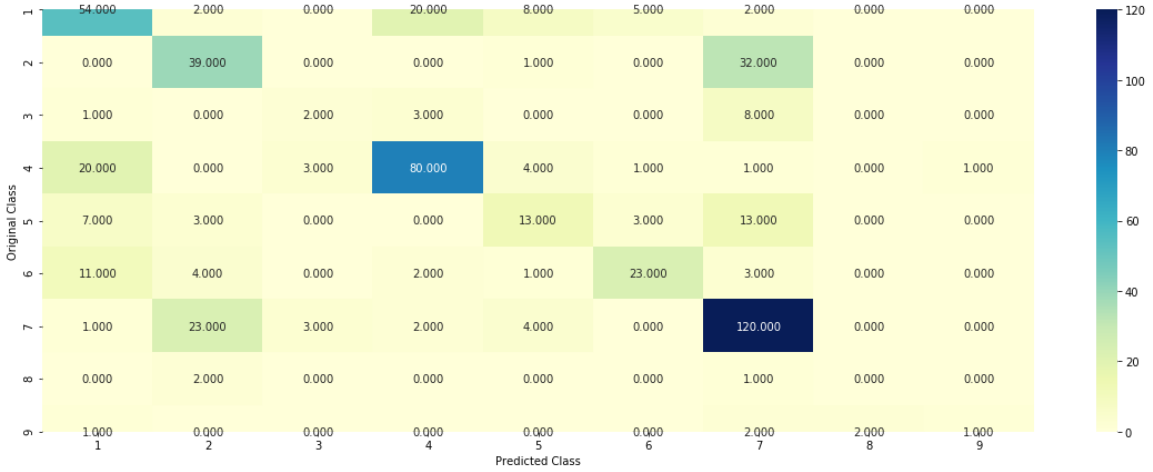

```
For values of best alpha =  15 The train log loss is: 0.6622743801074927
For values of best alpha =  15 The cross validation log loss is: 1.0388636
940007498
For values of best alpha =  15 The test log loss is: 1.1063143859715616
```

4.2.2. Testing the model with best hyper paramters

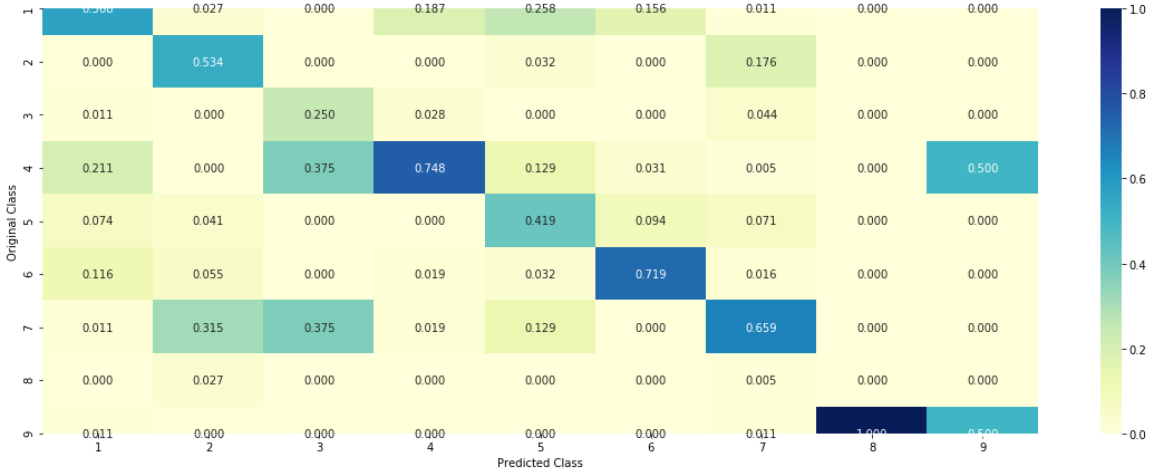
In [61]:

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

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

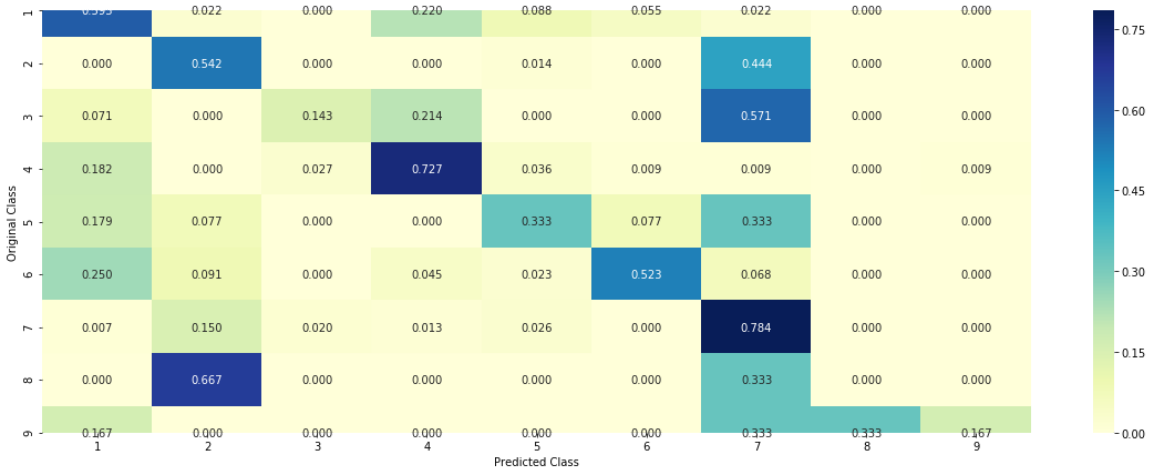
```
Log loss : 1.0388636940007498
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [62]:

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

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

```
Predicted Class : 2
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [2 7 2 2
2 2 2 2 2 2 2 2 2 2 2]
Fequency of nearest points : Counter({2: 14, 7: 1})
```

4.2.4. Sample Query Point-2

In [63]:

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

test_point_index = 100

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

```
Predicted Class : 4
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points be
longs to classes [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
Fequency of nearest points : Counter({4: 15})
```

4.3. Logistic Regression

4.3.1. With Class balancing

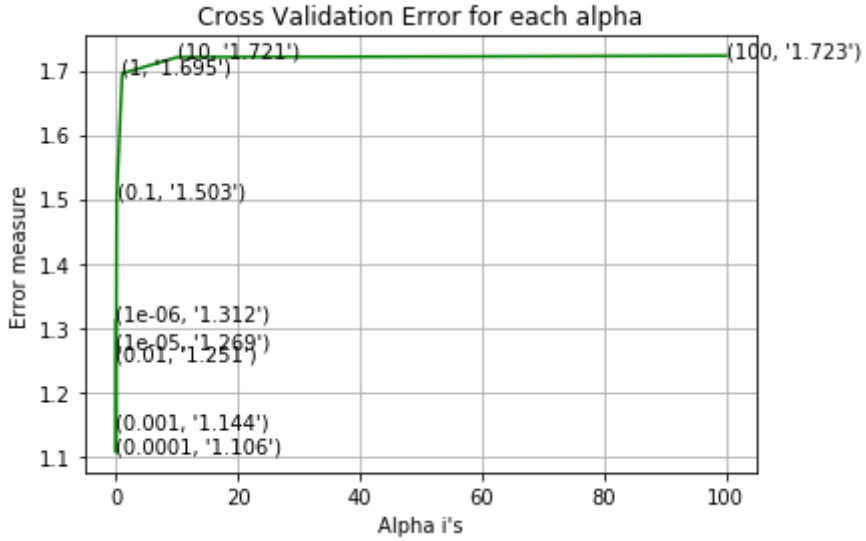
4.3.1.1. Hyper paramter tuning

In [64]:

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

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

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


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

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3116318719715243
for alpha = 1e-05
Log Loss : 1.2693914879394215
for alpha = 0.0001
Log Loss : 1.105822540061205
for alpha = 0.001
Log Loss : 1.143680991638789
for alpha = 0.01
Log Loss : 1.251359677476088
for alpha = 0.1
Log Loss : 1.5033554172357497
for alpha = 1
Log Loss : 1.6952891318661762
for alpha = 10
Log Loss : 1.720514524913257
for alpha = 100
Log Loss : 1.7228503740144863
```



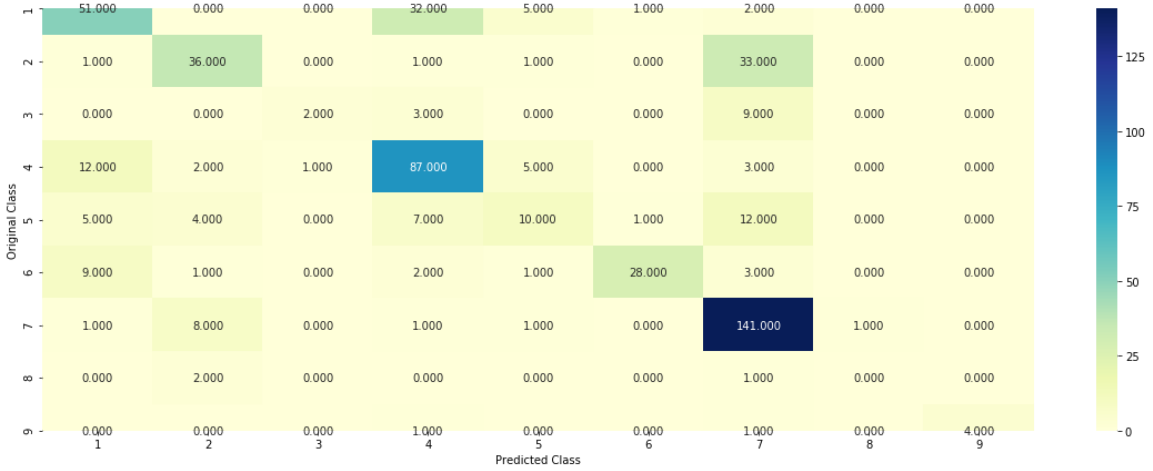

```
For values of best alpha =  0.0001 The train log loss is: 0.47584382684332
793
For values of best alpha =  0.0001 The cross validation log loss is: 1.105
822540061205
For values of best alpha =  0.0001 The test log loss is: 1.193122794133811
```

4.3.1.2. Testing the model with best hyper paramters

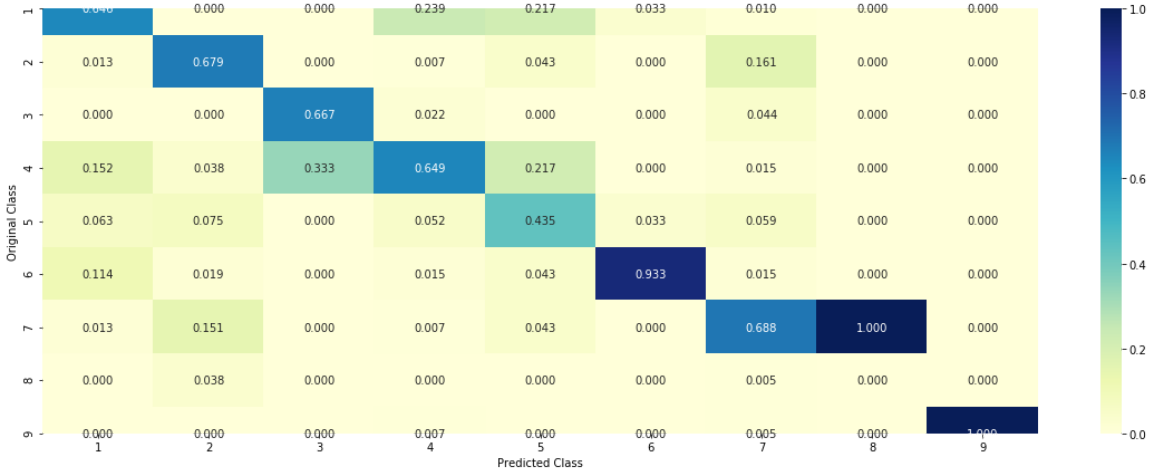
In [65]:

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

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

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

```
Log loss : 1.105822540061205
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```




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




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

4.3.1.3. Feature Importance

In [66]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], 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 [67]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0402 0.3347 0.0152 0.0981 0.0356 0.028
0.4218 0.0117 0.0146]]
Actual Class : 7
--------------------------------------------------
234 Text feature [transforming] present in test data point [True]
239 Text feature [3t3] present in test data point [True]
466 Text feature [constitutively] present in test data point [True]
487 Text feature [balb] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [68]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1888 0.0598 0.0075 0.1891 0.0155 0.0062
0.5096 0.0129 0.0106]]
Actual Class : 7
--------------------------------------------------
234 Text feature [transforming] present in test data point [True]
351 Text feature [stat] present in test data point [True]
421 Text feature [cysteine] present in test data point [True]
486 Text feature [deeper] present in test data point [True]
496 Text feature [oncoprotein] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.2. Without Class balancing

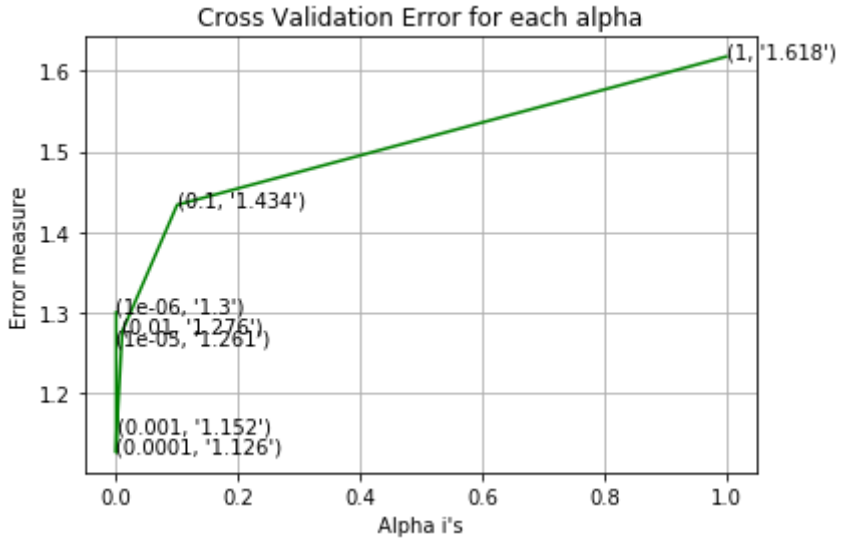
4.3.2.1. Hyper paramter tuning

In [69]:

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

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

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



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

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

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

```
plt.show()


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

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

```
for alpha = 1e-06
Log Loss : 1.3000934081400095
for alpha = 1e-05
Log Loss : 1.260577134807515
for alpha = 0.0001
Log Loss : 1.1259691595530223
for alpha = 0.001
Log Loss : 1.151690833863343
for alpha = 0.01
Log Loss : 1.27641819251939
for alpha = 0.1
Log Loss : 1.4336265026156312
for alpha = 1
Log Loss : 1.6176872438054224
```




```
For values of best alpha =  0.0001 The train log loss is: 0.45856150795822
886
For values of best alpha =  0.0001 The cross validation log loss is: 1.125
9691595530223
For values of best alpha =  0.0001 The test log loss is: 1.218111657340715
8
```

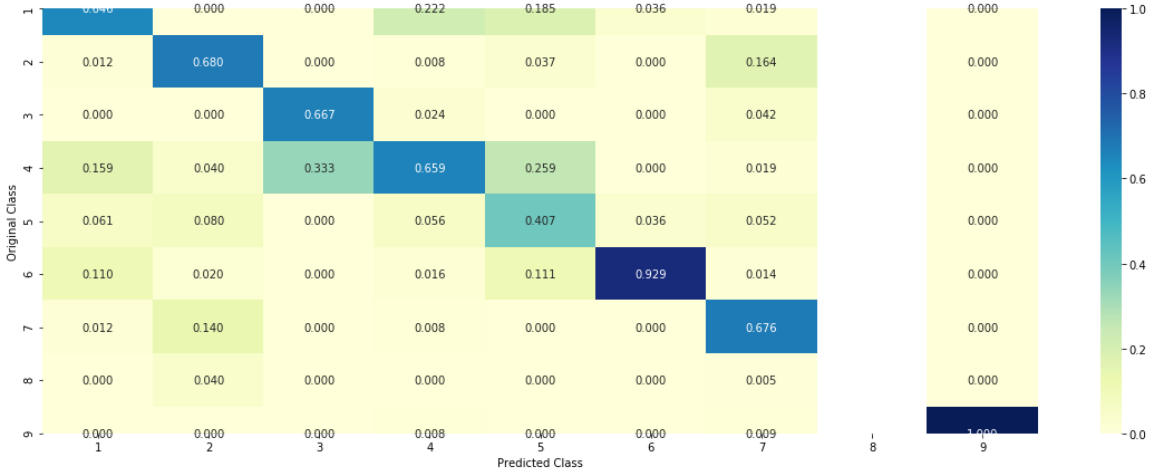
4.3.2.2. Testing model with best hyper parameters

In [70]:

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

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

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

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

```
Log loss : 1.1259691595530223
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```


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




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

4.3.2.3. Feature Importance, Correctly Classified point

In [71]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.045  0.3363 0.008  0.1198 0.0299 0.0223
0.4252 0.0096 0.004 ]]
Actual Class : 7
--------------------------------------------------
255 Text feature [transforming] present in test data point [True]
271 Text feature [3t3] present in test data point [True]
475 Text feature [balb] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [73]:

```
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0621 0.1748 0.0085 0.0566 0.0238 0.0209
0.6431 0.0061 0.004 ]]
Actual Class : 2
--------------------------------------------------
303 Text feature [folfox] present in test data point [True]
328 Text feature [btc] present in test data point [True]
371 Text feature [gallbladder] present in test data point [True]
487 Text feature [oncogene] present in test data point [True]
495 Text feature [nanomolar] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.4. Linear Support Vector Machines

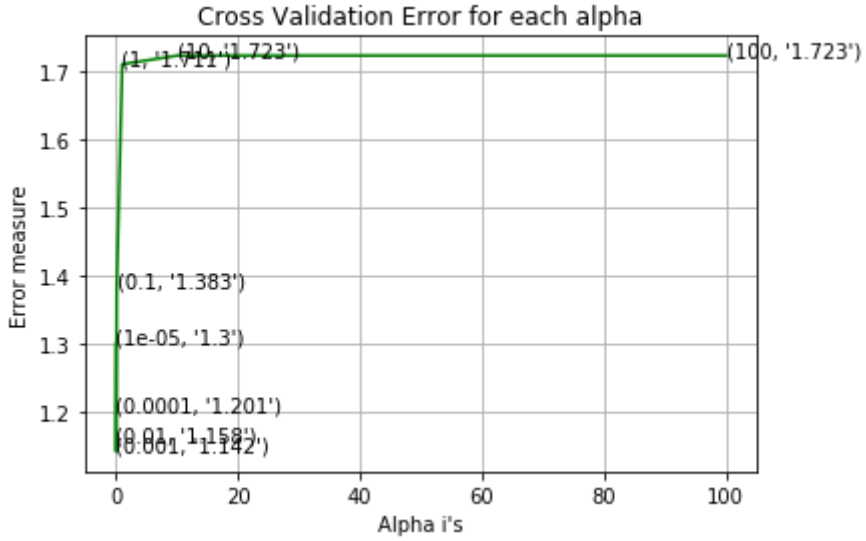
4.4.1. Hyper paramter tuning

In [74]:

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

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

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



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

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

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

```
plt.show()


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

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

```
for C = 1e-05
Log Loss : 1.3002971214715453
for C = 0.0001
Log Loss : 1.20086531700698
for C = 0.001
Log Loss : 1.142058778596847
for C = 0.01
Log Loss : 1.158425123792503
for C = 0.1
Log Loss : 1.3829105858474178
for C = 1
Log Loss : 1.7105073770818124
for C = 10
Log Loss : 1.7232030288214488
for C = 100
Log Loss : 1.7232067651662926
```




```
For values of best alpha =  0.001 The train log loss is: 0.500778351908342
6
For values of best alpha =  0.001 The cross validation log loss is: 1.1420
58778596847
For values of best alpha =  0.001 The test log loss is: 1.2420840190457343
```

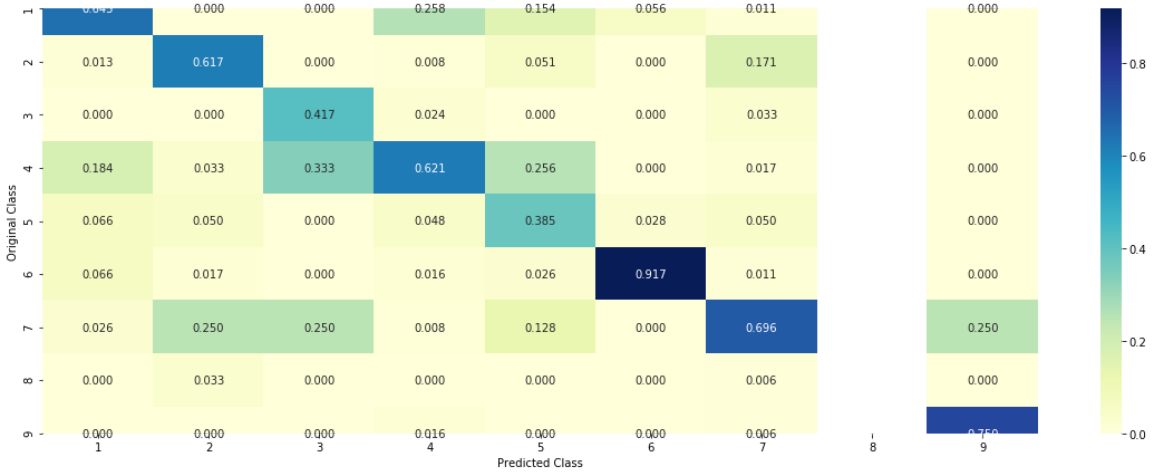
4.4.2. Testing model with best hyper parameters

In [75]:

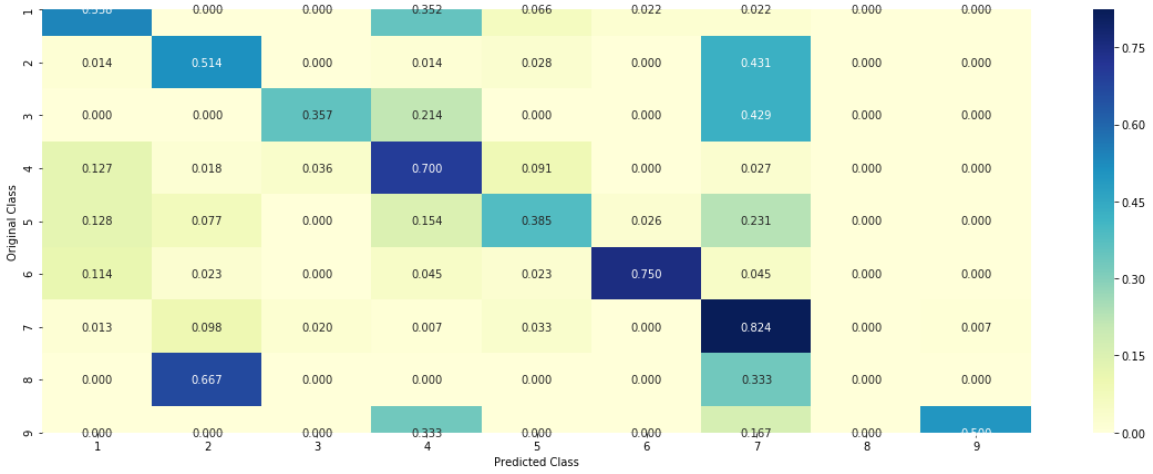
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ------------------------------
# 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')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.142058778596847
Number of mis-classified points : 0.35150375939849626
-------------------- 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 [76]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0783 0.328  0.0114 0.1471 0.0433 0.032
0.3438 0.0071 0.0091]]
Actual Class : 7
--------------------------------------------------
291 Text feature [transforming] present in test data point [True]
309 Text feature [3t3] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [77]:

```
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0746 0.1345 0.0095 0.0674 0.0245 0.0228
0.6545 0.0055 0.0067]]
Actual Class : 2
--------------------------------------------------
180 Text feature [folfox] present in test data point [True]
491 Text feature [pazopanib] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [78]:

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

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

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

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


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

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

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

```
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_err
or_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

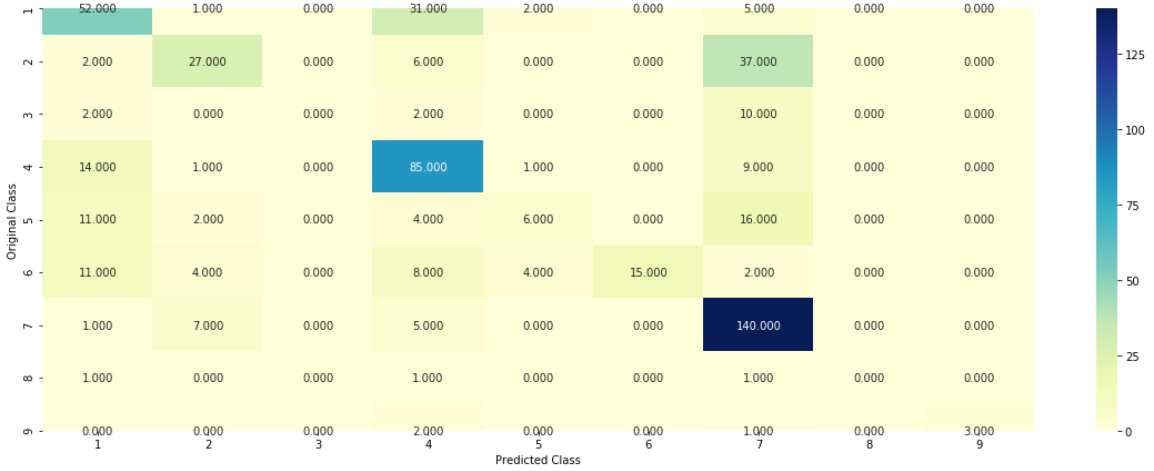
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2949349511210442
for n_estimators = 100 and max depth =  10
Log Loss : 1.2047764171271176
for n_estimators = 200 and max depth =  5
Log Loss : 1.2837833682324409
for n_estimators = 200 and max depth =  10
Log Loss : 1.194673657712319
for n_estimators = 500 and max depth =  5
Log Loss : 1.2848304647428852
for n_estimators = 500 and max depth =  10
Log Loss : 1.1918767524278169
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2867577623594677
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1885620036850304
for n_estimators = 2000 and max depth =  5
Log Loss : 1.28767470459427
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1859371671691232
For values of best estimator =  2000 The train log loss is: 0.693506937496
1269
For values of best estimator =  2000 The cross validation log loss is: 1.1
859371671691232
For values of best estimator =  2000 The test log loss is: 1.2058550908321
892
```

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

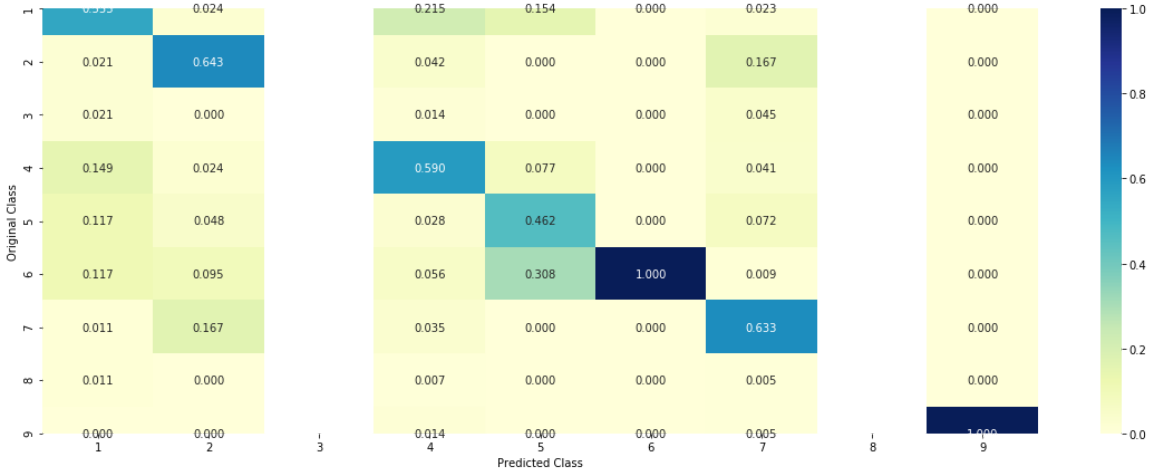
In [79]:

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

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

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

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

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

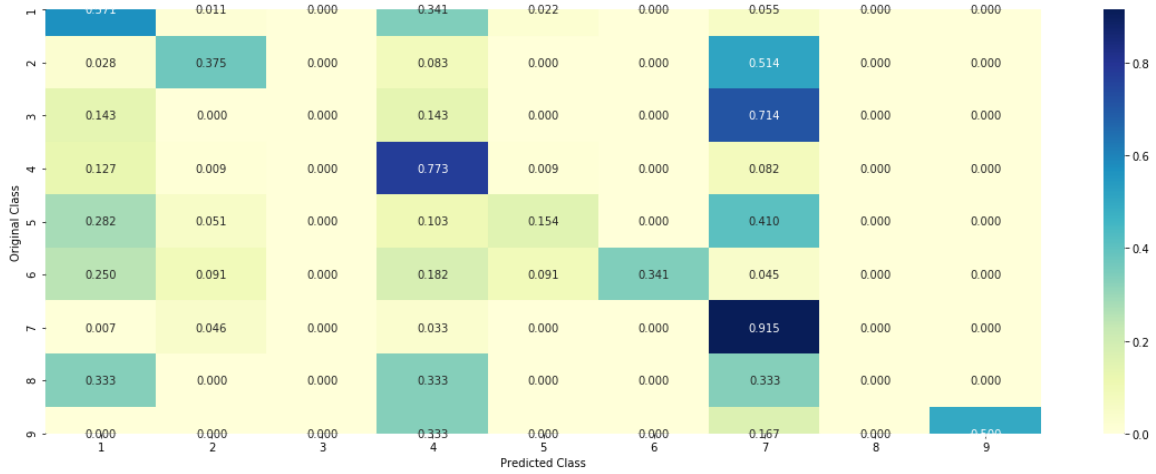
```
Log loss : 1.1859371671691232
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```




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




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [80]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_
df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_featu
re)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0721 0.1109 0.0165 0.0899 0.0458 0.0431
0.6089 0.0063 0.0065]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
8 Text feature [activated] present in test data point [True]
11 Text feature [inhibitor] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [treatment] present in test data point [True]
15 Text feature [signaling] present in test data point [True]
16 Text feature [therapy] present in test data point [True]
20 Text feature [akt] present in test data point [True]
21 Text feature [loss] present in test data point [True]
22 Text feature [3t3] present in test data point [True]
23 Text feature [patients] present in test data point [True]
24 Text feature [resistance] present in test data point [True]
25 Text feature [constitutively] present in test data point [True]
27 Text feature [drug] present in test data point [True]
29 Text feature [ba] present in test data point [True]
30 Text feature [erk] present in test data point [True]
31 Text feature [variants] present in test data point [True]
32 Text feature [inhibition] present in test data point [True]
33 Text feature [cells] present in test data point [True]
35 Text feature [downstream] present in test data point [True]
39 Text feature [f3] present in test data point [True]
41 Text feature [therapeutic] present in test data point [True]
42 Text feature [growth] present in test data point [True]
44 Text feature [expressing] present in test data point [True]
45 Text feature [autophosphorylation] present in test data point [True]
50 Text feature [cell] present in test data point [True]
51 Text feature [proliferation] present in test data point [True]
53 Text feature [imatinib] present in test data point [True]
54 Text feature [phospho] present in test data point [True]
57 Text feature [advanced] present in test data point [True]
59 Text feature [activate] present in test data point [True]
60 Text feature [protein] present in test data point [True]
62 Text feature [clinical] present in test data point [True]
64 Text feature [transforming] present in test data point [True]
66 Text feature [phosphatase] present in test data point [True]
70 Text feature [survival] present in test data point [True]
72 Text feature [sensitivity] present in test data point [True]
73 Text feature [serum] present in test data point [True]
80 Text feature [mitogen] present in test data point [True]
81 Text feature [proteins] present in test data point [True]
84 Text feature [patient] present in test data point [True]
85 Text feature [lung] present in test data point [True]
86 Text feature [lines] present in test data point [True]
97 Text feature [acquired] present in test data point [True]
Out of the top  100  features  48 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [81]:

```
test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("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_featu
re)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0703 0.374  0.0145 0.0417 0.0394 0.039
0.4101 0.0059 0.0051]]
Actuall Class : 2
---------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [suppressor] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
10 Text feature [nonsense] present in test data point [True]
11 Text feature [inhibitor] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [treatment] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [signaling] present in test data point [True]
16 Text feature [therapy] present in test data point [True]
17 Text feature [trials] present in test data point [True]
18 Text feature [receptor] present in test data point [True]
20 Text feature [akt] present in test data point [True]
21 Text feature [loss] present in test data point [True]
23 Text feature [patients] present in test data point [True]
24 Text feature [resistance] present in test data point [True]
27 Text feature [drug] present in test data point [True]
30 Text feature [erk] present in test data point [True]
31 Text feature [variants] present in test data point [True]
32 Text feature [inhibition] present in test data point [True]
33 Text feature [cells] present in test data point [True]
35 Text feature [downstream] present in test data point [True]
41 Text feature [therapeutic] present in test data point [True]
42 Text feature [growth] present in test data point [True]
47 Text feature [kinases] present in test data point [True]
49 Text feature [egfr] present in test data point [True]
50 Text feature [cell] present in test data point [True]
51 Text feature [proliferation] present in test data point [True]
52 Text feature [ic50] present in test data point [True]
55 Text feature [oncogene] present in test data point [True]
56 Text feature [months] present in test data point [True]
57 Text feature [advanced] present in test data point [True]
60 Text feature [protein] present in test data point [True]
62 Text feature [clinical] present in test data point [True]
63 Text feature [efficacy] present in test data point [True]
67 Text feature [treated] present in test data point [True]
71 Text feature [response] present in test data point [True]
72 Text feature [sensitivity] present in test data point [True]
78 Text feature [pten] present in test data point [True]
81 Text feature [proteins] present in test data point [True]
83 Text feature [harboring] present in test data point [True]
84 Text feature [patient] present in test data point [True]
85 Text feature [lung] present in test data point [True]
86 Text feature [lines] present in test data point [True]
88 Text feature [tki] present in test data point [True]
91 Text feature [ligand] present in test data point [True]
Out of the top  100  features  50 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [82]:

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

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

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

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


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

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

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

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 1.9047683350744757
for n_estimators = 10 and max depth =  3
Log Loss : 1.5282794467722431
for n_estimators = 10 and max depth =  5
Log Loss : 1.2884296952097405
for n_estimators = 10 and max depth =  10
Log Loss : 1.5839979157785737
for n_estimators = 50 and max depth =  2
Log Loss : 1.6093904747643943
for n_estimators = 50 and max depth =  3
Log Loss : 1.2229917974863622
for n_estimators = 50 and max depth =  5
Log Loss : 1.2664235336221208
for n_estimators = 50 and max depth =  10
Log Loss : 1.5282617834283754
for n_estimators = 100 and max depth =  2
Log Loss : 1.3980678922440628
for n_estimators = 100 and max depth =  3
Log Loss : 1.2390438302067757
for n_estimators = 100 and max depth =  5
Log Loss : 1.2660901561898177
for n_estimators = 100 and max depth =  10
Log Loss : 1.5213808751084754
for n_estimators = 200 and max depth =  2
Log Loss : 1.450176796233898
for n_estimators = 200 and max depth =  3
Log Loss : 1.2649948365796582
for n_estimators = 200 and max depth =  5
Log Loss : 1.2905635500276895
for n_estimators = 200 and max depth =  10
Log Loss : 1.5697998194202771
for n_estimators = 500 and max depth =  2
Log Loss : 1.467933894899738
for n_estimators = 500 and max depth =  3
Log Loss : 1.3357270388554403
for n_estimators = 500 and max depth =  5
Log Loss : 1.284612653924482
for n_estimators = 500 and max depth =  10
Log Loss : 1.6117144810242863
for n_estimators = 1000 and max depth =  2
Log Loss : 1.448913395057494
for n_estimators = 1000 and max depth =  3
Log Loss : 1.3270077350375011
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2695086417014618
for n_estimators = 1000 and max depth =  10
Log Loss : 1.571355068153723
For values of best alpha =  50 The train log loss is: 0.16488939696395333
For values of best alpha =  50 The cross validation log loss is: 1.2229917
974863622
For values of best alpha =  50 The test log loss is: 1.337165550087482
```

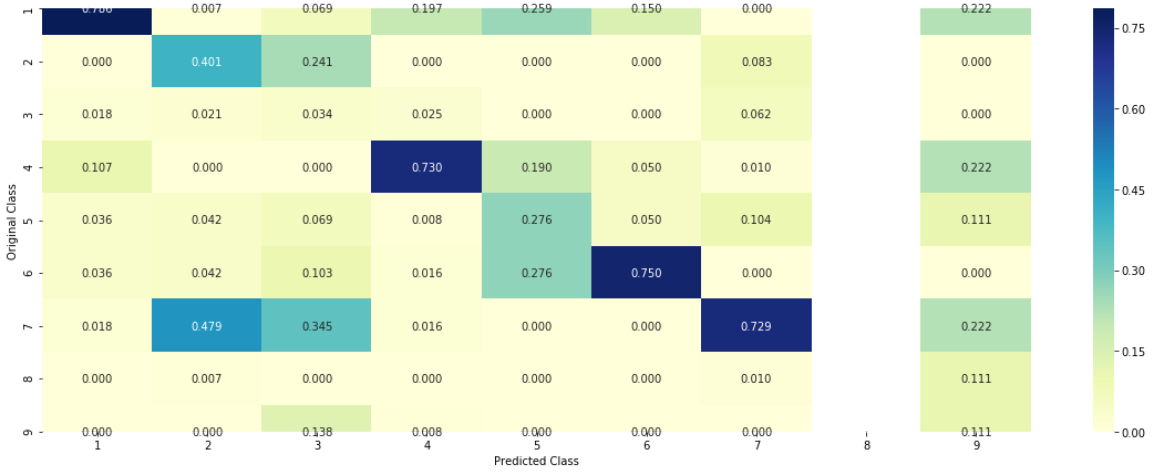
4.5.4. Testing model with best hyper parameters (Response Coding)

In [83]:

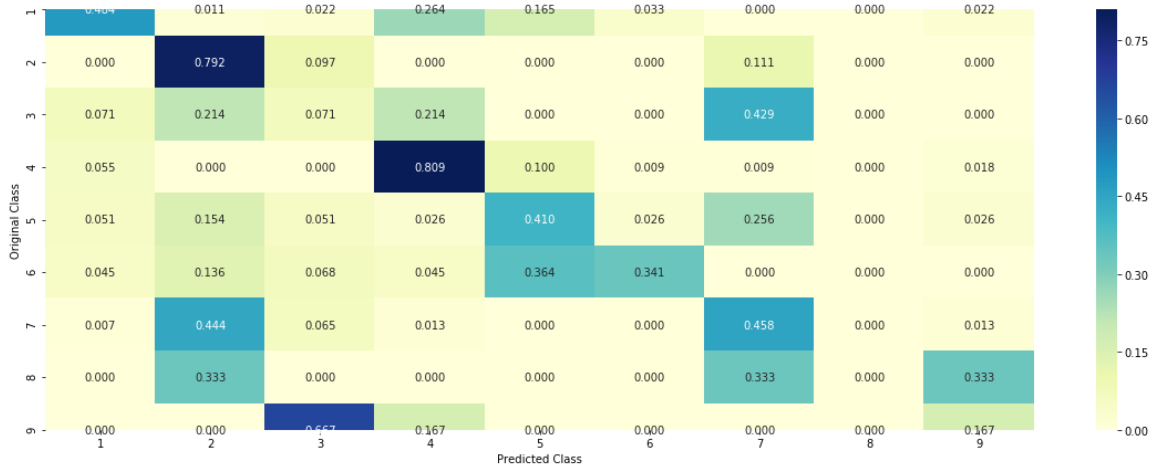
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.2229917974863622
Number of mis-classified points : 0.4492481203007519
-------------------- Confusion matrix --------------------
```


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




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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [86]:

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


test_point_index = 10
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseC
oding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0074 0.4934 0.116  0.0047 0.0207 0.013
0.3267 0.0128 0.0053]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [87]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2342 0.0133 0.0538 0.3816 0.1008 0.0453
0.0031 0.0531 0.1148]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [88]:

```
# 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: 1.13
Support vector machines : Log Loss: 1.71
Naive Bayes : Log Loss: 1.32
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.818
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.722
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.338
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.190
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.427
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.751
```

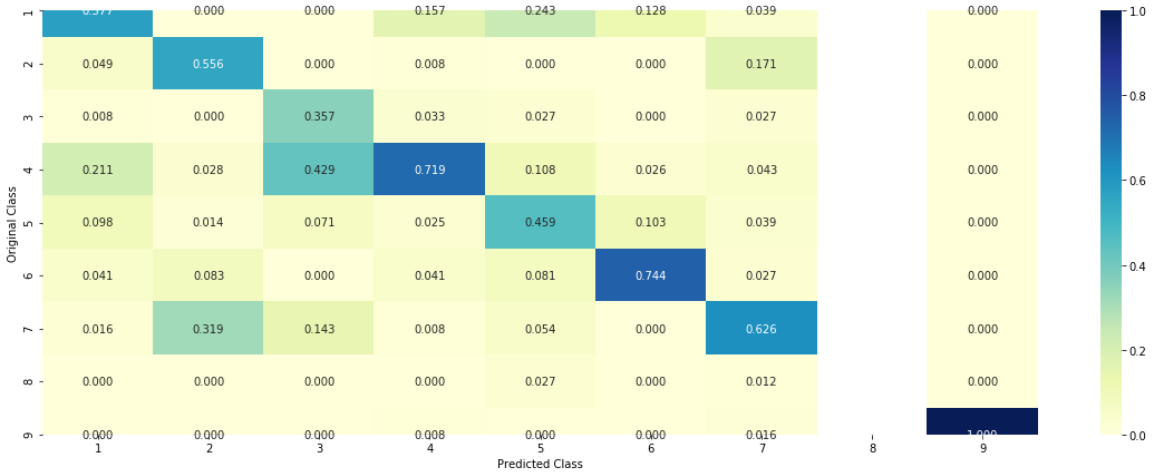
4.7.2 testing the model with the best hyper parameters

In [89]:

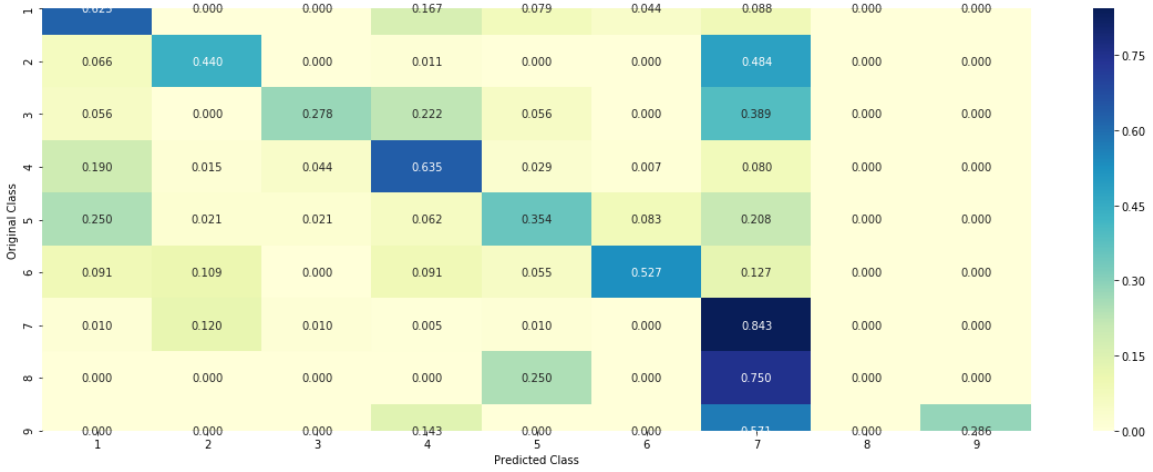
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
r, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.47492704409424413
Log loss (CV) on the stacking classifier : 1.1900690498539257
Log loss (test) on the stacking classifier : 1.2727832749339425
Number of missclassified point : 0.3804511278195489
-------------------- Confusion matrix --------------------
```


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




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



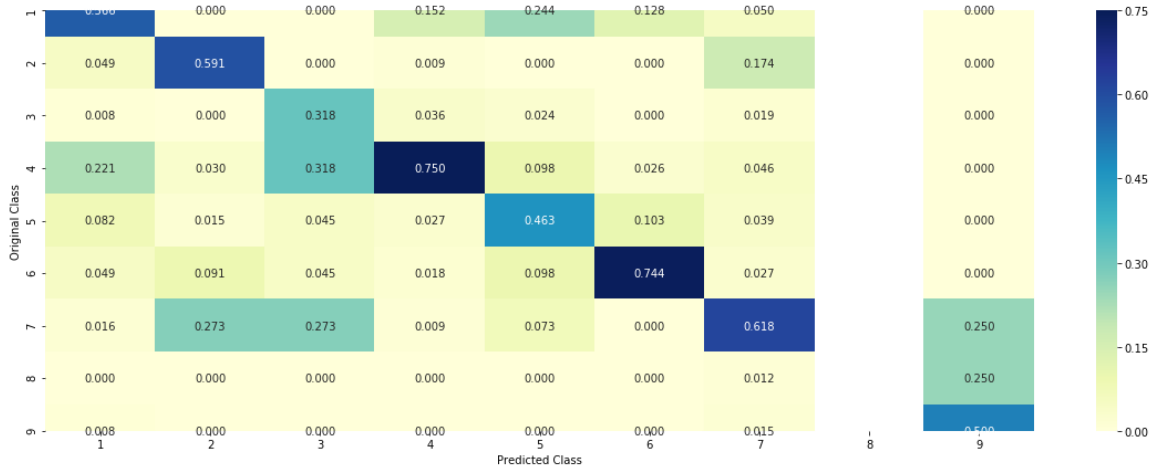
4.7.3 Maximum Voting classifier

In [90]:

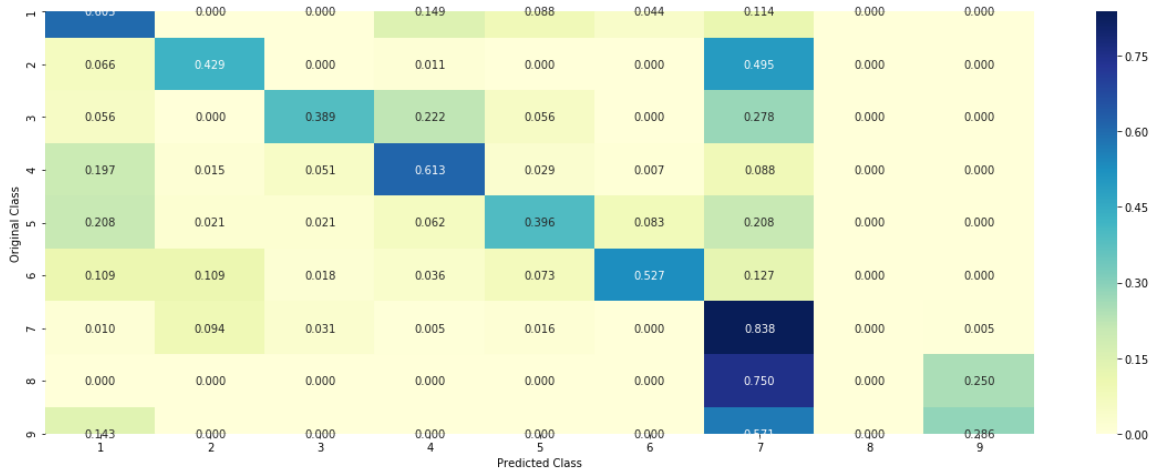
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log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

```
Log loss (train) on the VotingClassifier : 0.8330352045357216
Log loss (CV) on the VotingClassifier : 1.2074615316133255
Log loss (test) on the VotingClassifier : 1.2514757049038179
Number of missclassified point : 0.3849624060150376
-------------------- 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

APPLYING ALL MODELS WITH TFIDF FEATURES

In [56]:

```
def get_impfeature_names_tf(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

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

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

In [110]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train d
ata
tfidf_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_tf = tfidf_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features_tf= tfidf_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numb
er of features) vector
train_text_fea_counts_tf = train_text_feature_tf.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_tf = dict(zip(list(train_text_features_tf),train_text_fea_counts_tf))


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

```
Total number of unique words in train data : 53909
```

In [111]:

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

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

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

In [112]:

```
sorted_text_fea_dict_tf = dict(sorted(text_fea_dict_tf.items(), key=lambda x: x[1] , re
verse=True))
sorted_text_occur_tf = np.array(list(sorted_text_fea_dict.values()))
```

In [113]:

```
print(Counter(sorted_text_occur_tf))
```

```
Counter({3: 5566, 4: 3754, 6: 3035, 5: 2989, 7: 2208, 9: 1758, 8: 1737, 1
2: 1545, 10: 1371, 11: 1306, 13: 991, 16: 831, 14: 830, 15: 779, 18: 735,
20: 609, 17: 602, 22: 573, 24: 515, 21: 489, 19: 461, 26: 412, 28: 390, 4
5: 388, 23: 355, 25: 352, 27: 348, 32: 337, 30: 333, 48: 318, 29: 309, 36:
272, 33: 258, 35: 251, 34: 248, 40: 238, 31: 228, 39: 194, 44: 192, 38: 18
9, 37: 185, 46: 183, 42: 181, 41: 178, 50: 175, 47: 173, 60: 170, 54: 163,
49: 155, 51: 150, 58: 147, 55: 142, 43: 141, 57: 132, 52: 130, 72: 128, 5
6: 121, 63: 120, 53: 115, 64: 106, 66: 105, 73: 103, 59: 102, 65: 101, 70:
100, 61: 100, 69: 99, 68: 97, 62: 97, 78: 94, 77: 94, 90: 93, 74: 90, 76:
84, 67: 83, 87: 80, 71: 79, 91: 76, 75: 74, 92: 73, 84: 73, 81: 72, 96: 7
0, 88: 69, 82: 69, 93: 67, 108: 66, 99: 66, 80: 66, 97: 64, 95: 64, 98: 6
3, 94: 62, 85: 61, 104: 59, 102: 58, 79: 58, 86: 53, 83: 53, 117: 52, 105:
52, 101: 52, 135: 50, 144: 49, 103: 48, 89: 48, 130: 46, 120: 46, 111: 45,
106: 45, 100: 45, 142: 44, 116: 44, 112: 44, 125: 43, 115: 43, 138: 42, 13
2: 42, 124: 42, 110: 42, 121: 41, 114: 41, 149: 39, 127: 38, 147: 37, 141:
37, 131: 37, 170: 36, 119: 36, 113: 36, 153: 35, 140: 35, 133: 35, 123: 3
5, 122: 35, 128: 34, 109: 34, 154: 33, 134: 33, 129: 33, 137: 32, 136: 32,
107: 32, 160: 31, 150: 31, 139: 31, 126: 31, 118: 31, 159: 30, 157: 30, 16
8: 29, 176: 28, 173: 28, 169: 28, 155: 28, 151: 27, 228: 26, 201: 26, 200:
26, 158: 26, 152: 26, 195: 25, 180: 25, 179: 25, 166: 25, 165: 25, 156: 2
5, 145: 25, 143: 25, 302: 24, 193: 24, 167: 24, 234: 23, 212: 23, 210: 23,
198: 23, 196: 23, 192: 23, 177: 23, 163: 23, 148: 23, 213: 22, 209: 22, 20
2: 22, 146: 22, 215: 21, 205: 21, 204: 21, 191: 21, 189: 21, 185: 21, 181:
21, 172: 21, 162: 21, 270: 20, 232: 20, 225: 20, 186: 20, 178: 20, 175: 2
0, 164: 20, 161: 20, 235: 19, 227: 19, 226: 19, 224: 19, 222: 19, 211: 19,
183: 19, 260: 18, 243: 18, 240: 18, 236: 18, 230: 18, 217: 18, 199: 18, 18
8: 18, 184: 18, 174: 18, 171: 18, 257: 17, 245: 17, 241: 17, 229: 17, 223:
17, 208: 17, 206: 17, 203: 17, 295: 16, 283: 16, 267: 16, 259: 16, 238: 1
6, 221: 16, 214: 16, 194: 16, 366: 15, 324: 15, 317: 15, 299: 15, 290: 15,
286: 15, 255: 15, 254: 15, 253: 15, 246: 15, 242: 15, 239: 15, 220: 15, 20
7: 15, 190: 15, 333: 14, 322: 14, 272: 14, 268: 14, 266: 14, 261: 14, 256:
14, 218: 14, 216: 14, 197: 14, 182: 14, 453: 13, 403: 13, 334: 13, 275: 1
3, 273: 13, 269: 13, 263: 13, 258: 13, 252: 13, 495: 12, 384: 12, 382: 12,
361: 12, 360: 12, 335: 12, 315: 12, 314: 12, 297: 12, 288: 12, 279: 12, 27
8: 12, 271: 12, 264: 12, 435: 11, 365: 11, 350: 11, 345: 11, 338: 11, 328:
11, 321: 11, 313: 11, 306: 11, 300: 11, 296: 11, 289: 11, 285: 11, 277: 1
1, 274: 11, 251: 11, 231: 11, 219: 11, 187: 11, 543: 10, 450: 10, 422: 10,
418: 10, 417: 10, 410: 10, 389: 10, 378: 10, 373: 10, 368: 10, 357: 10, 35
6: 10, 344: 10, 343: 10, 332: 10, 320: 10, 319: 10, 305: 10, 291: 10, 284:
10, 282: 10, 280: 10, 250: 10, 248: 10, 247: 10, 587: 9, 522: 9, 514: 9, 5
12: 9, 509: 9, 440: 9, 409: 9, 408: 9, 394: 9, 374: 9, 372: 9, 339: 9, 32
6: 9, 325: 9, 309: 9, 308: 9, 301: 9, 298: 9, 294: 9, 293: 9, 292: 9, 287:
9, 265: 9, 262: 9, 237: 9, 927: 8, 787: 8, 658: 8, 589: 8, 584: 8, 554: 8,
540: 8, 525: 8, 521: 8, 478: 8, 461: 8, 458: 8, 447: 8, 426: 8, 397: 8, 38
8: 8, 371: 8, 354: 8, 353: 8, 340: 8, 310: 8, 307: 8, 304: 8, 281: 8, 249:
8, 244: 8, 233: 8, 811: 7, 716: 7, 702: 7, 624: 7, 623: 7, 579: 7, 571: 7,
560: 7, 559: 7, 538: 7, 528: 7, 508: 7, 473: 7, 463: 7, 460: 7, 456: 7, 44
8: 7, 445: 7, 432: 7, 406: 7, 404: 7, 398: 7, 392: 7, 383: 7, 375: 7, 362:
7, 359: 7, 348: 7, 342: 7, 337: 7, 336: 7, 323: 7, 318: 7, 316: 7, 311: 7,
1274: 6, 909: 6, 871: 6, 844: 6, 817: 6, 786: 6, 780: 6, 681: 6, 660: 6, 6
55: 6, 653: 6, 632: 6, 626: 6, 573: 6, 572: 6, 550: 6, 545: 6, 539: 6, 52
6: 6, 524: 6, 516: 6, 510: 6, 503: 6, 502: 6, 500: 6, 497: 6, 494: 6, 487:
6, 484: 6, 474: 6, 471: 6, 468: 6, 466: 6, 457: 6, 454: 6, 449: 6, 446: 6,
442: 6, 441: 6, 437: 6, 433: 6, 431: 6, 429: 6, 425: 6, 416: 6, 413: 6, 40
2: 6, 391: 6, 380: 6, 376: 6, 364: 6, 352: 6, 349: 6, 347: 6, 346: 6, 341:
6, 330: 6, 327: 6, 303: 6, 1224: 5, 1114: 5, 937: 5, 853: 5, 799: 5, 781:
5, 776: 5, 774: 5, 769: 5, 760: 5, 748: 5, 747: 5, 740: 5, 705: 5, 651: 5,
650: 5, 643: 5, 619: 5, 618: 5, 613: 5, 604: 5, 581: 5, 578: 5, 553: 5, 54
8: 5, 534: 5, 531: 5, 511: 5, 507: 5, 492: 5, 480: 5, 477: 5, 475: 5, 470:
5, 464: 5, 462: 5, 452: 5, 443: 5, 428: 5, 415: 5, 414: 5, 412: 5, 405: 5,
401: 5, 396: 5, 395: 5, 390: 5, 387: 5, 386: 5, 381: 5, 379: 5, 370: 5, 36
```

```
9: 5, 358: 5, 355: 5, 331: 5, 312: 5, 276: 5, 2115: 4, 1926: 4, 1495: 4, 1
355: 4, 1319: 4, 1263: 4, 1261: 4, 1244: 4, 1234: 4, 1233: 4, 1180: 4, 113
3: 4, 1046: 4, 1036: 4, 1026: 4, 1009: 4, 1001: 4, 993: 4, 981: 4, 972: 4,
964: 4, 952: 4, 945: 4, 941: 4, 936: 4, 918: 4, 904: 4, 902: 4, 880: 4, 87
7: 4, 829: 4, 824: 4, 822: 4, 815: 4, 814: 4, 809: 4, 808: 4, 807: 4, 800:
4, 789: 4, 779: 4, 772: 4, 768: 4, 767: 4, 761: 4, 750: 4, 742: 4, 741: 4,
734: 4, 729: 4, 726: 4, 713: 4, 711: 4, 710: 4, 699: 4, 696: 4, 695: 4, 69
1: 4, 690: 4, 680: 4, 677: 4, 676: 4, 673: 4, 662: 4, 659: 4, 654: 4, 649:
4, 647: 4, 644: 4, 642: 4, 635: 4, 633: 4, 628: 4, 625: 4, 610: 4, 598: 4,
594: 4, 592: 4, 591: 4, 583: 4, 580: 4, 570: 4, 568: 4, 567: 4, 562: 4, 56
1: 4, 558: 4, 557: 4, 544: 4, 542: 4, 535: 4, 532: 4, 520: 4, 517: 4, 505:
4, 504: 4, 501: 4, 496: 4, 489: 4, 488: 4, 483: 4, 482: 4, 472: 4, 469: 4,
467: 4, 455: 4, 451: 4, 444: 4, 438: 4, 427: 4, 423: 4, 407: 4, 400: 4, 37
7: 4, 367: 4, 363: 4, 351: 4, 329: 4, 3578: 3, 3449: 3, 3161: 3, 2570: 3,
2542: 3, 2538: 3, 2533: 3, 2400: 3, 2378: 3, 2256: 3, 2135: 3, 2132: 3, 20
82: 3, 1857: 3, 1809: 3, 1800: 3, 1788: 3, 1752: 3, 1699: 3, 1667: 3, 165
7: 3, 1655: 3, 1651: 3, 1639: 3, 1605: 3, 1599: 3, 1595: 3, 1589: 3, 1552:
3, 1541: 3, 1523: 3, 1521: 3, 1516: 3, 1503: 3, 1502: 3, 1492: 3, 1482: 3,
1463: 3, 1448: 3, 1384: 3, 1375: 3, 1365: 3, 1360: 3, 1338: 3, 1305: 3, 12
97: 3, 1291: 3, 1286: 3, 1284: 3, 1282: 3, 1267: 3, 1266: 3, 1258: 3, 123
1: 3, 1214: 3, 1203: 3, 1166: 3, 1160: 3, 1159: 3, 1142: 3, 1135: 3, 1132:
3, 1131: 3, 1121: 3, 1115: 3, 1113: 3, 1110: 3, 1109: 3, 1099: 3, 1092: 3,
1091: 3, 1087: 3, 1072: 3, 1066: 3, 1063: 3, 1031: 3, 1025: 3, 1002: 3, 99
7: 3, 992: 3, 991: 3, 987: 3, 971: 3, 968: 3, 959: 3, 955: 3, 942: 3, 939:
3, 938: 3, 934: 3, 930: 3, 920: 3, 919: 3, 895: 3, 894: 3, 892: 3, 888: 3,
886: 3, 881: 3, 869: 3, 858: 3, 840: 3, 838: 3, 837: 3, 836: 3, 832: 3, 83
1: 3, 825: 3, 812: 3, 796: 3, 782: 3, 777: 3, 755: 3, 752: 3, 749: 3, 745:
3, 744: 3, 743: 3, 733: 3, 731: 3, 724: 3, 721: 3, 717: 3, 712: 3, 708: 3,
707: 3, 698: 3, 687: 3, 684: 3, 679: 3, 675: 3, 674: 3, 669: 3, 666: 3, 66
4: 3, 657: 3, 645: 3, 638: 3, 631: 3, 630: 3, 621: 3, 617: 3, 615: 3, 612:
3, 609: 3, 607: 3, 605: 3, 603: 3, 602: 3, 599: 3, 596: 3, 590: 3, 582: 3,
577: 3, 576: 3, 574: 3, 569: 3, 563: 3, 555: 3, 546: 3, 541: 3, 537: 3, 53
0: 3, 527: 3, 518: 3, 513: 3, 499: 3, 498: 3, 490: 3, 481: 3, 479: 3, 465:
3, 439: 3, 436: 3, 434: 3, 424: 3, 421: 3, 420: 3, 419: 3, 411: 3, 385: 3,
15456: 2, 12636: 2, 12460: 2, 7004: 2, 6972: 2, 6666: 2, 6302: 2, 6275: 2,
5146: 2, 4979: 2, 4892: 2, 4656: 2, 4247: 2, 4232: 2, 4223: 2, 4200: 2, 39
77: 2, 3780: 2, 3759: 2, 3663: 2, 3626: 2, 3618: 2, 3570: 2, 3560: 2, 352
2: 2, 3505: 2, 3479: 2, 3477: 2, 3459: 2, 3422: 2, 3399: 2, 3361: 2, 3348:
2, 3318: 2, 3315: 2, 3299: 2, 3282: 2, 3192: 2, 3165: 2, 3119: 2, 3102: 2,
3092: 2, 3080: 2, 3014: 2, 3011: 2, 2886: 2, 2883: 2, 2868: 2, 2822: 2, 28
04: 2, 2791: 2, 2698: 2, 2691: 2, 2639: 2, 2620: 2, 2572: 2, 2561: 2, 255
0: 2, 2541: 2, 2506: 2, 2496: 2, 2484: 2, 2440: 2, 2416: 2, 2408: 2, 2277:
2, 2266: 2, 2260: 2, 2258: 2, 2257: 2, 2188: 2, 2145: 2, 2129: 2, 2100: 2,
2098: 2, 2063: 2, 2059: 2, 2014: 2, 2006: 2, 1988: 2, 1974: 2, 1966: 2, 19
54: 2, 1952: 2, 1948: 2, 1944: 2, 1943: 2, 1941: 2, 1931: 2, 1921: 2, 191
3: 2, 1912: 2, 1901: 2, 1900: 2, 1896: 2, 1872: 2, 1864: 2, 1847: 2, 1845:
2, 1833: 2, 1831: 2, 1812: 2, 1806: 2, 1799: 2, 1796: 2, 1787: 2, 1775: 2,
1770: 2, 1742: 2, 1741: 2, 1740: 2, 1735: 2, 1729: 2, 1707: 2, 1704: 2, 16
95: 2, 1690: 2, 1687: 2, 1677: 2, 1658: 2, 1656: 2, 1650: 2, 1649: 2, 164
3: 2, 1637: 2, 1629: 2, 1614: 2, 1612: 2, 1610: 2, 1602: 2, 1598: 2, 1597:
2, 1561: 2, 1560: 2, 1558: 2, 1557: 2, 1555: 2, 1549: 2, 1542: 2, 1537: 2,
1530: 2, 1526: 2, 1525: 2, 1514: 2, 1513: 2, 1511: 2, 1496: 2, 1466: 2, 14
58: 2, 1446: 2, 1431: 2, 1415: 2, 1409: 2, 1408: 2, 1401: 2, 1395: 2, 139
0: 2, 1378: 2, 1373: 2, 1372: 2, 1371: 2, 1366: 2, 1364: 2, 1362: 2, 1352:
2, 1349: 2, 1345: 2, 1343: 2, 1342: 2, 1336: 2, 1334: 2, 1333: 2, 1330: 2,
1325: 2, 1321: 2, 1316: 2, 1314: 2, 1312: 2, 1296: 2, 1293: 2, 1288: 2, 12
56: 2, 1254: 2, 1245: 2, 1243: 2, 1242: 2, 1240: 2, 1237: 2, 1236: 2, 123
2: 2, 1228: 2, 1227: 2, 1223: 2, 1216: 2, 1209: 2, 1206: 2, 1201: 2, 1198:
2, 1197: 2, 1189: 2, 1185: 2, 1178: 2, 1177: 2, 1173: 2, 1170: 2, 1168: 2,
1157: 2, 1155: 2, 1153: 2, 1149: 2, 1148: 2, 1146: 2, 1140: 2, 1136: 2, 11
28: 2, 1127: 2, 1126: 2, 1125: 2, 1124: 2, 1122: 2, 1120: 2, 1111: 2, 110
```

```
0: 2, 1098: 2, 1090: 2, 1088: 2, 1086: 2, 1083: 2, 1075: 2, 1074: 2, 1073:
2, 1062: 2, 1060: 2, 1052: 2, 1051: 2, 1038: 2, 1037: 2, 1033: 2, 1028: 2,
1027: 2, 1023: 2, 1021: 2, 1019: 2, 1017: 2, 1015: 2, 1006: 2, 1005: 2, 10
04: 2, 1003: 2, 994: 2, 990: 2, 984: 2, 978: 2, 977: 2, 975: 2, 974: 2, 96
6: 2, 963: 2, 962: 2, 961: 2, 958: 2, 956: 2, 951: 2, 949: 2, 948: 2, 947:
2, 944: 2, 940: 2, 935: 2, 933: 2, 932: 2, 926: 2, 921: 2, 916: 2, 906: 2,
903: 2, 901: 2, 899: 2, 893: 2, 891: 2, 890: 2, 889: 2, 887: 2, 883: 2, 87
9: 2, 878: 2, 875: 2, 872: 2, 868: 2, 866: 2, 864: 2, 861: 2, 856: 2, 855:
2, 854: 2, 852: 2, 851: 2, 850: 2, 849: 2, 841: 2, 834: 2, 833: 2, 830: 2,
827: 2, 826: 2, 821: 2, 818: 2, 805: 2, 803: 2, 801: 2, 797: 2, 793: 2, 78
8: 2, 783: 2, 775: 2, 771: 2, 766: 2, 765: 2, 764: 2, 762: 2, 756: 2, 754:
2, 753: 2, 739: 2, 737: 2, 735: 2, 732: 2, 727: 2, 722: 2, 719: 2, 718: 2,
715: 2, 709: 2, 706: 2, 704: 2, 701: 2, 700: 2, 689: 2, 688: 2, 683: 2, 68
2: 2, 672: 2, 670: 2, 661: 2, 656: 2, 648: 2, 646: 2, 641: 2, 639: 2, 634:
2, 629: 2, 627: 2, 620: 2, 608: 2, 601: 2, 600: 2, 597: 2, 595: 2, 588: 2,
586: 2, 585: 2, 575: 2, 566: 2, 565: 2, 556: 2, 549: 2, 536: 2, 533: 2, 52
9: 2, 519: 2, 493: 2, 485: 2, 476: 2, 459: 2, 430: 2, 399: 2, 393: 2, 1511
98: 1, 120077: 1, 80659: 1, 66595: 1, 66142: 1, 65482: 1, 65308: 1, 63914:
1, 63239: 1, 54736: 1, 54049: 1, 50764: 1, 49681: 1, 46849: 1, 46322: 1, 4
4442: 1, 43873: 1, 42297: 1, 42208: 1, 41976: 1, 40824: 1, 40767: 1, 4031
4: 1, 39473: 1, 38908: 1, 37942: 1, 36681: 1, 36231: 1, 36051: 1, 34361:
1, 34245: 1, 33741: 1, 33115: 1, 32779: 1, 31748: 1, 31357: 1, 29622: 1, 2
8166: 1, 27550: 1, 26128: 1, 25965: 1, 25882: 1, 25820: 1, 25507: 1, 2523
9: 1, 24924: 1, 24647: 1, 24254: 1, 24157: 1, 24076: 1, 23822: 1, 23790:
1, 23112: 1, 22393: 1, 22155: 1, 22151: 1, 21953: 1, 21590: 1, 21451: 1, 2
1065: 1, 20786: 1, 20490: 1, 20434: 1, 20077: 1, 20064: 1, 19612: 1, 1906
5: 1, 19059: 1, 18993: 1, 18719: 1, 18589: 1, 18570: 1, 18568: 1, 18545:
1, 18478: 1, 18254: 1, 18244: 1, 18152: 1, 18005: 1, 17950: 1, 17802: 1, 1
7747: 1, 17605: 1, 17518: 1, 17475: 1, 17468: 1, 17287: 1, 17263: 1, 1696
0: 1, 16943: 1, 16914: 1, 16898: 1, 16782: 1, 16699: 1, 16487: 1, 16244:
1, 16115: 1, 16071: 1, 16038: 1, 15967: 1, 15686: 1, 15684: 1, 15545: 1, 1
5538: 1, 15318: 1, 15257: 1, 15206: 1, 15167: 1, 15140: 1, 14988: 1, 1474
9: 1, 14745: 1, 14620: 1, 14604: 1, 14442: 1, 14158: 1, 14107: 1, 14059:
1, 14046: 1, 14027: 1, 13935: 1, 13841: 1, 13766: 1, 13582: 1, 13444: 1, 1
3359: 1, 13252: 1, 13236: 1, 13231: 1, 13176: 1, 13023: 1, 12985: 1, 1291
2: 1, 12861: 1, 12827: 1, 12790: 1, 12712: 1, 12660: 1, 12652: 1, 12641:
1, 12610: 1, 12568: 1, 12437: 1, 12387: 1, 12366: 1, 12308: 1, 12306: 1, 1
2301: 1, 12221: 1, 12197: 1, 12158: 1, 12114: 1, 12096: 1, 12061: 1, 1202
7: 1, 12025: 1, 11993: 1, 11955: 1, 11950: 1, 11912: 1, 11889: 1, 11878:
1, 11786: 1, 11783: 1, 11753: 1, 11721: 1, 11630: 1, 11616: 1, 11559: 1, 1
1544: 1, 11540: 1, 11436: 1, 11404: 1, 11315: 1, 11117: 1, 11102: 1, 1108
0: 1, 10901: 1, 10833: 1, 10779: 1, 10744: 1, 10683: 1, 10579: 1, 10546:
1, 10523: 1, 10442: 1, 10314: 1, 10274: 1, 10246: 1, 10115: 1, 10047: 1, 1
0021: 1, 10015: 1, 9991: 1, 9985: 1, 9971: 1, 9959: 1, 9951: 1, 9916: 1, 9
861: 1, 9829: 1, 9807: 1, 9731: 1, 9690: 1, 9653: 1, 9645: 1, 9607: 1, 960
5: 1, 9588: 1, 9585: 1, 9540: 1, 9534: 1, 9529: 1, 9435: 1, 9410: 1, 9402:
1, 9376: 1, 9274: 1, 9263: 1, 9241: 1, 9217: 1, 9210: 1, 9186: 1, 9132: 1,
9116: 1, 9057: 1, 9032: 1, 9026: 1, 9025: 1, 9011: 1, 9001: 1, 8985: 1, 89
45: 1, 8889: 1, 8888: 1, 8870: 1, 8869: 1, 8787: 1, 8775: 1, 8747: 1, 869
2: 1, 8573: 1, 8543: 1, 8503: 1, 8479: 1, 8434: 1, 8402: 1, 8348: 1, 8320:
1, 8300: 1, 8295: 1, 8294: 1, 8268: 1, 8234: 1, 8230: 1, 8218: 1, 8196: 1,
8166: 1, 8160: 1, 8142: 1, 8128: 1, 8122: 1, 8097: 1, 8084: 1, 8082: 1, 80
79: 1, 8004: 1, 7978: 1, 7974: 1, 7964: 1, 7963: 1, 7962: 1, 7945: 1, 792
0: 1, 7894: 1, 7877: 1, 7855: 1, 7852: 1, 7806: 1, 7803: 1, 7796: 1, 7768:
1, 7727: 1, 7628: 1, 7597: 1, 7592: 1, 7572: 1, 7544: 1, 7541: 1, 7520: 1,
7519: 1, 7510: 1, 7495: 1, 7421: 1, 7384: 1, 7363: 1, 7318: 1, 7312: 1, 72
70: 1, 7259: 1, 7258: 1, 7248: 1, 7246: 1, 7236: 1, 7224: 1, 7221: 1, 721
4: 1, 7199: 1, 7185: 1, 7173: 1, 7168: 1, 7159: 1, 7137: 1, 7124: 1, 7109:
1, 7081: 1, 7073: 1, 7070: 1, 7056: 1, 7048: 1, 7042: 1, 7001: 1, 7000: 1,
6976: 1, 6956: 1, 6947: 1, 6935: 1, 6899: 1, 6877: 1, 6853: 1, 6825: 1, 68
24: 1, 6817: 1, 6813: 1, 6777: 1, 6728: 1, 6724: 1, 6717: 1, 6700: 1, 669
```

```
6: 1, 6670: 1, 6661: 1, 6620: 1, 6611: 1, 6610: 1, 6589: 1, 6578: 1, 6572:
1, 6560: 1, 6546: 1, 6529: 1, 6526: 1, 6521: 1, 6489: 1, 6437: 1, 6408: 1,
6394: 1, 6373: 1, 6360: 1, 6333: 1, 6323: 1, 6317: 1, 6304: 1, 6301: 1, 62
95: 1, 6291: 1, 6264: 1, 6255: 1, 6250: 1, 6249: 1, 6241: 1, 6172: 1, 616
4: 1, 6154: 1, 6119: 1, 6116: 1, 6100: 1, 6083: 1, 6073: 1, 6067: 1, 6011:
1, 6010: 1, 6008: 1, 6004: 1, 5992: 1, 5979: 1, 5968: 1, 5958: 1, 5950: 1,
5939: 1, 5928: 1, 5918: 1, 5901: 1, 5882: 1, 5881: 1, 5862: 1, 5859: 1, 58
13: 1, 5806: 1, 5796: 1, 5779: 1, 5777: 1, 5772: 1, 5767: 1, 5762: 1, 575
8: 1, 5748: 1, 5747: 1, 5745: 1, 5740: 1, 5732: 1, 5730: 1, 5722: 1, 5670:
1, 5656: 1, 5650: 1, 5649: 1, 5641: 1, 5628: 1, 5592: 1, 5590: 1, 5551: 1,
5533: 1, 5529: 1, 5525: 1, 5522: 1, 5501: 1, 5494: 1, 5485: 1, 5481: 1, 54
73: 1, 5463: 1, 5455: 1, 5450: 1, 5437: 1, 5431: 1, 5423: 1, 5420: 1, 541
4: 1, 5408: 1, 5406: 1, 5403: 1, 5344: 1, 5332: 1, 5275: 1, 5259: 1, 5241:
1, 5213: 1, 5193: 1, 5169: 1, 5165: 1, 5157: 1, 5145: 1, 5143: 1, 5140: 1,
5138: 1, 5135: 1, 5127: 1, 5106: 1, 5105: 1, 5100: 1, 5069: 1, 5068: 1, 50
64: 1, 5061: 1, 5022: 1, 5012: 1, 5010: 1, 4981: 1, 4971: 1, 4968: 1, 495
2: 1, 4949: 1, 4935: 1, 4926: 1, 4918: 1, 4917: 1, 4913: 1, 4903: 1, 4898:
1, 4897: 1, 4895: 1, 4875: 1, 4870: 1, 4868: 1, 4856: 1, 4850: 1, 4841: 1,
4840: 1, 4837: 1, 4835: 1, 4822: 1, 4816: 1, 4803: 1, 4797: 1, 4794: 1, 47
88: 1, 4785: 1, 4779: 1, 4771: 1, 4769: 1, 4765: 1, 4760: 1, 4746: 1, 474
3: 1, 4741: 1, 4734: 1, 4718: 1, 4711: 1, 4683: 1, 4647: 1, 4642: 1, 4609:
1, 4587: 1, 4585: 1, 4582: 1, 4575: 1, 4573: 1, 4572: 1, 4562: 1, 4560: 1,
4553: 1, 4530: 1, 4517: 1, 4494: 1, 4483: 1, 4441: 1, 4436: 1, 4431: 1, 44
18: 1, 4417: 1, 4416: 1, 4415: 1, 4401: 1, 4400: 1, 4399: 1, 4392: 1, 438
5: 1, 4371: 1, 4370: 1, 4364: 1, 4362: 1, 4345: 1, 4344: 1, 4337: 1, 4328:
1, 4327: 1, 4322: 1, 4321: 1, 4301: 1, 4284: 1, 4278: 1, 4275: 1, 4273: 1,
4261: 1, 4258: 1, 4233: 1, 4224: 1, 4220: 1, 4207: 1, 4205: 1, 4199: 1, 41
93: 1, 4190: 1, 4189: 1, 4181: 1, 4178: 1, 4177: 1, 4172: 1, 4170: 1, 415
4: 1, 4152: 1, 4151: 1, 4146: 1, 4145: 1, 4140: 1, 4132: 1, 4131: 1, 4128:
1, 4125: 1, 4109: 1, 4106: 1, 4099: 1, 4092: 1, 4089: 1, 4086: 1, 4076: 1,
4074: 1, 4069: 1, 4038: 1, 4035: 1, 4025: 1, 4024: 1, 4021: 1, 4018: 1, 40
16: 1, 4013: 1, 4003: 1, 4000: 1, 3996: 1, 3988: 1, 3987: 1, 3986: 1, 398
5: 1, 3970: 1, 3964: 1, 3961: 1, 3959: 1, 3958: 1, 3957: 1, 3949: 1, 3923:
1, 3921: 1, 3919: 1, 3918: 1, 3911: 1, 3905: 1, 3904: 1, 3894: 1, 3891: 1,
3878: 1, 3873: 1, 3869: 1, 3867: 1, 3866: 1, 3865: 1, 3862: 1, 3858: 1, 38
45: 1, 3839: 1, 3835: 1, 3829: 1, 3823: 1, 3822: 1, 3807: 1, 3796: 1, 379
1: 1, 3779: 1, 3773: 1, 3766: 1, 3763: 1, 3760: 1, 3756: 1, 3744: 1, 3734:
1, 3731: 1, 3718: 1, 3713: 1, 3712: 1, 3711: 1, 3703: 1, 3701: 1, 3681: 1,
3680: 1, 3675: 1, 3669: 1, 3653: 1, 3646: 1, 3641: 1, 3637: 1, 3635: 1, 36
33: 1, 3627: 1, 3617: 1, 3610: 1, 3609: 1, 3606: 1, 3595: 1, 3590: 1, 357
7: 1, 3559: 1, 3558: 1, 3554: 1, 3552: 1, 3550: 1, 3549: 1, 3542: 1, 3541:
1, 3539: 1, 3534: 1, 3531: 1, 3528: 1, 3503: 1, 3501: 1, 3499: 1, 3496: 1,
3493: 1, 3484: 1, 3471: 1, 3469: 1, 3466: 1, 3465: 1, 3455: 1, 3446: 1, 34
45: 1, 3444: 1, 3443: 1, 3437: 1, 3427: 1, 3406: 1, 3392: 1, 3390: 1, 338
9: 1, 3388: 1, 3384: 1, 3375: 1, 3371: 1, 3366: 1, 3364: 1, 3360: 1, 3351:
1, 3349: 1, 3347: 1, 3346: 1, 3337: 1, 3335: 1, 3324: 1, 3323: 1, 3321: 1,
3316: 1, 3313: 1, 3298: 1, 3295: 1, 3291: 1, 3288: 1, 3285: 1, 3283: 1, 32
80: 1, 3279: 1, 3277: 1, 3273: 1, 3269: 1, 3249: 1, 3248: 1, 3246: 1, 323
7: 1, 3236: 1, 3233: 1, 3231: 1, 3229: 1, 3225: 1, 3224: 1, 3218: 1, 3211:
1, 3203: 1, 3190: 1, 3176: 1, 3174: 1, 3162: 1, 3160: 1, 3152: 1, 3146: 1,
3145: 1, 3143: 1, 3139: 1, 3132: 1, 3129: 1, 3118: 1, 3108: 1, 3107: 1, 31
06: 1, 3100: 1, 3096: 1, 3088: 1, 3079: 1, 3078: 1, 3055: 1, 3053: 1, 305
0: 1, 3042: 1, 3034: 1, 3032: 1, 3026: 1, 3024: 1, 3021: 1, 3020: 1, 3012:
1, 3009: 1, 3008: 1, 3007: 1, 3006: 1, 3003: 1, 3001: 1, 3000: 1, 2997: 1,
2990: 1, 2988: 1, 2987: 1, 2986: 1, 2981: 1, 2978: 1, 2975: 1, 2972: 1, 29
71: 1, 2969: 1, 2967: 1, 2959: 1, 2955: 1, 2949: 1, 2943: 1, 2939: 1, 293
5: 1, 2931: 1, 2925: 1, 2921: 1, 2918: 1, 2912: 1, 2910: 1, 2900: 1, 2899:
1, 2882: 1, 2879: 1, 2873: 1, 2872: 1, 2862: 1, 2860: 1, 2857: 1, 2849: 1,
2847: 1, 2845: 1, 2838: 1, 2831: 1, 2830: 1, 2829: 1, 2827: 1, 2818: 1, 28
10: 1, 2807: 1, 2797: 1, 2790: 1, 2789: 1, 2781: 1, 2775: 1, 2772: 1, 276
7: 1, 2756: 1, 2751: 1, 2745: 1, 2735: 1, 2722: 1, 2715: 1, 2713: 1, 2712:
```

```
1, 2711: 1, 2708: 1, 2700: 1, 2699: 1, 2695: 1, 2693: 1, 2687: 1, 2683: 1,
2682: 1, 2675: 1, 2671: 1, 2669: 1, 2666: 1, 2665: 1, 2664: 1, 2663: 1, 26
58: 1, 2656: 1, 2655: 1, 2654: 1, 2653: 1, 2648: 1, 2645: 1, 2644: 1, 264
1: 1, 2637: 1, 2636: 1, 2635: 1, 2624: 1, 2616: 1, 2615: 1, 2608: 1, 2599:
1, 2594: 1, 2593: 1, 2591: 1, 2590: 1, 2589: 1, 2584: 1, 2577: 1, 2576: 1,
2573: 1, 2560: 1, 2559: 1, 2556: 1, 2554: 1, 2548: 1, 2544: 1, 2543: 1, 25
39: 1, 2534: 1, 2527: 1, 2519: 1, 2518: 1, 2517: 1, 2511: 1, 2508: 1, 250
3: 1, 2500: 1, 2499: 1, 2498: 1, 2493: 1, 2492: 1, 2489: 1, 2485: 1, 2480:
1, 2479: 1, 2477: 1, 2475: 1, 2464: 1, 2463: 1, 2462: 1, 2461: 1, 2459: 1,
2455: 1, 2451: 1, 2450: 1, 2449: 1, 2447: 1, 2445: 1, 2439: 1, 2429: 1, 24
25: 1, 2420: 1, 2419: 1, 2415: 1, 2414: 1, 2410: 1, 2409: 1, 2405: 1, 240
2: 1, 2396: 1, 2393: 1, 2392: 1, 2385: 1, 2383: 1, 2377: 1, 2373: 1, 2368:
1, 2366: 1, 2365: 1, 2362: 1, 2356: 1, 2350: 1, 2339: 1, 2334: 1, 2333: 1,
2332: 1, 2329: 1, 2328: 1, 2327: 1, 2325: 1, 2320: 1, 2319: 1, 2311: 1, 23
08: 1, 2303: 1, 2300: 1, 2299: 1, 2298: 1, 2289: 1, 2288: 1, 2287: 1, 228
6: 1, 2284: 1, 2283: 1, 2281: 1, 2280: 1, 2279: 1, 2276: 1, 2275: 1, 2274:
1, 2273: 1, 2271: 1, 2268: 1, 2261: 1, 2259: 1, 2253: 1, 2251: 1, 2250: 1,
2247: 1, 2243: 1, 2240: 1, 2238: 1, 2235: 1, 2234: 1, 2233: 1, 2227: 1, 22
25: 1, 2223: 1, 2216: 1, 2213: 1, 2207: 1, 2206: 1, 2205: 1, 2200: 1, 219
9: 1, 2196: 1, 2190: 1, 2187: 1, 2186: 1, 2185: 1, 2184: 1, 2180: 1, 2179:
1, 2177: 1, 2173: 1, 2168: 1, 2167: 1, 2166: 1, 2165: 1, 2164: 1, 2163: 1,
2161: 1, 2160: 1, 2157: 1, 2156: 1, 2155: 1, 2153: 1, 2151: 1, 2149: 1, 21
48: 1, 2144: 1, 2140: 1, 2139: 1, 2137: 1, 2133: 1, 2127: 1, 2124: 1, 212
3: 1, 2117: 1, 2114: 1, 2109: 1, 2108: 1, 2104: 1, 2103: 1, 2099: 1, 2095:
1, 2094: 1, 2091: 1, 2089: 1, 2087: 1, 2083: 1, 2076: 1, 2072: 1, 2071: 1,
2066: 1, 2062: 1, 2060: 1, 2058: 1, 2053: 1, 2052: 1, 2050: 1, 2048: 1, 20
47: 1, 2044: 1, 2041: 1, 2035: 1, 2033: 1, 2032: 1, 2031: 1, 2030: 1, 202
6: 1, 2024: 1, 2023: 1, 2015: 1, 2011: 1, 2010: 1, 2009: 1, 2007: 1, 2004:
1, 2001: 1, 1996: 1, 1992: 1, 1987: 1, 1982: 1, 1977: 1, 1976: 1, 1975: 1,
1973: 1, 1971: 1, 1963: 1, 1961: 1, 1960: 1, 1959: 1, 1956: 1, 1955: 1, 19
51: 1, 1946: 1, 1942: 1, 1930: 1, 1929: 1, 1927: 1, 1925: 1, 1922: 1, 192
0: 1, 1919: 1, 1918: 1, 1917: 1, 1914: 1, 1908: 1, 1907: 1, 1906: 1, 1903:
1, 1902: 1, 1898: 1, 1897: 1, 1893: 1, 1892: 1, 1890: 1, 1889: 1, 1885: 1,
1881: 1, 1878: 1, 1876: 1, 1874: 1, 1871: 1, 1870: 1, 1866: 1, 1863: 1, 18
62: 1, 1860: 1, 1859: 1, 1854: 1, 1852: 1, 1851: 1, 1850: 1, 1849: 1, 184
8: 1, 1846: 1, 1842: 1, 1840: 1, 1838: 1, 1837: 1, 1832: 1, 1829: 1, 1825:
1, 1821: 1, 1820: 1, 1819: 1, 1818: 1, 1817: 1, 1816: 1, 1813: 1, 1811: 1,
1803: 1, 1802: 1, 1801: 1, 1798: 1, 1793: 1, 1789: 1, 1785: 1, 1781: 1, 17
80: 1, 1779: 1, 1778: 1, 1776: 1, 1773: 1, 1767: 1, 1766: 1, 1765: 1, 176
3: 1, 1762: 1, 1760: 1, 1749: 1, 1748: 1, 1746: 1, 1745: 1, 1744: 1, 1743:
1, 1738: 1, 1730: 1, 1726: 1, 1718: 1, 1717: 1, 1716: 1, 1714: 1, 1712: 1,
1711: 1, 1708: 1, 1706: 1, 1705: 1, 1702: 1, 1700: 1, 1689: 1, 1685: 1, 16
84: 1, 1683: 1, 1682: 1, 1681: 1, 1679: 1, 1678: 1, 1675: 1, 1673: 1, 167
2: 1, 1670: 1, 1669: 1, 1668: 1, 1664: 1, 1654: 1, 1645: 1, 1640: 1, 1633:
1, 1628: 1, 1627: 1, 1624: 1, 1621: 1, 1619: 1, 1618: 1, 1617: 1, 1615: 1,
1613: 1, 1611: 1, 1609: 1, 1607: 1, 1606: 1, 1604: 1, 1601: 1, 1600: 1, 15
96: 1, 1594: 1, 1593: 1, 1592: 1, 1591: 1, 1588: 1, 1586: 1, 1585: 1, 158
4: 1, 1580: 1, 1579: 1, 1574: 1, 1573: 1, 1572: 1, 1571: 1, 1568: 1, 1566:
1, 1564: 1, 1559: 1, 1556: 1, 1548: 1, 1546: 1, 1543: 1, 1540: 1, 1539: 1,
1538: 1, 1536: 1, 1535: 1, 1533: 1, 1531: 1, 1529: 1, 1528: 1, 1524: 1, 15
22: 1, 1519: 1, 1518: 1, 1517: 1, 1508: 1, 1507: 1, 1506: 1, 1501: 1, 149
7: 1, 1485: 1, 1484: 1, 1481: 1, 1479: 1, 1478: 1, 1477: 1, 1476: 1, 1473:
1, 1472: 1, 1471: 1, 1470: 1, 1469: 1, 1467: 1, 1465: 1, 1461: 1, 1460: 1,
1459: 1, 1457: 1, 1454: 1, 1452: 1, 1451: 1, 1450: 1, 1447: 1, 1445: 1, 14
44: 1, 1443: 1, 1442: 1, 1440: 1, 1438: 1, 1437: 1, 1435: 1, 1434: 1, 143
2: 1, 1430: 1, 1428: 1, 1426: 1, 1422: 1, 1421: 1, 1419: 1, 1418: 1, 1417:
1, 1416: 1, 1414: 1, 1413: 1, 1412: 1, 1410: 1, 1402: 1, 1400: 1, 1398: 1,
1397: 1, 1394: 1, 1393: 1, 1392: 1, 1388: 1, 1387: 1, 1386: 1, 1383: 1, 13
74: 1, 1370: 1, 1368: 1, 1363: 1, 1361: 1, 1357: 1, 1356: 1, 1354: 1, 135
3: 1, 1347: 1, 1346: 1, 1340: 1, 1337: 1, 1332: 1, 1331: 1, 1329: 1, 1328:
1, 1324: 1, 1323: 1, 1322: 1, 1320: 1, 1318: 1, 1317: 1, 1315: 1, 1311: 1,
```

```
1310: 1, 1308: 1, 1307: 1, 1306: 1, 1304: 1, 1302: 1, 1300: 1, 1299: 1, 12
98: 1, 1295: 1, 1290: 1, 1285: 1, 1283: 1, 1281: 1, 1279: 1, 1278: 1, 127
3: 1, 1272: 1, 1264: 1, 1262: 1, 1260: 1, 1259: 1, 1257: 1, 1253: 1, 1252:
1, 1251: 1, 1250: 1, 1249: 1, 1247: 1, 1238: 1, 1235: 1, 1230: 1, 1226: 1,
1221: 1, 1220: 1, 1218: 1, 1217: 1, 1215: 1, 1213: 1, 1212: 1, 1211: 1, 12
10: 1, 1207: 1, 1205: 1, 1204: 1, 1202: 1, 1200: 1, 1199: 1, 1195: 1, 119
2: 1, 1188: 1, 1187: 1, 1186: 1, 1184: 1, 1183: 1, 1181: 1, 1179: 1, 1174:
1, 1171: 1, 1169: 1, 1165: 1, 1162: 1, 1158: 1, 1156: 1, 1154: 1, 1151: 1,
1147: 1, 1145: 1, 1137: 1, 1134: 1, 1129: 1, 1117: 1, 1116: 1, 1112: 1, 11
08: 1, 1107: 1, 1106: 1, 1102: 1, 1101: 1, 1096: 1, 1095: 1, 1094: 1, 108
5: 1, 1082: 1, 1078: 1, 1076: 1, 1064: 1, 1061: 1, 1058: 1, 1057: 1, 1055:
1, 1054: 1, 1053: 1, 1050: 1, 1048: 1, 1044: 1, 1043: 1, 1041: 1, 1040: 1,
1039: 1, 1035: 1, 1032: 1, 1030: 1, 1020: 1, 1018: 1, 1013: 1, 1012: 1, 10
11: 1, 999: 1, 998: 1, 996: 1, 989: 1, 986: 1, 985: 1, 980: 1, 979: 1, 97
6: 1, 970: 1, 969: 1, 967: 1, 965: 1, 960: 1, 954: 1, 953: 1, 950: 1, 946:
1, 943: 1, 929: 1, 928: 1, 925: 1, 924: 1, 923: 1, 922: 1, 917: 1, 915: 1,
912: 1, 911: 1, 910: 1, 905: 1, 900: 1, 898: 1, 896: 1, 885: 1, 884: 1, 88
2: 1, 876: 1, 873: 1, 865: 1, 862: 1, 860: 1, 857: 1, 848: 1, 847: 1, 846:
1, 839: 1, 828: 1, 823: 1, 820: 1, 816: 1, 813: 1, 810: 1, 806: 1, 802: 1,
798: 1, 794: 1, 792: 1, 791: 1, 790: 1, 785: 1, 784: 1, 773: 1, 770: 1, 76
3: 1, 759: 1, 758: 1, 757: 1, 751: 1, 746: 1, 736: 1, 725: 1, 723: 1, 714:
1, 703: 1, 697: 1, 694: 1, 693: 1, 692: 1, 686: 1, 685: 1, 671: 1, 667: 1,
665: 1, 652: 1, 640: 1, 637: 1, 636: 1, 622: 1, 616: 1, 611: 1, 606: 1, 59
3: 1, 564: 1, 552: 1, 551: 1, 547: 1, 523: 1, 515: 1, 491: 1, 486: 1})
```

In [114]:

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

train_x_tf_1 = hstack((train_gene_var_onehotCoding, train_text_feature_tf)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_tf_1 = hstack((test_gene_var_onehotCoding, test_text_feature_tf)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_tf_1 = hstack((cv_gene_var_onehotCoding, cv_text_feature_tf)).tocsr()
cv_y = np.array(list(cv_df['Class']))


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

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

In [115]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_tf_1.sha
pe)
print("(number of data points * number of features) in test data = ", test_x_tf_1.shape
)
print("(number of data points * number of features) in cross validation data =", cv_x_t
f_1.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 5607
9)
(number of data points * number of features) in test data =  (665, 56079)
(number of data points * number of features) in cross validation data = (5
32, 56079)
```

In [116]:

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

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (5
32, 27)
```

In [117]:

```
train_combined=hstack((train_x_tf,train_x_responseCoding)).tocsr()
cv_combined=hstack((cv_x_tf,cv_x_responseCoding)).tocsr()
test_combined=hstack((test_x_tf,test_x_responseCoding)).tocsr()
print(train_combined.shape)
print(cv_combined.shape)
print(test_combined.shape)
```

```
(2124, 783576)
(532, 783576)
(665, 783576)
```

In [118]:

```
gene_vectorizer = TfidfVectorizer()
train_gene_feature_tf = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tf = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tf = gene_vectorizer.transform(cv_df['Gene'])
print(train_gene_feature_tf.shape)
print(cv_gene_feature_tf.shape)
print(test_gene_feature_tf.shape)
```

```
(2124, 229)
(532, 229)
(665, 229)
```

In [119]:

```
train_df.columns
```

Out[119]:

```
Index(['ID', 'Gene', 'Variation', 'Class', 'TEXT'], dtype='object')
```

In [120]:

```
var_vectorizer = TfidfVectorizer()
train_var_feature_tf = var_vectorizer.fit_transform(train_df['Variation'])
test_var_feature_tf = var_vectorizer.transform(test_df['Variation'])
cv_var_feature_tf = var_vectorizer.transform(cv_df['Variation'])
print(train_var_feature_tf.shape)
print(cv_var_feature_tf.shape)
print(test_var_feature_tf.shape)
```

```
(2124, 1941)
(532, 1941)
(665, 1941)
```

In [121]:

```
train_gene_var_tf = hstack((train_gene_feature_tf,train_var_feature_tf))
test_gene_var_tf = hstack((test_gene_feature_tf,test_var_feature_tf))
cv_gene_var_tf = hstack((cv_gene_feature_tf,cv_var_feature_tf))

train_x_tf = hstack((train_gene_var_tf, train_text_feature_tf)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_tf = hstack((test_gene_var_tf, test_text_feature_tf)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_tf = hstack((cv_gene_var_tf, cv_text_feature_tf)).tocsr()
cv_y = np.array(list(cv_df['Class']))


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

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

In [122]:

```
print(train_x_tf.shape)
print(cv_x_tf.shape)
print(test_x_tf.shape)
```

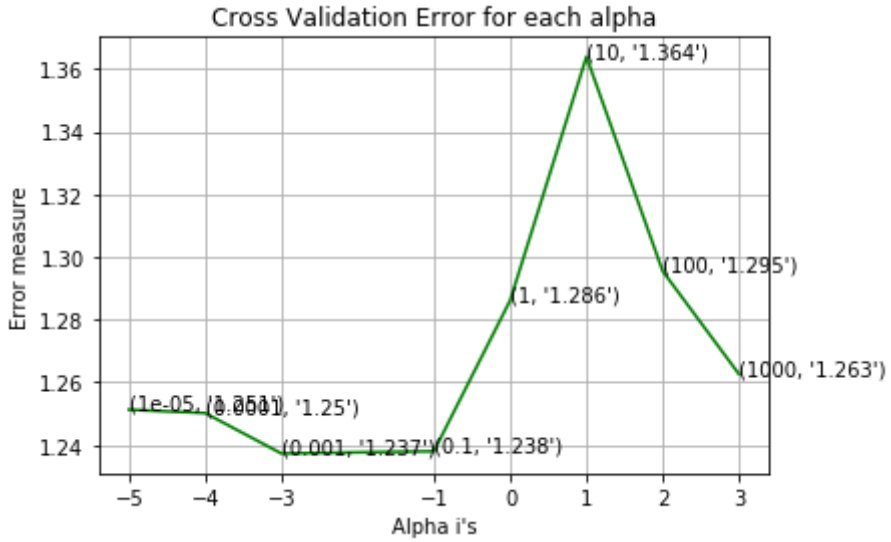
```
(2124, 56079)
(532, 56079)
(665, 56079)
```

MODEL:1 NAIVE BAYES(HYPERPARAMETER TUNING)

In [124]:

```
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_tf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    # to avoid rounding error while multiplying probabilites we use log-probability est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 1.251304959096892
for alpha = 0.0001
Log Loss : 1.2502230925145792
for alpha = 0.001
Log Loss : 1.2373265676103478
for alpha = 0.1
Log Loss : 1.238032477051545
for alpha = 1
Log Loss : 1.286157189826193
for alpha = 10
Log Loss : 1.363848698739689
for alpha = 100
Log Loss : 1.2954749855284557
for alpha = 1000
Log Loss : 1.2626535648516635
```




```
For values of best alpha =  0.001 The train log loss is: 0.782855039827901
9
For values of best alpha =  0.001 The cross validation log loss is: 1.2373
265676103478
For values of best alpha =  0.001 The test log loss is: 1.293048133941736
```

TESTING WITH BEST HYPERPARAMETERS

In [125]:

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

```
Log Loss : 1.2373265676103478
Number of missclassified point : 0.37969924812030076
-------------------- Confusion matrix --------------------
```


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


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

FEATURE IMPORTANCE FOR CORRECTLY CLASSIFIED POINT

In [126]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tf[test_p
oint_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tf(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
e'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0792 0.0752 0.0099 0.0987 0.0415 0.0347
0.6502 0.006  0.0045]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

FEATURE IMPORTANCE FOR CORRECTLY MISCLASSIFIED POINT

In [127]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tf[test_p
oint_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tf(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
e'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0991 0.0941 0.4471 0.1242 0.0526 0.0433
0.1263 0.0077 0.0055]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

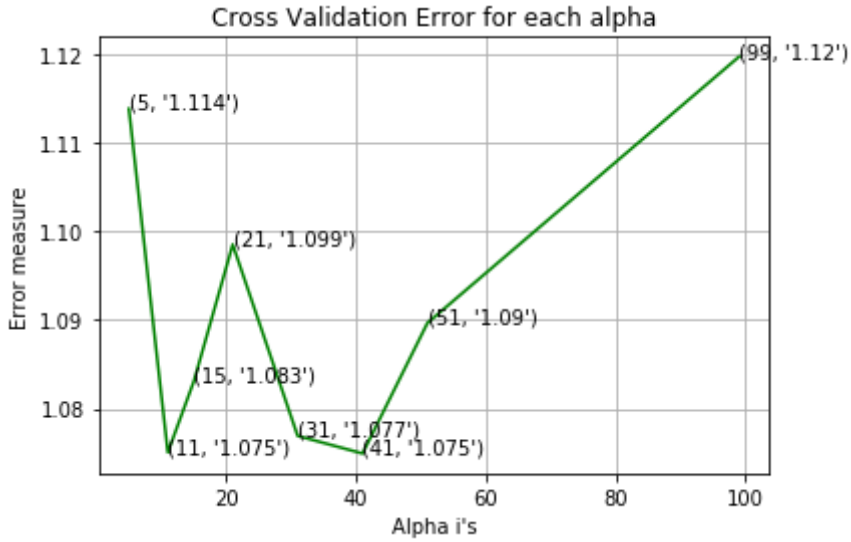
MODEL 2 KNN

HYPERPARAMETER TUNING

In [128]:

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

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


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

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

```
for alpha = 5
Log Loss : 1.1138025920287913
for alpha = 11
Log Loss : 1.0750859152636334
for alpha = 15
Log Loss : 1.0830728454096696
for alpha = 21
Log Loss : 1.098504378772884
for alpha = 31
Log Loss : 1.0769449632488242
for alpha = 41
Log Loss : 1.0749593090136231
for alpha = 51
Log Loss : 1.0896663165873033
for alpha = 99
Log Loss : 1.1196627032119757
```




```
For values of best alpha =  41 The train log loss is: 0.8394215400356405
For values of best alpha =  41 The cross validation log loss is: 1.0749593
090136231
For values of best alpha =  41 The test log loss is: 1.0908995378262252
```

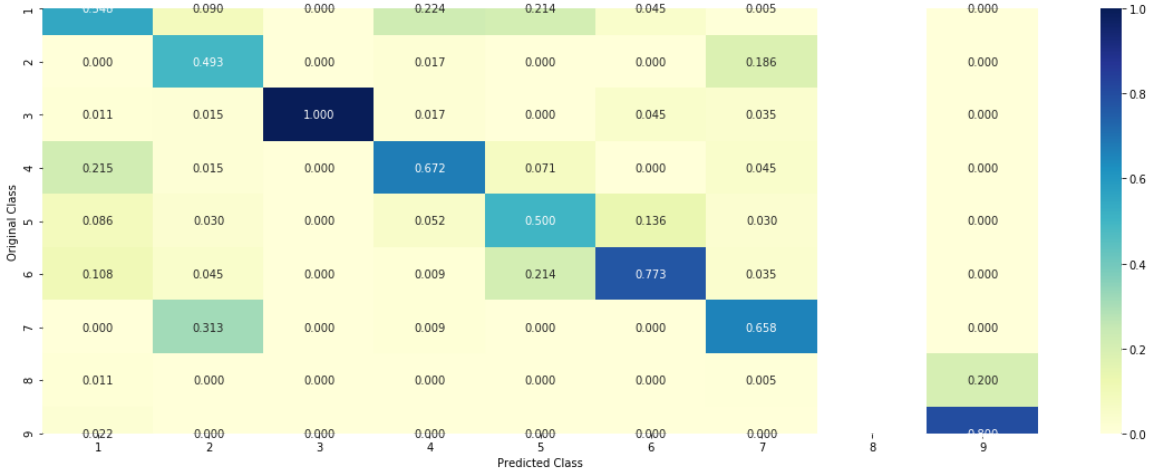
TESTING WITH BEST HYPERPARAMETERS

In [129]:

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

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

```
Log loss : 1.0749593090136231
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```


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




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

MODEL 3 LOGISTIC REGRESSION WITH CLASS BALANCING

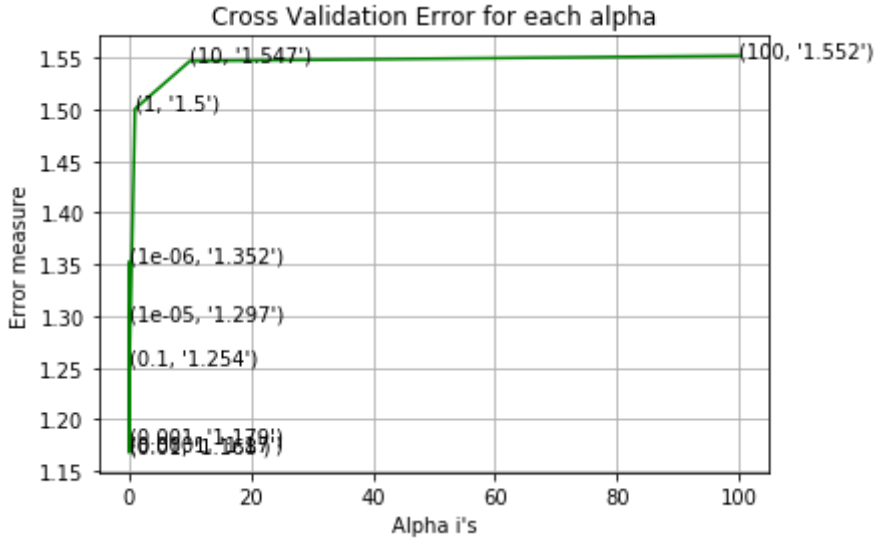
HYPERPARAMETER TUNING

In [130]:

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3521246594932628
for alpha = 1e-05
Log Loss : 1.2966264881738778
for alpha = 0.0001
Log Loss : 1.1697889186722332
for alpha = 0.001
Log Loss : 1.1788365961480598
for alpha = 0.01
Log Loss : 1.16797532125689
for alpha = 0.1
Log Loss : 1.2543379627260283
for alpha = 1
Log Loss : 1.500467174405293
for alpha = 10
Log Loss : 1.5467530361581285
for alpha = 100
Log Loss : 1.5518915354409843
```




```
For values of best alpha =  0.01 The train log loss is: 0.6292284782643831
For values of best alpha =  0.01 The cross validation log loss is: 1.16797
532125689
For values of best alpha =  0.01 The test log loss is: 1.18616529242018
```

WITH BEST HYPERPARAMETER VALUES

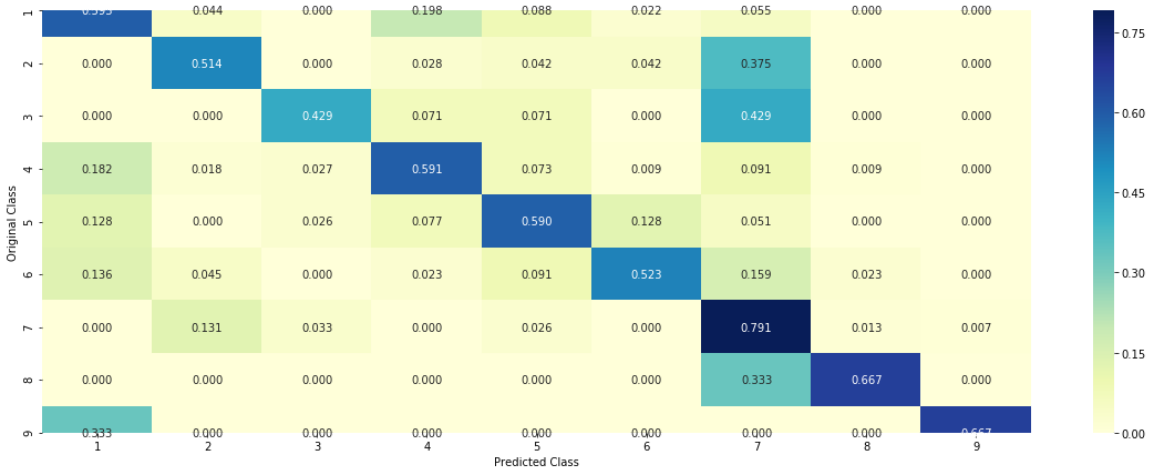
In [131]:

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

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

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

```
Log loss : 1.16797532125689
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```


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


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



FEATURE IMPORTANCE

In [132]:

```
def get_imp_feature_names_tf(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_tf[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']))
```

FEATURE IMPORTANCE FOR CORRECTLY CLASSIFIED POINT

In [133]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
s='log', random_state=42)
clf.fit(train_x_tf,train_y)
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tf(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
e'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0287 0.2303 0.006  0.0322 0.0446 0.0288
0.591  0.035  0.0035]]
Actual Class : 7
--------------------------------------------------
26 Text feature [loss] present in test data point [True]
34 Text feature [protein] present in test data point [True]
39 Text feature [predicted] present in test data point [True]
40 Text feature [function] present in test data point [True]
43 Text feature [functions] present in test data point [True]
44 Text feature [type] present in test data point [True]
46 Text feature [reduced] present in test data point [True]
50 Text feature [analysis] present in test data point [True]
51 Text feature [co] present in test data point [True]
52 Text feature [important] present in test data point [True]
53 Text feature [dna] present in test data point [True]
54 Text feature [affected] present in test data point [True]
55 Text feature [wild] present in test data point [True]
57 Text feature [involved] present in test data point [True]
59 Text feature [control] present in test data point [True]
60 Text feature [binding] present in test data point [True]
61 Text feature [whether] present in test data point [True]
62 Text feature [gene] present in test data point [True]
64 Text feature [proteins] present in test data point [True]
67 Text feature [results] present in test data point [True]
69 Text feature [one] present in test data point [True]
70 Text feature [ability] present in test data point [True]
71 Text feature [changes] present in test data point [True]
72 Text feature [functional] present in test data point [True]
73 Text feature [terminal] present in test data point [True]
75 Text feature [full] present in test data point [True]
76 Text feature [required] present in test data point [True]
77 Text feature [complex] present in test data point [True]
80 Text feature [two] present in test data point [True]
81 Text feature [variant] present in test data point [True]
82 Text feature [based] present in test data point [True]
83 Text feature [lacking] present in test data point [True]
85 Text feature [indicated] present in test data point [True]
86 Text feature [multiple] present in test data point [True]
87 Text feature [used] present in test data point [True]
88 Text feature [eight] present in test data point [True]
89 Text feature [analyzed] present in test data point [True]
91 Text feature [table] present in test data point [True]
92 Text feature [possible] present in test data point [True]
94 Text feature [inactivation] present in test data point [True]
95 Text feature [using] present in test data point [True]
96 Text feature [critical] present in test data point [True]
105 Text feature [thus] present in test data point [True]
106 Text feature [purified] present in test data point [True]
107 Text feature [site] present in test data point [True]
108 Text feature [putative] present in test data point [True]
109 Text feature [expected] present in test data point [True]
111 Text feature [several] present in test data point [True]
115 Text feature [genetic] present in test data point [True]
119 Text feature [performed] present in test data point [True]
120 Text feature [30] present in test data point [True]
123 Text feature [splice] present in test data point [True]
124 Text feature [studies] present in test data point [True]
125 Text feature [methods] present in test data point [True]
127 Text feature [damage] present in test data point [True]
128 Text feature [associated] present in test data point [True]
```

```
129 Text feature [previous] present in test data point [True]
130 Text feature [highly] present in test data point [True]
131 Text feature [affects] present in test data point [True]
132 Text feature [included] present in test data point [True]
135 Text feature [relevant] present in test data point [True]
136 Text feature [relevance] present in test data point [True]
137 Text feature [many] present in test data point [True]
138 Text feature [study] present in test data point [True]
139 Text feature [indicates] present in test data point [True]
140 Text feature [likely] present in test data point [True]
141 Text feature [expression] present in test data point [True]
143 Text feature [three] present in test data point [True]
144 Text feature [suppresses] present in test data point [True]
145 Text feature [therefore] present in test data point [True]
147 Text feature [26] present in test data point [True]
148 Text feature [according] present in test data point [True]
149 Text feature [quantitative] present in test data point [True]
150 Text feature [data] present in test data point [True]
153 Text feature [genes] present in test data point [True]
155 Text feature [determine] present in test data point [True]
158 Text feature [lower] present in test data point [True]
160 Text feature [algorithm] present in test data point [True]
161 Text feature [assay] present in test data point [True]
162 Text feature [determined] present in test data point [True]
164 Text feature [discussion] present in test data point [True]
165 Text feature [interact] present in test data point [True]
166 Text feature [interacts] present in test data point [True]
167 Text feature [system] present in test data point [True]
168 Text feature [conversely] present in test data point [True]
169 Text feature [see] present in test data point [True]
171 Text feature [42] present in test data point [True]
172 Text feature [http] present in test data point [True]
173 Text feature [lack] present in test data point [True]
174 Text feature [although] present in test data point [True]
176 Text feature [suggesting] present in test data point [True]
177 Text feature [relatively] present in test data point [True]
178 Text feature [provide] present in test data point [True]
179 Text feature [analyses] present in test data point [True]
180 Text feature [introduction] present in test data point [True]
184 Text feature [either] present in test data point [True]
185 Text feature [containing] present in test data point [True]
186 Text feature [known] present in test data point [True]
189 Text feature [least] present in test data point [True]
190 Text feature [functionally] present in test data point [True]
192 Text feature [specifically] present in test data point [True]
194 Text feature [shown] present in test data point [True]
195 Text feature [transfection] present in test data point [True]
201 Text feature [retention] present in test data point [True]
203 Text feature [tested] present in test data point [True]
204 Text feature [evidence] present in test data point [True]
205 Text feature [observation] present in test data point [True]
207 Text feature [significant] present in test data point [True]
208 Text feature [5a] present in test data point [True]
209 Text feature [also] present in test data point [True]
211 Text feature [observed] present in test data point [True]
214 Text feature [method] present in test data point [True]
215 Text feature [16] present in test data point [True]
217 Text feature [controls] present in test data point [True]
218 Text feature [test] present in test data point [True]
219 Text feature [directly] present in test data point [True]
220 Text feature [activity] present in test data point [True]
```

```
222 Text feature [including] present in test data point [True]
225 Text feature [consequences] present in test data point [True]
226 Text feature [tagged] present in test data point [True]
227 Text feature [cancer] present in test data point [True]
228 Text feature [different] present in test data point [True]
229 Text feature [information] present in test data point [True]
233 Text feature [limited] present in test data point [True]
238 Text feature [applied] present in test data point [True]
239 Text feature [encodes] present in test data point [True]
240 Text feature [considered] present in test data point [True]
241 Text feature [carrying] present in test data point [True]
242 Text feature [population] present in test data point [True]
244 Text feature [number] present in test data point [True]
250 Text feature [potentially] present in test data point [True]
251 Text feature [may] present in test data point [True]
253 Text feature [reported] present in test data point [True]
254 Text feature [region] present in test data point [True]
255 Text feature [available] present in test data point [True]
256 Text feature [mutations] present in test data point [True]
257 Text feature [present] present in test data point [True]
258 Text feature [change] present in test data point [True]
260 Text feature [31] present in test data point [True]
262 Text feature [showing] present in test data point [True]
263 Text feature [terms] present in test data point [True]
264 Text feature [38] present in test data point [True]
265 Text feature [sequences] present in test data point [True]
266 Text feature [chromosome] present in test data point [True]
267 Text feature [levels] present in test data point [True]
270 Text feature [failed] present in test data point [True]
271 Text feature [referred] present in test data point [True]
276 Text feature [substrate] present in test data point [True]
277 Text feature [5c] present in test data point [True]
278 Text feature [whereas] present in test data point [True]
279 Text feature [effect] present in test data point [True]
282 Text feature [proposed] present in test data point [True]
283 Text feature [33] present in test data point [True]
284 Text feature [would] present in test data point [True]
285 Text feature [indicating] present in test data point [True]
287 Text feature [2011] present in test data point [True]
288 Text feature [altered] present in test data point [True]
289 Text feature [predict] present in test data point [True]
290 Text feature [within] present in test data point [True]
291 Text feature [unable] present in test data point [True]
292 Text feature [binds] present in test data point [True]
294 Text feature [interaction] present in test data point [True]
295 Text feature [large] present in test data point [True]
297 Text feature [interacting] present in test data point [True]
298 Text feature [40] present in test data point [True]
299 Text feature [15] present in test data point [True]
300 Text feature [panel] present in test data point [True]
305 Text feature [note] present in test data point [True]
306 Text feature [unlikely] present in test data point [True]
307 Text feature [stress] present in test data point [True]
308 Text feature [constitutive] present in test data point [True]
309 Text feature [decreased] present in test data point [True]
310 Text feature [larger] present in test data point [True]
311 Text feature [four] present in test data point [True]
312 Text feature [deleted] present in test data point [True]
313 Text feature [27] present in test data point [True]
318 Text feature [another] present in test data point [True]
319 Text feature [complete] present in test data point [True]
```

```
320 Text feature [reduction] present in test data point [True]
321 Text feature [edu] present in test data point [True]
324 Text feature [identify] present in test data point [True]
325 Text feature [role] present in test data point [True]
329 Text feature [global] present in test data point [True]
330 Text feature [p53] present in test data point [True]
331 Text feature [importantly] present in test data point [True]
333 Text feature [transfected] present in test data point [True]
334 Text feature [set] present in test data point [True]
336 Text feature [indicate] present in test data point [True]
337 Text feature [make] present in test data point [True]
338 Text feature [however] present in test data point [True]
341 Text feature [occurring] present in test data point [True]
344 Text feature [35] present in test data point [True]
346 Text feature [family] present in test data point [True]
350 Text feature [mediated] present in test data point [True]
355 Text feature [recognized] present in test data point [True]
359 Text feature [previously] present in test data point [True]
361 Text feature [mutation] present in test data point [True]
363 Text feature [identified] present in test data point [True]
364 Text feature [low] present in test data point [True]
365 Text feature [interactions] present in test data point [True]
368 Text feature [first] present in test data point [True]
371 Text feature [assays] present in test data point [True]
374 Text feature [association] present in test data point [True]
376 Text feature [confirmed] present in test data point [True]
381 Text feature [deficient] present in test data point [True]
383 Text feature [reporter] present in test data point [True]
384 Text feature [key] present in test data point [True]
385 Text feature [final] present in test data point [True]
394 Text feature [negatively] present in test data point [True]
395 Text feature [6a] present in test data point [True]
400 Text feature [5d] present in test data point [True]
402 Text feature [finally] present in test data point [True]
405 Text feature [model] present in test data point [True]
407 Text feature [conclusions] present in test data point [True]
409 Text feature [nearly] present in test data point [True]
410 Text feature [little] present in test data point [True]
411 Text feature [severely] present in test data point [True]
414 Text feature [could] present in test data point [True]
417 Text feature [specific] present in test data point [True]
418 Text feature [partially] present in test data point [True]
419 Text feature [individual] present in test data point [True]
420 Text feature [predictions] present in test data point [True]
421 Text feature [negative] present in test data point [True]
424 Text feature [resulted] present in test data point [True]
426 Text feature [contains] present in test data point [True]
428 Text feature [effects] present in test data point [True]
430 Text feature [specificity] present in test data point [True]
431 Text feature [transcription] present in test data point [True]
432 Text feature [29] present in test data point [True]
435 Text feature [substrates] present in test data point [True]
439 Text feature [include] present in test data point [True]
444 Text feature [discovery] present in test data point [True]
445 Text feature [vitro] present in test data point [True]
446 Text feature [expressed] present in test data point [True]
447 Text feature [tgf] present in test data point [True]
448 Text feature [published] present in test data point [True]
449 Text feature [due] present in test data point [True]
452 Text feature [regions] present in test data point [True]
453 Text feature [roles] present in test data point [True]
```

```
455 Text feature [reduce] present in test data point [True]
456 Text feature [fraction] present in test data point [True]
457 Text feature [eluted] present in test data point [True]
464 Text feature [interest] present in test data point [True]
467 Text feature [values] present in test data point [True]
470 Text feature [definition] present in test data point [True]
471 Text feature [mapping] present in test data point [True]
476 Text feature [deletion] present in test data point [True]
477 Text feature [respectively] present in test data point [True]
478 Text feature [constructs] present in test data point [True]
479 Text feature [demonstrate] present in test data point [True]
480 Text feature [located] present in test data point [True]
482 Text feature [view] present in test data point [True]
483 Text feature [nine] present in test data point [True]
484 Text feature [six] present in test data point [True]
486 Text feature [frequency] present in test data point [True]
488 Text feature [pten] present in test data point [True]
491 Text feature [alter] present in test data point [True]
492 Text feature [remaining] present in test data point [True]
496 Text feature [significantly] present in test data point [True]
498 Text feature [cause] present in test data point [True]
499 Text feature [construct] present in test data point [True]
Out of the top  500  features  261 are present in query point
```

FEATURE IMPORTANCE FOR MISCLASSIFIED POINTS

In [134]:

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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.015  0.0161 0.6782 0.2482 0.0169 0.0093
0.0093 0.0042 0.0027]]
Actual Class : 4
--------------------------------------------------
47 Text feature [shown] present in test data point [True]
53 Text feature [cell] present in test data point [True]
76 Text feature [also] present in test data point [True]
82 Text feature [1a] present in test data point [True]
101 Text feature [respectively] present in test data point [True]
102 Text feature [development] present in test data point [True]
106 Text feature [containing] present in test data point [True]
109 Text feature [another] present in test data point [True]
110 Text feature [studies] present in test data point [True]
111 Text feature [form] present in test data point [True]
112 Text feature [using] present in test data point [True]
116 Text feature [including] present in test data point [True]
120 Text feature [however] present in test data point [True]
121 Text feature [10] present in test data point [True]
123 Text feature [role] present in test data point [True]
124 Text feature [observed] present in test data point [True]
130 Text feature [protein] present in test data point [True]
131 Text feature [type] present in test data point [True]
136 Text feature [may] present in test data point [True]
137 Text feature [resulting] present in test data point [True]
140 Text feature [obtained] present in test data point [True]
143 Text feature [could] present in test data point [True]
144 Text feature [following] present in test data point [True]
148 Text feature [addition] present in test data point [True]
152 Text feature [performed] present in test data point [True]
154 Text feature [either] present in test data point [True]
156 Text feature [therefore] present in test data point [True]
157 Text feature [majority] present in test data point [True]
158 Text feature [two] present in test data point [True]
161 Text feature [cells] present in test data point [True]
166 Text feature [normal] present in test data point [True]
168 Text feature [confirmed] present in test data point [True]
169 Text feature [recently] present in test data point [True]
170 Text feature [proteins] present in test data point [True]
172 Text feature [2b] present in test data point [True]
174 Text feature [domain] present in test data point [True]
178 Text feature [contribute] present in test data point [True]
179 Text feature [compared] present in test data point [True]
180 Text feature [due] present in test data point [True]
182 Text feature [gene] present in test data point [True]
185 Text feature [previous] present in test data point [True]
188 Text feature [additional] present in test data point [True]
190 Text feature [increased] present in test data point [True]
191 Text feature [mutant] present in test data point [True]
195 Text feature [one] present in test data point [True]
198 Text feature [show] present in test data point [True]
199 Text feature [similar] present in test data point [True]
200 Text feature [found] present in test data point [True]
202 Text feature [suggesting] present in test data point [True]
203 Text feature [expression] present in test data point [True]
207 Text feature [mechanisms] present in test data point [True]
210 Text feature [next] present in test data point [True]
215 Text feature [15] present in test data point [True]
216 Text feature [30] present in test data point [True]
221 Text feature [demonstrate] present in test data point [True]
226 Text feature [cause] present in test data point [True]
```

```
232 Text feature [effect] present in test data point [True]
234 Text feature [furthermore] present in test data point [True]
235 Text feature [data] present in test data point [True]
236 Text feature [yet] present in test data point [True]
246 Text feature [directly] present in test data point [True]
247 Text feature [mutation] present in test data point [True]
248 Text feature [presence] present in test data point [True]
249 Text feature [wild] present in test data point [True]
250 Text feature [reported] present in test data point [True]
252 Text feature [showed] present in test data point [True]
254 Text feature [multiple] present in test data point [True]
260 Text feature [within] present in test data point [True]
261 Text feature [discussion] present in test data point [True]
263 Text feature [use] present in test data point [True]
266 Text feature [used] present in test data point [True]
268 Text feature [total] present in test data point [True]
270 Text feature [specific] present in test data point [True]
279 Text feature [corresponding] present in test data point [True]
283 Text feature [table] present in test data point [True]
285 Text feature [see] present in test data point [True]
289 Text feature [complex] present in test data point [True]
295 Text feature [growth] present in test data point [True]
300 Text feature [lines] present in test data point [True]
302 Text feature [12] present in test data point [True]
303 Text feature [3b] present in test data point [True]
304 Text feature [previously] present in test data point [True]
306 Text feature [2a] present in test data point [True]
307 Text feature [four] present in test data point [True]
308 Text feature [100] present in test data point [True]
311 Text feature [indicating] present in test data point [True]
312 Text feature [introduction] present in test data point [True]
316 Text feature [since] present in test data point [True]
320 Text feature [25] present in test data point [True]
325 Text feature [figure] present in test data point [True]
327 Text feature [reduced] present in test data point [True]
334 Text feature [enhanced] present in test data point [True]
335 Text feature [whereas] present in test data point [True]
338 Text feature [contrast] present in test data point [True]
339 Text feature [derived] present in test data point [True]
341 Text feature [frequently] present in test data point [True]
343 Text feature [1b] present in test data point [True]
347 Text feature [lacking] present in test data point [True]
350 Text feature [expressed] present in test data point [True]
361 Text feature [findings] present in test data point [True]
363 Text feature [antibody] present in test data point [True]
365 Text feature [dependent] present in test data point [True]
367 Text feature [40] present in test data point [True]
373 Text feature [common] present in test data point [True]
376 Text feature [identified] present in test data point [True]
377 Text feature [analysis] present in test data point [True]
380 Text feature [observation] present in test data point [True]
382 Text feature [well] present in test data point [True]
387 Text feature [single] present in test data point [True]
389 Text feature [direct] present in test data point [True]
391 Text feature [4a] present in test data point [True]
392 Text feature [caused] present in test data point [True]
393 Text feature [described] present in test data point [True]
398 Text feature [genes] present in test data point [True]
400 Text feature [relative] present in test data point [True]
402 Text feature [indicate] present in test data point [True]
406 Text feature [1c] present in test data point [True]
```

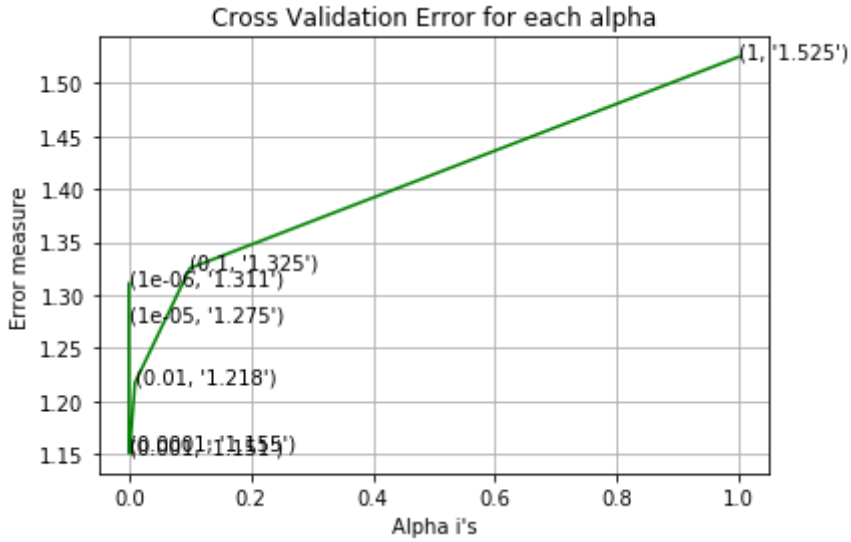
```
415 Text feature [target] present in test data point [True]
416 Text feature [20] present in test data point [True]
417 Text feature [potential] present in test data point [True]
465 Text feature [indicated] present in test data point [True]
467 Text feature [kinase] present in test data point [True]
Out of the top  500  features  122 are present in query point
```

LOGISTIC REGRESSION WITHOUT CLASS BALANCING

In [135]:

```
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_tf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tf)
    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_tf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tf, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.3109038950608252
for alpha = 1e-05
Log Loss : 1.2753383640980418
for alpha = 0.0001
Log Loss : 1.1551349566863918
for alpha = 0.001
Log Loss : 1.1508098400700097
for alpha = 0.01
Log Loss : 1.217504983516634
for alpha = 0.1
Log Loss : 1.3253391247243174
for alpha = 1
Log Loss : 1.5245976100688465
```



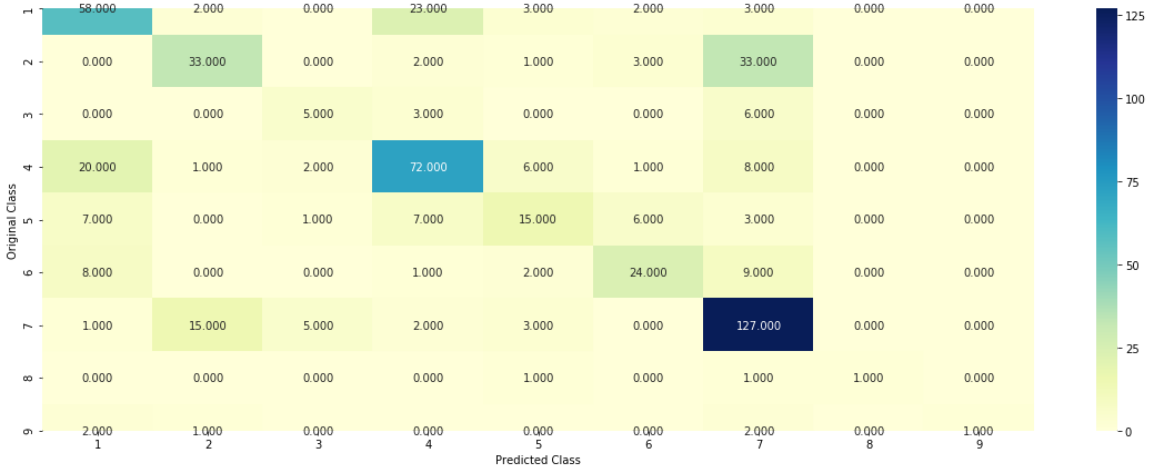

```
For values of best alpha =  0.001 The train log loss is: 0.474630250550087
95
For values of best alpha =  0.001 The cross validation log loss is: 1.1508
098400700097
For values of best alpha =  0.001 The test log loss is: 1.1109757923916277
```

WITH BEST HYPERPARAMETER VALUES

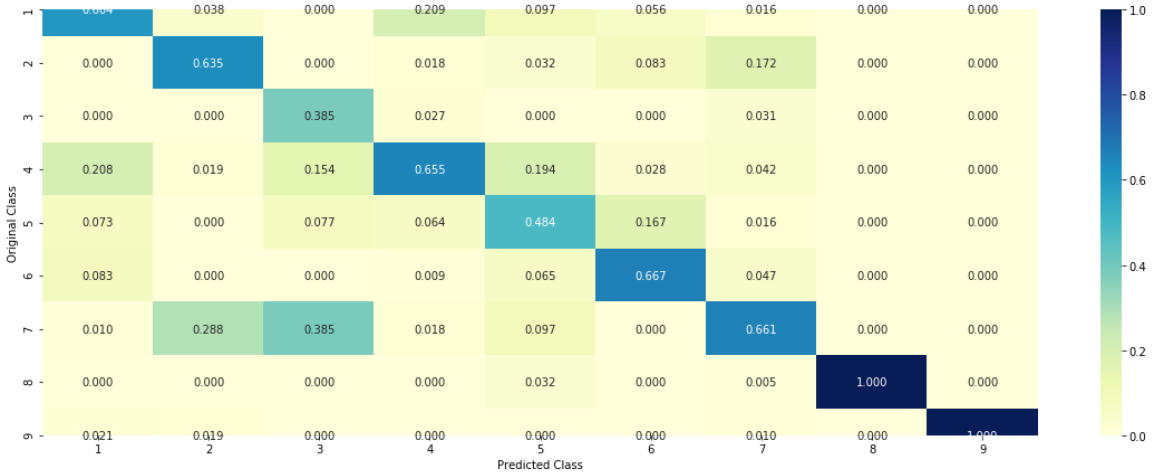
In [136]:

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

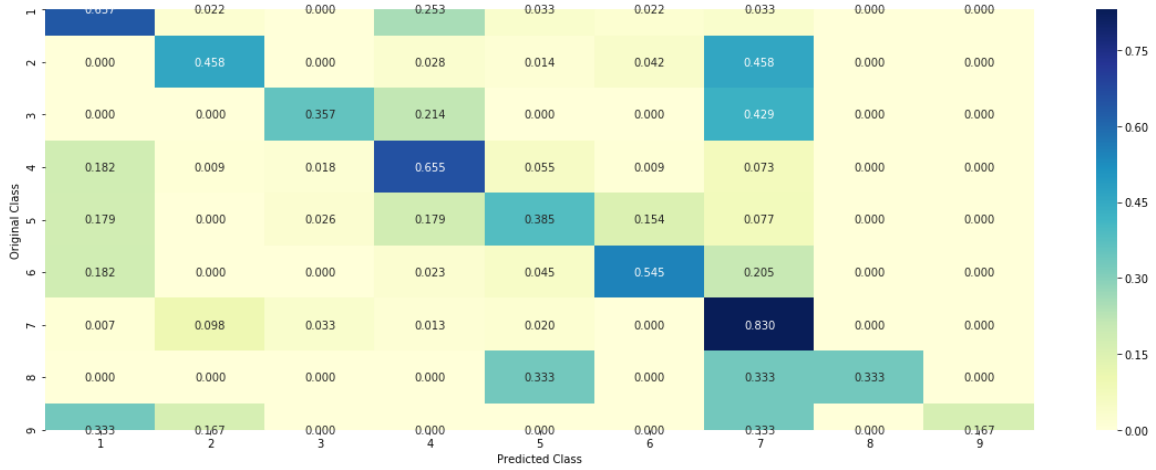
```
Log loss : 1.1508098400700097
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```




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




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



FEATURE IMPORTANCE FOR CORRECTLY CLASSIFIED POINT

In [137]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tf,train_y)
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tf(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
e'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.790e-02 1.423e-01 1.100e-03 2.600e-02
1.630e-02 1.150e-02 7.656e-01
  1.870e-02 7.000e-04]]
Actual Class : 7
--------------------------------------------------
109 Text feature [constitutive] present in test data point [True]
119 Text feature [3t3] present in test data point [True]
145 Text feature [constitutively] present in test data point [True]
159 Text feature [loss] present in test data point [True]
188 Text feature [activation] present in test data point [True]
207 Text feature [nf] present in test data point [True]
226 Text feature [downstream] present in test data point [True]
238 Text feature [oncogene] present in test data point [True]
251 Text feature [receptors] present in test data point [True]
252 Text feature [thyroid] present in test data point [True]
264 Text feature [adenocarcinoma] present in test data point [True]
279 Text feature [phosphorylation] present in test data point [True]
281 Text feature [expressing] present in test data point [True]
283 Text feature [ligand] present in test data point [True]
316 Text feature [autophosphorylation] present in test data point [True]
326 Text feature [extracellular] present in test data point [True]
355 Text feature [activate] present in test data point [True]
367 Text feature [inactivation] present in test data point [True]
369 Text feature [function] present in test data point [True]
370 Text feature [suppresses] present in test data point [True]
371 Text feature [hard] present in test data point [True]
376 Text feature [tyrosine] present in test data point [True]
377 Text feature [serum] present in test data point [True]
392 Text feature [tx] present in test data point [True]
421 Text feature [receptor] present in test data point [True]
424 Text feature [predicted] present in test data point [True]
429 Text feature [signaling] present in test data point [True]
455 Text feature [mb] present in test data point [True]
461 Text feature [activating] present in test data point [True]
467 Text feature [gene] present in test data point [True]
488 Text feature [oncogenes] present in test data point [True]
489 Text feature [affected] present in test data point [True]
492 Text feature [eight] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

FEATURE IMPORTANCE FOR MISCLASSIFIED POINT

In [138]:

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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0061 0.0123 0.6565 0.2931 0.0143 0.0055
0.0062 0.0043 0.0018]]
Actual Class : 4
--------------------------------------------------
90 Text feature [shown] present in test data point [True]
99 Text feature [cell] present in test data point [True]
109 Text feature [1a] present in test data point [True]
111 Text feature [also] present in test data point [True]
118 Text feature [demonstrate] present in test data point [True]
119 Text feature [respectively] present in test data point [True]
121 Text feature [suggesting] present in test data point [True]
126 Text feature [cells] present in test data point [True]
128 Text feature [development] present in test data point [True]
131 Text feature [recently] present in test data point [True]
139 Text feature [either] present in test data point [True]
140 Text feature [2b] present in test data point [True]
141 Text feature [may] present in test data point [True]
142 Text feature [role] present in test data point [True]
144 Text feature [domain] present in test data point [True]
145 Text feature [gene] present in test data point [True]
147 Text feature [expression] present in test data point [True]
148 Text feature [however] present in test data point [True]
149 Text feature [observed] present in test data point [True]
150 Text feature [confirmed] present in test data point [True]
151 Text feature [found] present in test data point [True]
153 Text feature [studies] present in test data point [True]
154 Text feature [previous] present in test data point [True]
155 Text feature [protein] present in test data point [True]
158 Text feature [using] present in test data point [True]
161 Text feature [additional] present in test data point [True]
163 Text feature [mutant] present in test data point [True]
165 Text feature [performed] present in test data point [True]
168 Text feature [type] present in test data point [True]
170 Text feature [including] present in test data point [True]
171 Text feature [form] present in test data point [True]
173 Text feature [compared] present in test data point [True]
174 Text feature [containing] present in test data point [True]
176 Text feature [two] present in test data point [True]
177 Text feature [10] present in test data point [True]
178 Text feature [addition] present in test data point [True]
179 Text feature [increased] present in test data point [True]
182 Text feature [therefore] present in test data point [True]
183 Text feature [resulting] present in test data point [True]
184 Text feature [normal] present in test data point [True]
185 Text feature [could] present in test data point [True]
186 Text feature [directly] present in test data point [True]
187 Text feature [proteins] present in test data point [True]
189 Text feature [another] present in test data point [True]
190 Text feature [presence] present in test data point [True]
191 Text feature [mutation] present in test data point [True]
193 Text feature [show] present in test data point [True]
197 Text feature [obtained] present in test data point [True]
199 Text feature [similar] present in test data point [True]
205 Text feature [contribute] present in test data point [True]
207 Text feature [due] present in test data point [True]
210 Text feature [majority] present in test data point [True]
213 Text feature [one] present in test data point [True]
214 Text feature [previously] present in test data point [True]
220 Text feature [lacking] present in test data point [True]
221 Text feature [furthermore] present in test data point [True]
```

```
222 Text feature [indicating] present in test data point [True]
224 Text feature [next] present in test data point [True]
225 Text feature [growth] present in test data point [True]
226 Text feature [following] present in test data point [True]
231 Text feature [figure] present in test data point [True]
233 Text feature [reported] present in test data point [True]
240 Text feature [mechanisms] present in test data point [True]
241 Text feature [2a] present in test data point [True]
242 Text feature [showed] present in test data point [True]
243 Text feature [3b] present in test data point [True]
245 Text feature [specific] present in test data point [True]
247 Text feature [cause] present in test data point [True]
249 Text feature [wild] present in test data point [True]
251 Text feature [reduced] present in test data point [True]
253 Text feature [identified] present in test data point [True]
255 Text feature [1b] present in test data point [True]
262 Text feature [expressed] present in test data point [True]
263 Text feature [discussion] present in test data point [True]
264 Text feature [lines] present in test data point [True]
268 Text feature [1c] present in test data point [True]
269 Text feature [complex] present in test data point [True]
281 Text feature [enhanced] present in test data point [True]
282 Text feature [effect] present in test data point [True]
285 Text feature [indicated] present in test data point [True]
289 Text feature [multiple] present in test data point [True]
290 Text feature [whereas] present in test data point [True]
296 Text feature [yet] present in test data point [True]
297 Text feature [confirm] present in test data point [True]
300 Text feature [dependent] present in test data point [True]
304 Text feature [described] present in test data point [True]
306 Text feature [15] present in test data point [True]
309 Text feature [data] present in test data point [True]
310 Text feature [introduction] present in test data point [True]
311 Text feature [contrast] present in test data point [True]
316 Text feature [within] present in test data point [True]
317 Text feature [indicate] present in test data point [True]
319 Text feature [antibody] present in test data point [True]
322 Text feature [findings] present in test data point [True]
326 Text feature [4a] present in test data point [True]
329 Text feature [12] present in test data point [True]
331 Text feature [potential] present in test data point [True]
335 Text feature [derived] present in test data point [True]
336 Text feature [common] present in test data point [True]
340 Text feature [30] present in test data point [True]
368 Text feature [table] present in test data point [True]
369 Text feature [mutations] present in test data point [True]
370 Text feature [well] present in test data point [True]
372 Text feature [four] present in test data point [True]
373 Text feature [consistent] present in test data point [True]
374 Text feature [identification] present in test data point [True]
375 Text feature [total] present in test data point [True]
381 Text feature [caused] present in test data point [True]
387 Text feature [40] present in test data point [True]
388 Text feature [kinase] present in test data point [True]
389 Text feature [frequently] present in test data point [True]
392 Text feature [3a] present in test data point [True]
395 Text feature [see] present in test data point [True]
401 Text feature [whether] present in test data point [True]
403 Text feature [analysis] present in test data point [True]
406 Text feature [used] present in test data point [True]
415 Text feature [downstream] present in test data point [True]
```

```
416 Text feature [determine] present in test data point [True]
418 Text feature [activity] present in test data point [True]
419 Text feature [target] present in test data point [True]
422 Text feature [direct] present in test data point [True]
424 Text feature [single] present in test data point [True]
428 Text feature [results] present in test data point [True]
429 Text feature [important] present in test data point [True]
432 Text feature [4b] present in test data point [True]
438 Text feature [relative] present in test data point [True]
441 Text feature [since] present in test data point [True]
445 Text feature [detected] present in test data point [True]
449 Text feature [observation] present in test data point [True]
453 Text feature [100] present in test data point [True]
454 Text feature [levels] present in test data point [True]
455 Text feature [use] present in test data point [True]
456 Text feature [genes] present in test data point [True]
458 Text feature [formation] present in test data point [True]
463 Text feature [lower] present in test data point [True]
466 Text feature [three] present in test data point [True]
467 Text feature [signaling] present in test data point [True]
469 Text feature [13] present in test data point [True]
475 Text feature [approximately] present in test data point [True]
476 Text feature [corresponding] present in test data point [True]
483 Text feature [although] present in test data point [True]
489 Text feature [chromosome] present in test data point [True]
497 Text feature [terminal] present in test data point [True]
499 Text feature [inhibit] present in test data point [True]
Out of the top  500  features  144 are present in query point
```

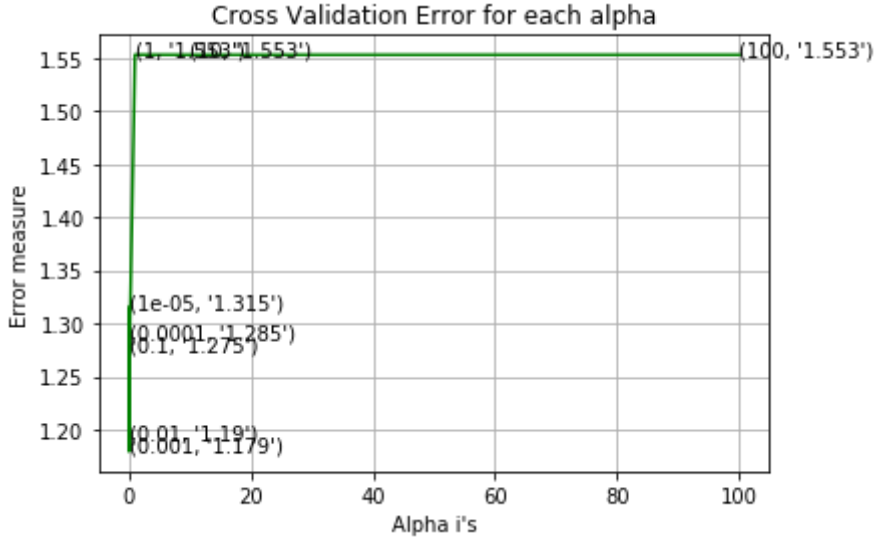
LINEAR SVM

HYPERPARAMETER TUNING

In [139]:

```
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_tf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for C = 1e-05
Log Loss : 1.3149009114698345
for C = 0.0001
Log Loss : 1.285460285298496
for C = 0.001
Log Loss : 1.1792885838399265
for C = 0.01
Log Loss : 1.1901873539688492
for C = 0.1
Log Loss : 1.2745672555090437
for C = 1
Log Loss : 1.5529530976306107
for C = 10
Log Loss : 1.5528383626671385
for C = 100
Log Loss : 1.5528463873292306
```




```
For values of best alpha =  0.001 The train log loss is: 0.517555295566969
8
For values of best alpha =  0.001 The cross validation log loss is: 1.1792
885838399265
For values of best alpha =  0.001 The test log loss is: 1.1478809188138568
```

WITH BEST HYPERPARAMETER VALUES

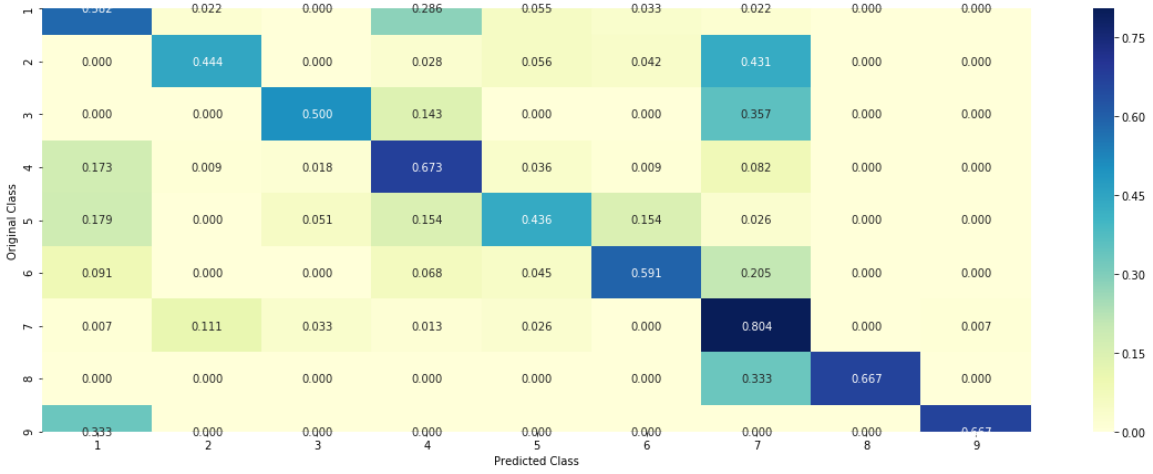
In [140]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
2,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_tf, train_y,cv_x_tf,cv_y, clf)
```

```
Log loss : 1.1792885838399265
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```


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


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



FEATURE IMPORTANCE FOR MISCLASSIFIED POINTS

In [141]:

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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0263 0.044  0.7517 0.0553 0.0431 0.016
0.0521 0.0042 0.0073]]
Actual Class : 4
--------------------------------------------------
211 Text feature [initial] present in test data point [True]
350 Text feature [shagreen] present in test data point [True]
380 Text feature [characterised] present in test data point [True]
383 Text feature [lacking] present in test data point [True]
389 Text feature [mental] present in test data point [True]
397 Text feature [demonstrate] present in test data point [True]
490 Text feature [amplification] present in test data point [True]
494 Text feature [mutation] present in test data point [True]
496 Text feature [g1035s] present in test data point [True]
497 Text feature [l180p] present in test data point [True]
498 Text feature [671t] present in test data point [True]
499 Text feature [381deltgt] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

FEATURE IMPORTANCE FOR CORRECTLY CLASSIFIED POINTS

In [142]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
2)
clf.fit(train_x_tf,train_y)
test_point_index = 10
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tf(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
e'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0457 0.1198 0.0059 0.0525 0.0382 0.0356
0.6796 0.0177 0.005 ]]
Actual Class : 7
--------------------------------------------------
281 Text feature [3t3] present in test data point [True]
286 Text feature [hard] present in test data point [True]
384 Text feature [constitutive] present in test data point [True]
386 Text feature [constitutively] present in test data point [True]
397 Text feature [intratumoral] present in test data point [True]
400 Text feature [adenocarcinoma] present in test data point [True]
424 Text feature [nf] present in test data point [True]
491 Text feature [oncogene] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

RANDOM FOREST CLASSIFIER

HYPERPARAMETER TUNING

In [143]:

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

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

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

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

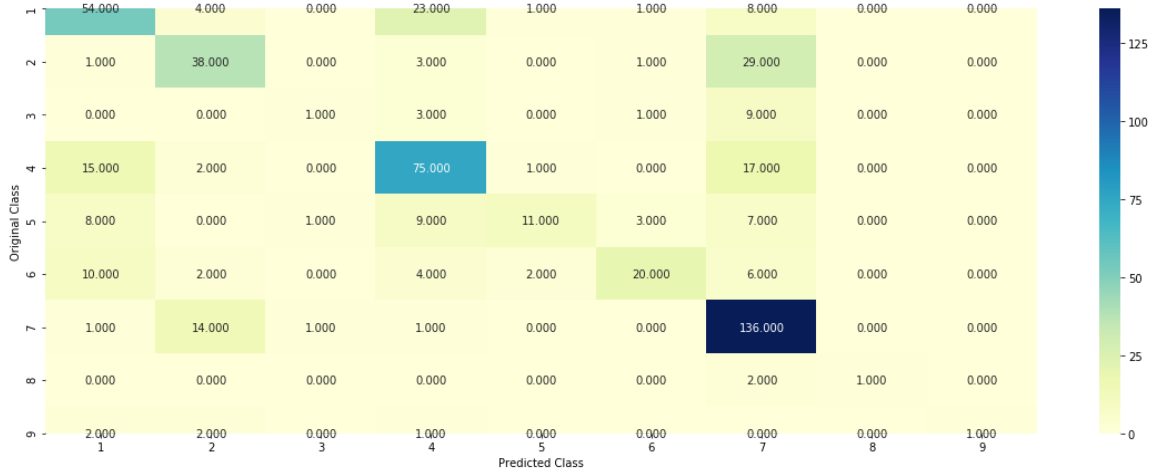
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.206486838965651
for n_estimators = 100 and max depth =  10
Log Loss : 1.1662297256095386
for n_estimators = 200 and max depth =  5
Log Loss : 1.2053458480406747
for n_estimators = 200 and max depth =  10
Log Loss : 1.1563745460961978
for n_estimators = 500 and max depth =  5
Log Loss : 1.2014459926258843
for n_estimators = 500 and max depth =  10
Log Loss : 1.1475382152453655
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1981731784745593
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1433626231024736
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1973873297497426
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1398791801231054
For values of best estimator =  2000 The train log loss is: 0.622559310952
9788
For values of best estimator =  2000 The cross validation log loss is: 1.1
398791801231054
For values of best estimator =  2000 The test log loss is: 1.1479508244242
453
```

TESTING WITH BEST HYPERPARAMETER VALUES

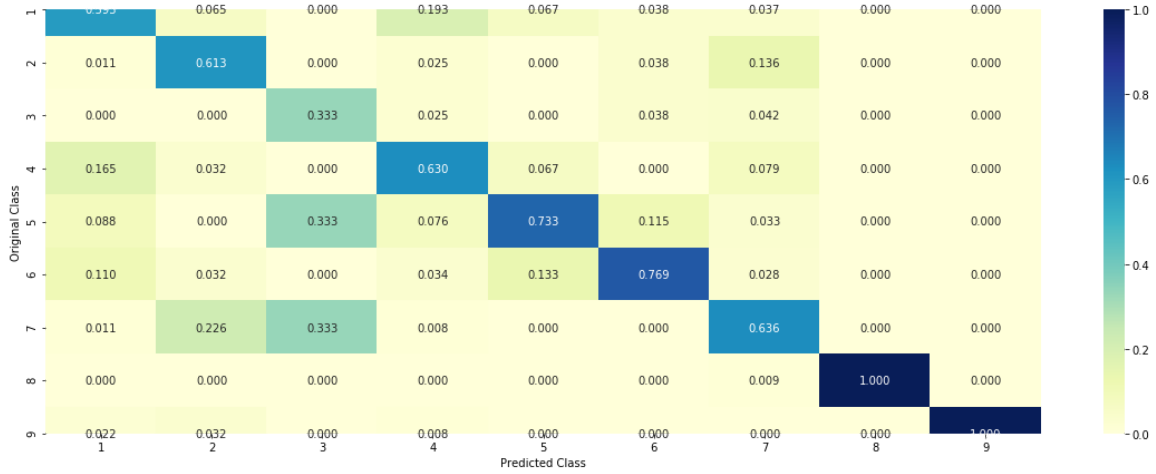
In [144]:

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

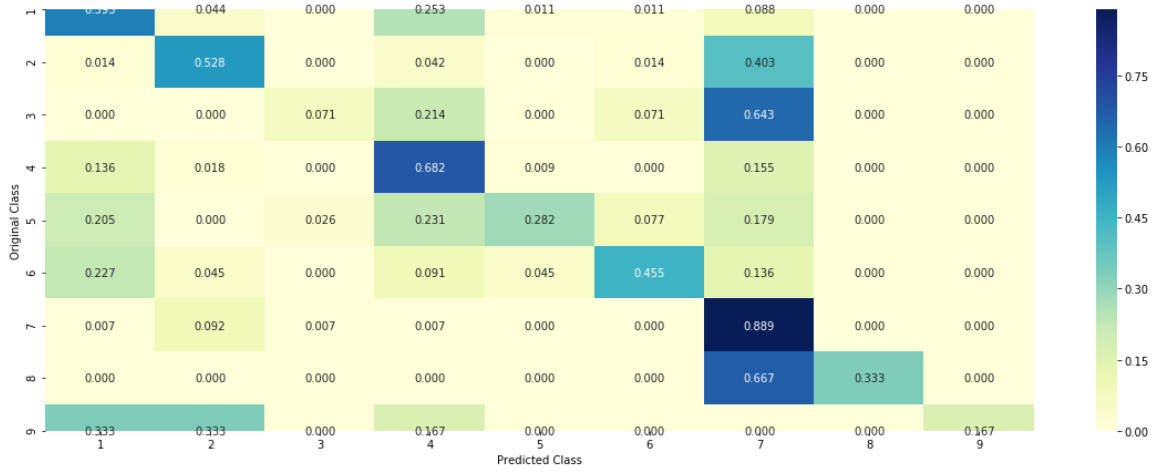
```
Log loss : 1.1398791801231054
Number of mis-classified points : 0.36654135338345867
-------------------- Confusion matrix --------------------
```




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




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



FEATURE IMPORTANCE FOR CORRECTLY CLASSIFIED POINT

In [145]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names_tf(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],te
st_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_fe
ature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1662 0.0467 0.1504 0.4667 0.0584 0.0447
0.0499 0.0069 0.0101]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [function] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
16 Text feature [growth] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
20 Text feature [missense] present in test data point [True]
22 Text feature [inhibition] present in test data point [True]
27 Text feature [downstream] present in test data point [True]
34 Text feature [constitutively] present in test data point [True]
38 Text feature [expressing] present in test data point [True]
39 Text feature [inhibited] present in test data point [True]
43 Text feature [serum] present in test data point [True]
44 Text feature [cells] present in test data point [True]
46 Text feature [variants] present in test data point [True]
49 Text feature [patients] present in test data point [True]
50 Text feature [protein] present in test data point [True]
51 Text feature [cell] present in test data point [True]
52 Text feature [pathogenic] present in test data point [True]
56 Text feature [functional] present in test data point [True]
57 Text feature [variant] present in test data point [True]
59 Text feature [amplification] present in test data point [True]
62 Text feature [phosphorylated] present in test data point [True]
65 Text feature [clinical] present in test data point [True]
71 Text feature [patient] present in test data point [True]
73 Text feature [proteins] present in test data point [True]
82 Text feature [neutral] present in test data point [True]
89 Text feature [lines] present in test data point [True]
95 Text feature [stimulation] present in test data point [True]
97 Text feature [predicted] present in test data point [True]
99 Text feature [months] present in test data point [True]
Out of the top  100  features  34 are present in query point
```

FEATURE IMPORTANCE FOR MISCLASSIFIED POINT

In [147]:

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

test_point_index = 25
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
ing[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names_tf(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],te
st_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_fe
ature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3641 0.0505 0.017  0.3846 0.0562 0.0474
0.0628 0.0065 0.0108]]
Actual Class : 1
--------------------------------------------------
1 Text feature [activating] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [activated] present in test data point [True]
6 Text feature [function] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
9 Text feature [treatment] present in test data point [True]
10 Text feature [suppressor] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
21 Text feature [activate] present in test data point [True]
22 Text feature [inhibition] present in test data point [True]
30 Text feature [stability] present in test data point [True]
31 Text feature [treated] present in test data point [True]
38 Text feature [expressing] present in test data point [True]
39 Text feature [inhibited] present in test data point [True]
44 Text feature [cells] present in test data point [True]
50 Text feature [protein] present in test data point [True]
51 Text feature [cell] present in test data point [True]
53 Text feature [defective] present in test data point [True]
56 Text feature [functional] present in test data point [True]
73 Text feature [proteins] present in test data point [True]
90 Text feature [functions] present in test data point [True]
92 Text feature [yeast] present in test data point [True]
96 Text feature [retained] present in test data point [True]
97 Text feature [predicted] present in test data point [True]
Out of the top  100  features  24 are present in query point
```

STACKING ALL THE MODELS

HYPERPARAMETER TUNING

In [148]:

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

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


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

sig_clf1.fit(train_x_tf, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba
(cv_x_tf))))
sig_clf2.fit(train_x_tf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pr
oba(cv_x_tf))))
sig_clf3.fit(train_x_tf, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tf
))))
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_classifi
er=lr, use_probas=True)
    sclf.fit(train_x_tf, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_l
oss(cv_y, sclf.predict_proba(cv_x_tf))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tf))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.16
Support vector machines : Log Loss: 1.55
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.713
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.314
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.217
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.518
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.947
```

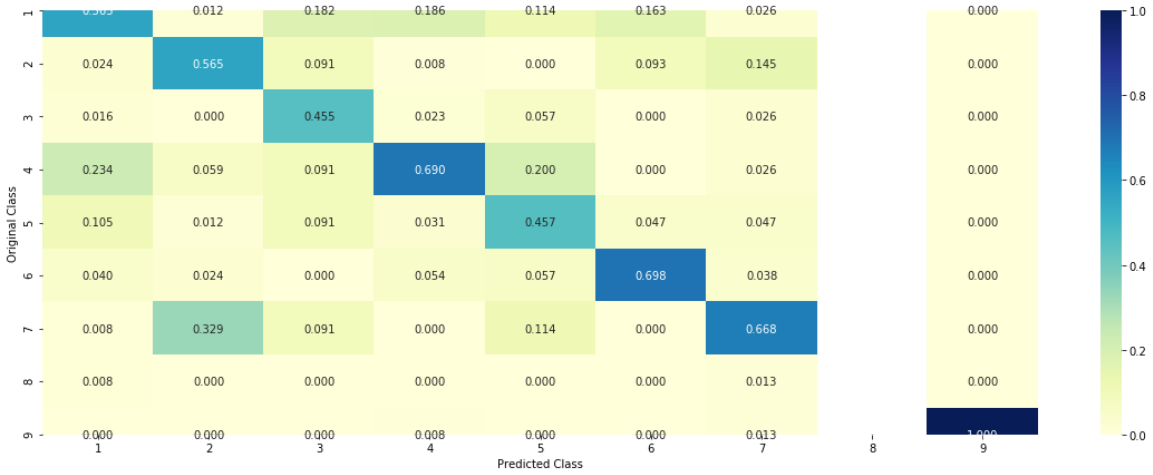
TESTING MODEL WITH BEST HYPERPARAMETER VALUES

In [149]:

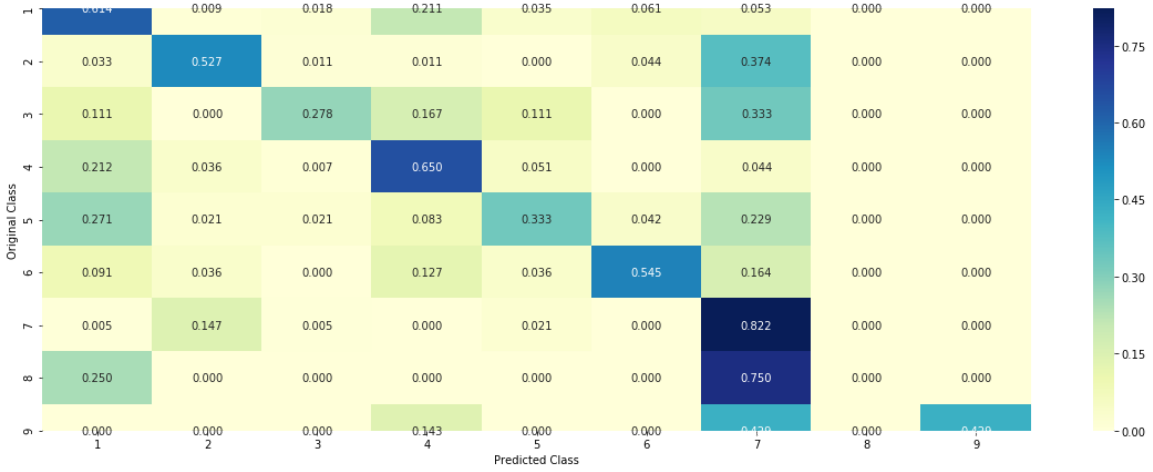
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
r, use_probas=True)
sclf.fit(train_x_tf, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.470185925287938
Log loss (CV) on the stacking classifier : 1.2169904468768782
Log loss (test) on the stacking classifier : 1.2357970868415415
Number of missclassified point : 0.37142857142857144
-------------------- Confusion matrix --------------------
```


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




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



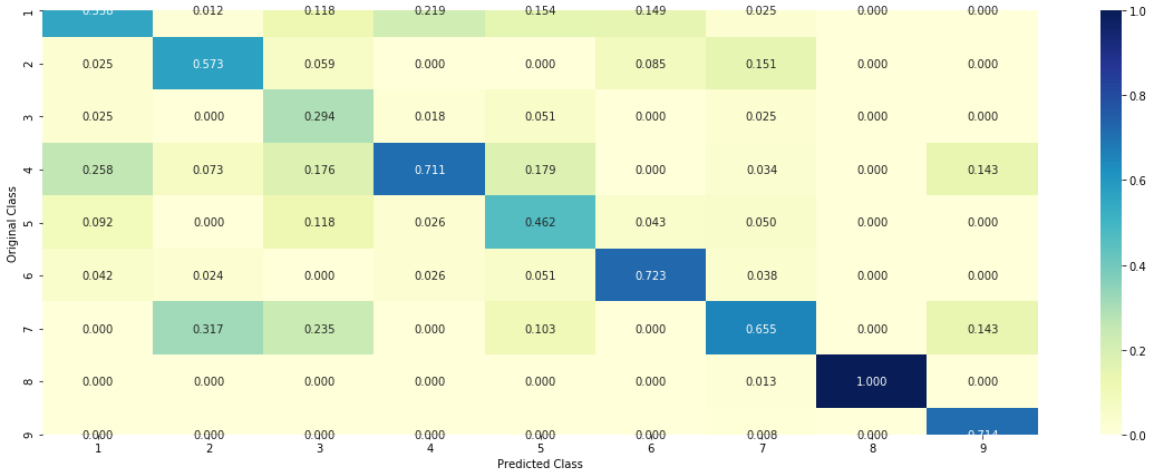
MAXIMUM VOTING CLASSIFIER

In [150]:

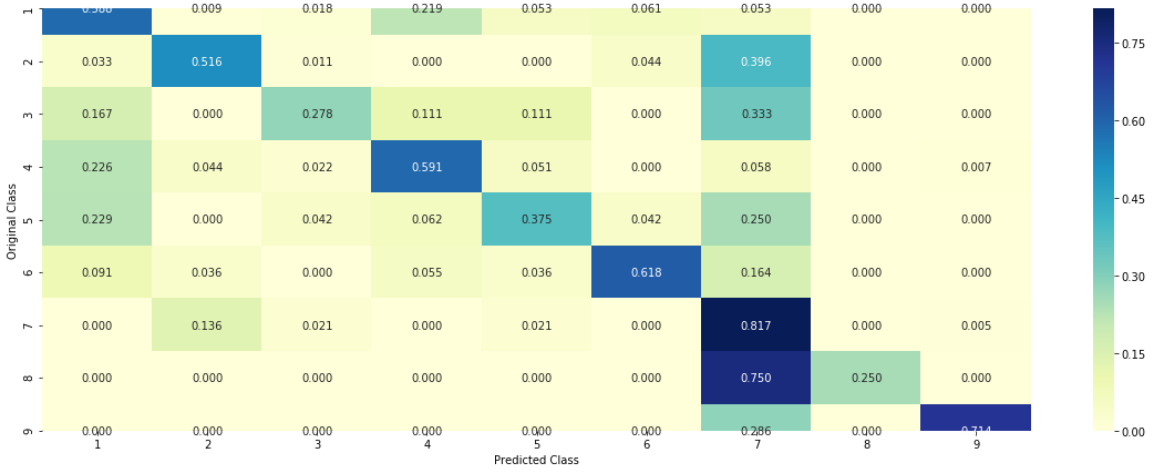
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_tf, train_y)
print("Log loss (train)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

```
Log loss (train) on the VotingClassifier : 0.7893180891877869
Log loss (CV) on the VotingClassifier : 1.1638614763895763
Log loss (test) on the VotingClassifier : 1.1807365073418046
Number of missclassified point : 0.3774436090225564
-------------------- Confusion matrix --------------------
```


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




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



RESULTS:

In [221]:

```
from prettytable import PrettyTable
tfidf_results=PrettyTable(['MODEL','TRAIN LOSS','CV LOSS','TEST LOSS','NO OF MISCLASSIF
IED POINTS'])
tfidf_results.add_row(['NAIVE_BAYES','0.7828','1.2373','1.29','0.3796'])
tfidf_results.add_row(['KNN','0.8394','1.0749','1.0908','0.3796'])
tfidf_results.add_row(['LR + CLASS BALANCING','0.6292','1.1679','1.1861','0.3703'])
tfidf_results.add_row(['LR WITHOUT CB','0.4746','1.1508','1.1109','0.3684'])
tfidf_results.add_row(['LR SVM','0.5175','1.1792','1.1478','0.3646'])
tfidf_results.add_row(['RANDOM FOREST','0.6225','1.1398','1.1479','0.3665'])
tfidf_results.add_row(['STACKING','0.4701','1.2169','1.2357','0.3714'])
tfidf_results.add_row(['MAX VOTING','0.7893','1.1638','1.1807','0.3774'])


print(tfidf_results.get_string(start=0,end=10))
```

```
+----------------------+------------+---------+-----------+--------------
-------------+
|        MODEL         | TRAIN LOSS | CV LOSS | TEST LOSS | NO OF MISCLASS
IFIED POINTS |
+----------------------+------------+---------+-----------+--------------
-------------+
|     NAIVE_BAYES      |   0.7828   |  1.2373 |    1.29   |           0.37
96           |
|         KNN          |   0.8394   |  1.0749 |   1.0908  |           0.37
96           |
| LR + CLASS BALANCING |   0.6292   |  1.1679 |   1.1861  |           0.37
03           |
|    LR WITHOUT CB     |   0.4746   |  1.1508 |   1.1109  |           0.36
84           |
|        LR SVM        |   0.5175   |  1.1792 |   1.1478  |           0.36
46           |
|    RANDOM FOREST     |   0.6225   |  1.1398 |   1.1479  |           0.36
65           |
|       STACKING       |   0.4701   |  1.2169 |   1.2357  |           0.37
14           |
|      MAX VOTING      |   0.7893   |  1.1638 |   1.1807  |           0.37
74           |
+----------------------+------------+---------+-----------+--------------
-------------+
```

In [ ]:

```

```

2 USING TOP 1000 WORDS

In [151]:

```
train_df.columns
```

Out[151]:

```
Index(['ID', 'Gene', 'Variation', 'Class', 'TEXT'], dtype='object')
```

In [152]:

```
vectorizer=TfidfVectorizer(use_idf=True)
tfidf=vectorizer.fit_transform(train_df['TEXT'].values)
idf_values=vectorizer.idf_
print(len(idf_values))
```

```
128283
```

In [153]:

```
idf_values_sorted=sorted(idf_values,reverse=True)
idf_values_sorted[0:5]
```

Out[153]:

```
[7.968379900798572,
 7.968379900798572,
 7.968379900798572,
 7.968379900798572,
 7.968379900798572]
```

In [154]:

```
ind=np.argsort(vectorizer.idf_)[ ::-1 ]
features=vectorizer.get_feature_names()
```

In [155]:

```
#taking top 1000 words
n=1000
topFeatures=[features[i] for i in ind[:n] ]
topIDF=[(vectorizer.idf_)[i] for i in ind[:n] ]
featIDF=dict( zip(topFeatures, topIDF) )

print(topFeatures[0:10])
```

```
['marcy', '639152', 'lmmunoblot', 'lmn', 'lmnantrk1', 'castellana', 'astro
cytomas27', '639241', '6390', 'llow']
```

VECTORIZING THE TEXT FEATURE USING TOP 1000 WORDS

In [156]:

```
from tqdm import tqdm
```

In [157]:

```
avgw2v_idf_train = []
for sentence in tqdm(train_df['TEXT']):
    vector = np.zeros(250)
    cnt_words =0;
    for word in sentence.split():
        if word in featIDF:
            vector += featIDF[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avgw2v_idf_train.append(vector)

print(len(avgw2v_idf_train))
```

```
100%|████████████████████████████████████████████████████████████████████████████
███████| 2124/2124 [00:02<00:00, 1032.64it/s]

2124
```

In [158]:

```
print(len(avgw2v_idf_train[0]))
```

```
250
```

In [159]:

```
cv_df.shape
```

Out[159]:

```
(532, 5)
```

In [160]:

```
avgw2v_idf_cv = []
for sentence in tqdm(cv_df['TEXT']):
    vector = np.zeros(250)
    cnt_words =0;
    for word in sentence.split():
        if word in featIDF:
            vector += featIDF[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avgw2v_idf_cv.append(vector)

print(len(avgw2v_idf_cv))
```

```
100%|████████████████████████████████████████████████████████████████████████████
████████| 532/532 [00:00<00:00, 1075.44it/s]

532
```

In [161]:

```
avgw2v_idf_test = []
for sentence in tqdm(test_df['TEXT']):
    vector = np.zeros(250)
    cnt_words =0;
    for word in sentence.split():
        if word in featIDF:
            vector += featIDF[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avgw2v_idf_test.append(vector)

print(len(avgw2v_idf_test))
```

```
100%|██████████| 665/665 [00:00<00:00, 1008.55it/s]

665
```

In [162]:

```
#preparing the data matrix
train_top=hstack((train_gene_var_tf,avgw2v_idf_train)).tocsr()
cv_top=hstack((cv_gene_var_tf,avgw2v_idf_cv)).tocsr()
test_top=hstack((test_gene_var_tf,avgw2v_idf_test)).tocsr()
print(train_top.shape)
print(cv_top.shape)
print(test_top.shape)
```

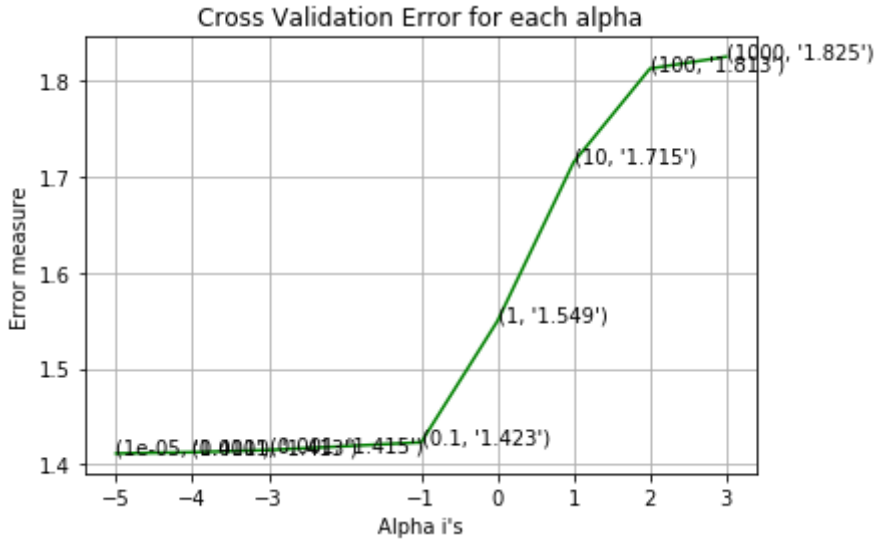
```
(2124, 2420)
(532, 2420)
(665, 2420)
```

MODEL:1 NAIVE BAYES(HYPERPARAMETER TUNING)

In [163]:

```
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_top, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_top, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_top)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    # to avoid rounding error while multiplying probabilites we use log-probability est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 1.4113609260276763
for alpha = 0.0001
Log Loss : 1.4126765866364355
for alpha = 0.001
Log Loss : 1.414948950990961
for alpha = 0.1
Log Loss : 1.4229880719966268
for alpha = 1
Log Loss : 1.5494674684820209
for alpha = 10
Log Loss : 1.7153101154313684
for alpha = 100
Log Loss : 1.8128527971895423
for alpha = 1000
Log Loss : 1.8252599064780615
```



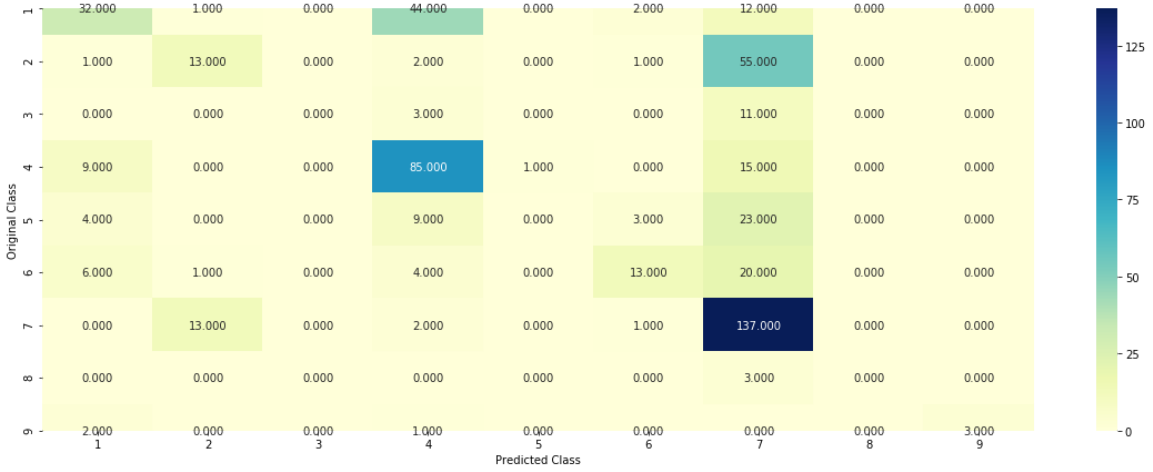

```
For values of best alpha =  1e-05 The train log loss is: 0.748573439629009
1
For values of best alpha =  1e-05 The cross validation log loss is: 1.4113
609260276763
For values of best alpha =  1e-05 The test log loss is: 1.394760552589141
```

TESTING WITH BEST HYPERPARAMETER VALUES

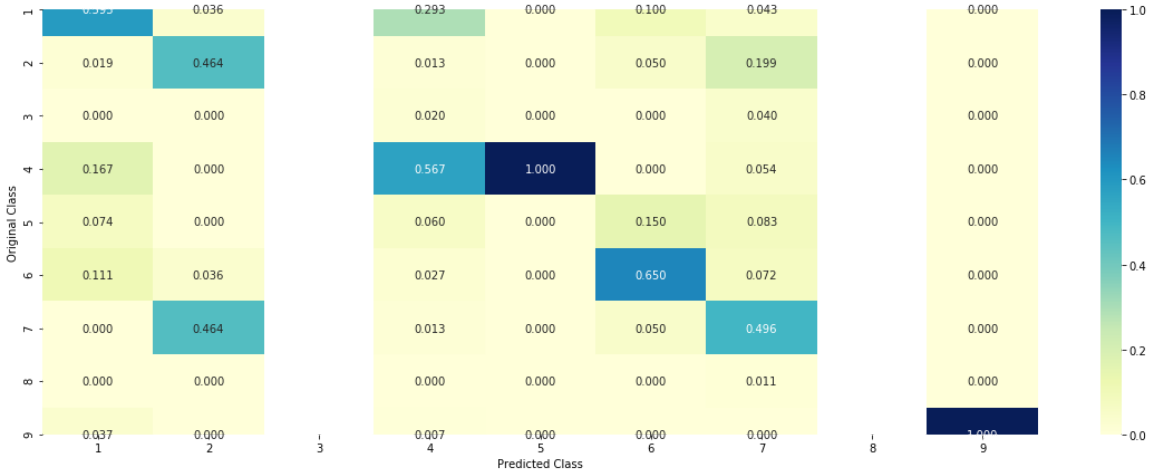
In [164]:

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

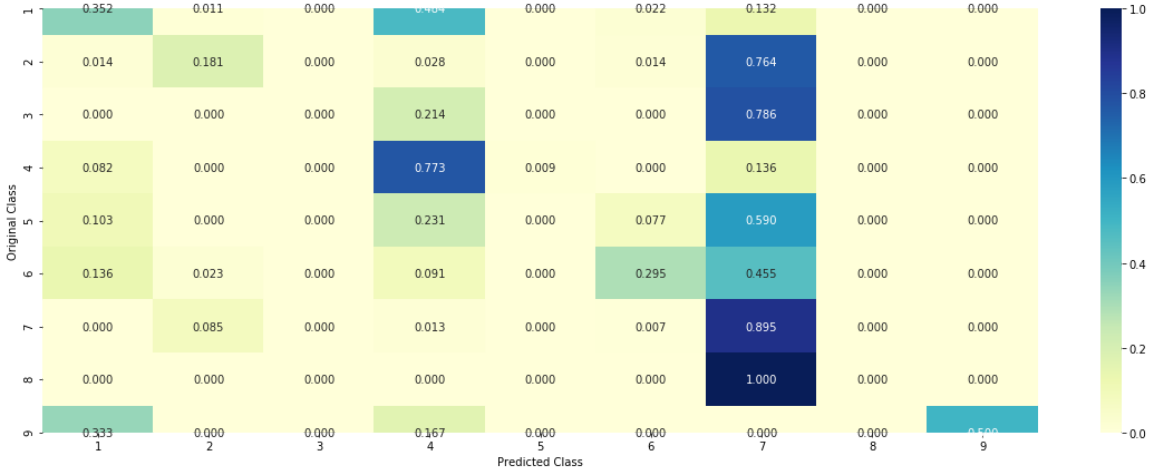
```
Log Loss : 1.4113609260276763
Number of missclassified point : 0.4680451127819549
-------------------- Confusion matrix --------------------
```




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




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

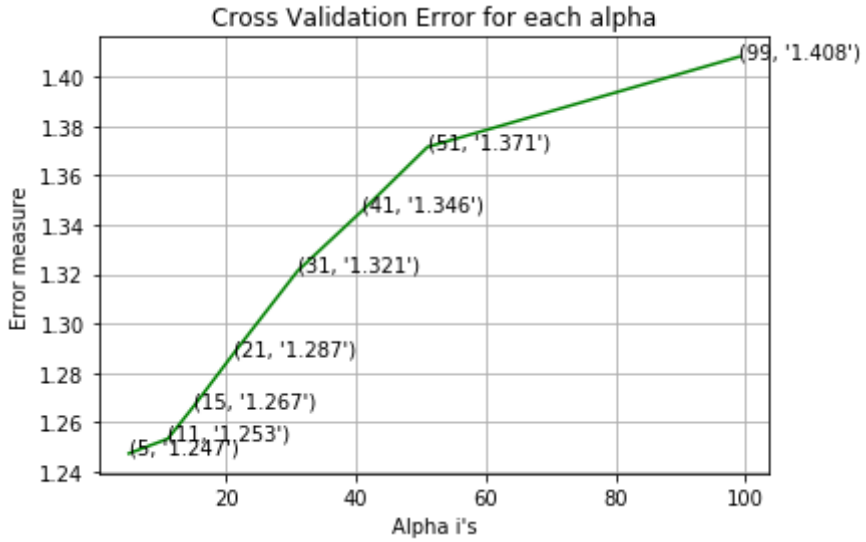


MODEL:2 KNN(HYPERPARAMETER TUNING)

In [165]:

```
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_top, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_top, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_top)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    # to avoid rounding error while multiplying probabilites we use log-probability est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 1.2473794770044497
for alpha = 11
Log Loss : 1.253326079735965
for alpha = 15
Log Loss : 1.2665485129704117
for alpha = 21
Log Loss : 1.2874166868628194
for alpha = 31
Log Loss : 1.3213040516914698
for alpha = 41
Log Loss : 1.3458034315902576
for alpha = 51
Log Loss : 1.3714662073316128
for alpha = 99
Log Loss : 1.4080285276568465
```




```
For values of best alpha =  5 The train log loss is: 0.9678649155320384
For values of best alpha =  5 The cross validation log loss is: 1.24737947
70044497
For values of best alpha =  5 The test log loss is: 1.2453965175727355
```

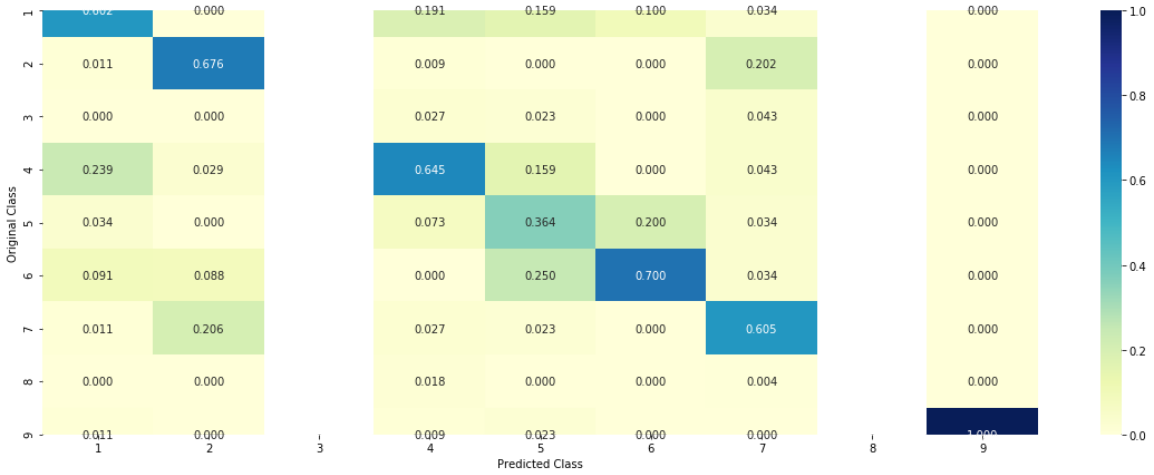
TESTING KNN WITH BEST HYPERPARAMETER VALUES

In [166]:

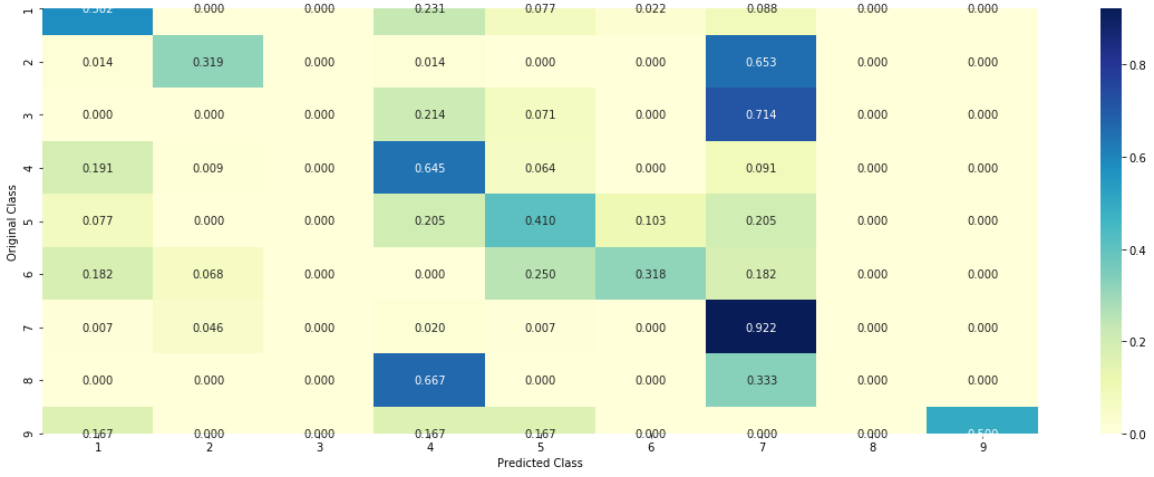
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_top, train_y, cv_top, cv_y, clf)
```

```
Log loss : 1.2473794770044497
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```


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




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

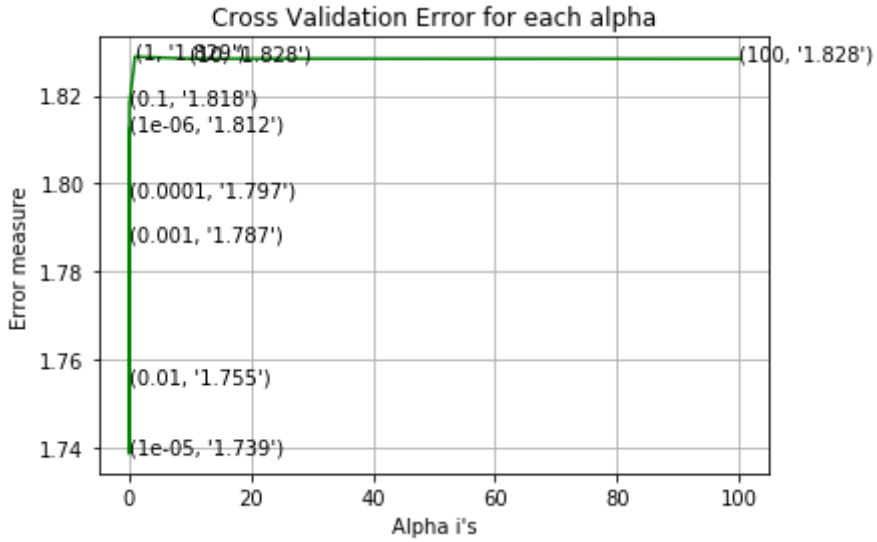


MODEL:3 LOGISTIC REGRESSION WITH BALANCED CLASS WEIGHT

In [167]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.8122591927849656
for alpha = 1e-05
Log Loss : 1.7386151238998704
for alpha = 0.0001
Log Loss : 1.7969337141579105
for alpha = 0.001
Log Loss : 1.7871352702124352
for alpha = 0.01
Log Loss : 1.7546255310881496
for alpha = 0.1
Log Loss : 1.8181198470601732
for alpha = 1
Log Loss : 1.8288246562387405
for alpha = 10
Log Loss : 1.8283824867623157
for alpha = 100
Log Loss : 1.8283493923532106
```



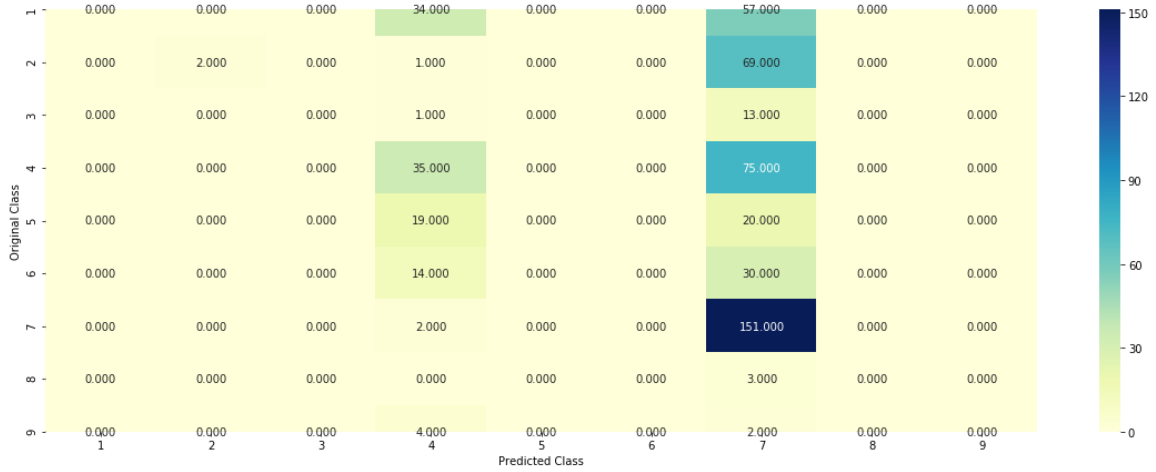

```
For values of best alpha =  1e-05 The train log loss is: 1.690783480853124
6
For values of best alpha =  1e-05 The cross validation log loss is: 1.7386
151238998704
For values of best alpha =  1e-05 The test log loss is: 1.7410556038932299
```

TESTING WITH BEST HYPERPARAMETERS

In [168]:

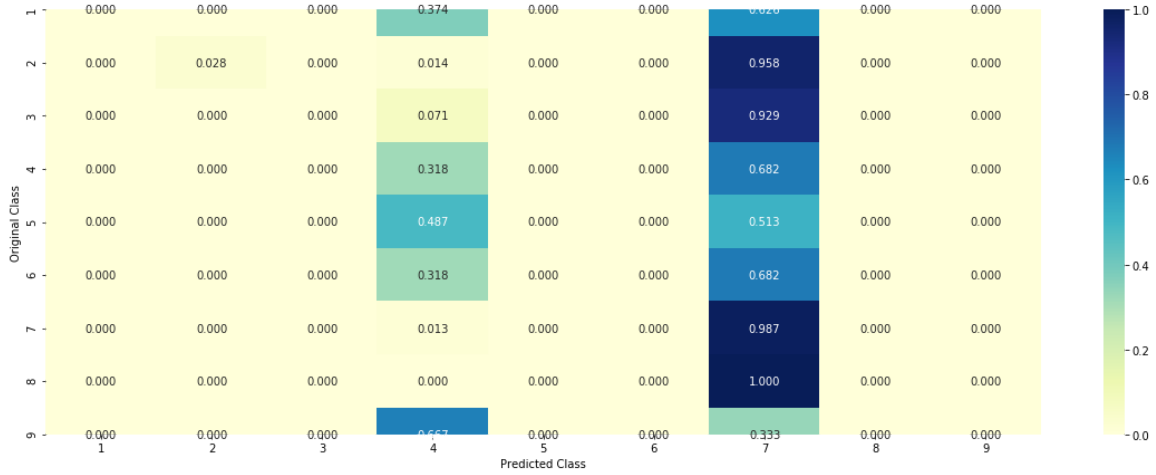
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
s='log', random_state=42)
predict_and_plot_confusion_matrix(train_top, train_y, cv_top, cv_y, clf)
```

```
Log loss : 1.7386151238998704
Number of mis-classified points : 0.6466165413533834
-------------------- Confusion matrix --------------------
```




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


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

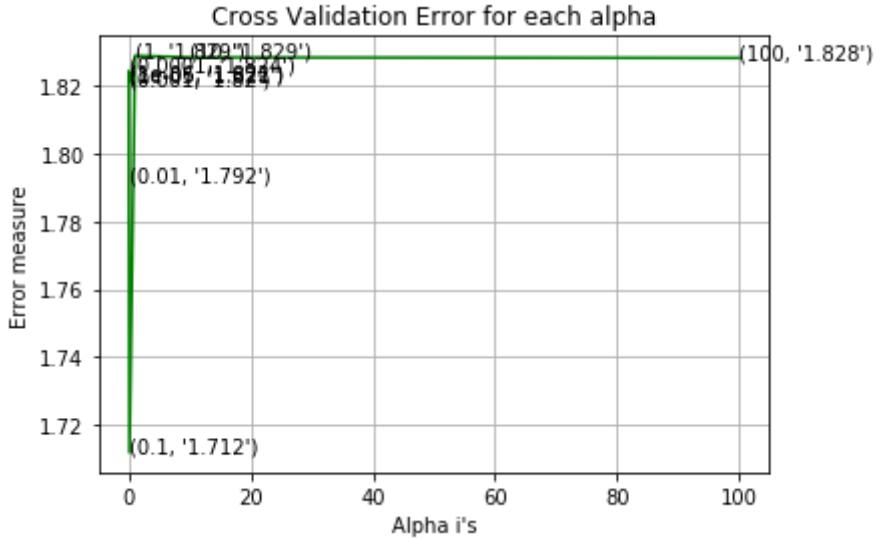


LOGISTIC REGRESSION WITHOUT BALANCED CLASS WEIGHT

In [169]:

```
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_top, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_top, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_top)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    # to avoid rounding error while multiplying probabilites we use log-probability est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.8222894301363577
for alpha = 1e-05
Log Loss : 1.8214110700400734
for alpha = 0.0001
Log Loss : 1.824497180108406
for alpha = 0.001
Log Loss : 1.820327876693684
for alpha = 0.01
Log Loss : 1.7922409094763814
for alpha = 0.1
Log Loss : 1.711694299084341
for alpha = 1
Log Loss : 1.8290287793851379
for alpha = 10
Log Loss : 1.8285327109435723
for alpha = 100
Log Loss : 1.8283624727789396
```




```
For values of best alpha =  0.1 The train log loss is: 1.675354686496457
For values of best alpha =  0.1 The cross validation log loss is: 1.711694
299084341
For values of best alpha =  0.1 The test log loss is: 1.7016048692081567
```

TESTING WITH BEST HYPERPARAMETERS

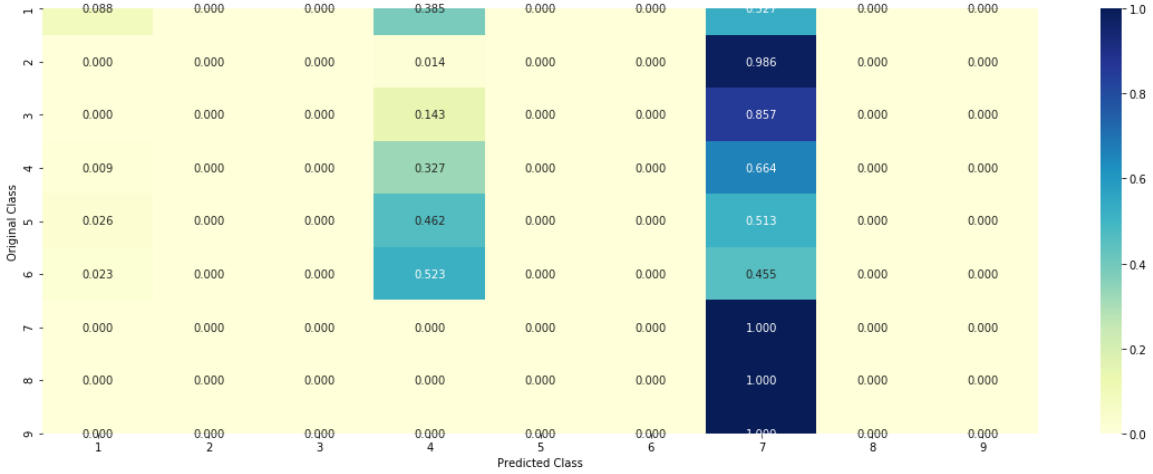
In [170]:

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

```
Log loss : 1.711694299084341
Number of mis-classified points : 0.6296992481203008
-------------------- Confusion matrix --------------------
```


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


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

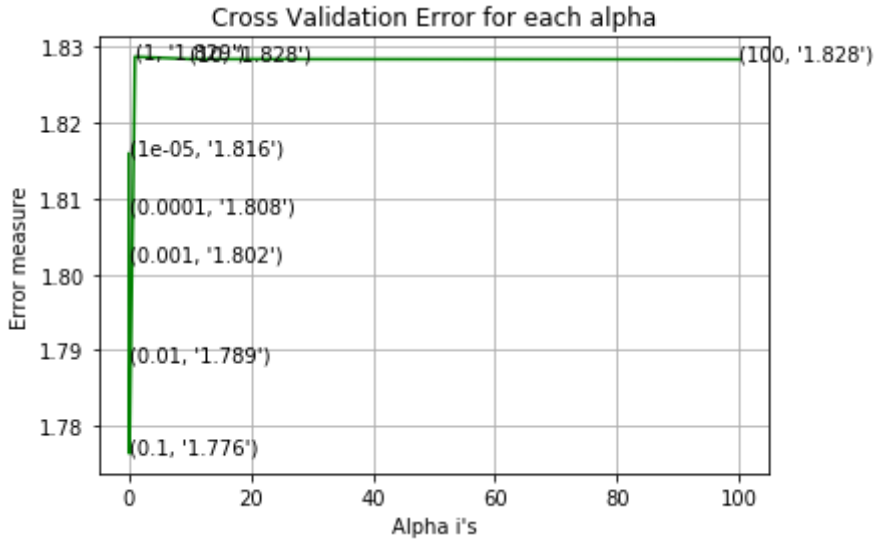


MODEL:4 LINEAR SVM(HYPERPARMETER TUNING)

In [171]:

```
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_top, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_top, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_top)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for C = 1e-05
Log Loss : 1.8159157409178894
for C = 0.0001
Log Loss : 1.8081147828483684
for C = 0.001
Log Loss : 1.8017635032032095
for C = 0.01
Log Loss : 1.7886143167713597
for C = 0.1
Log Loss : 1.7763760067026568
for C = 1
Log Loss : 1.8286998443833562
for C = 10
Log Loss : 1.8283936638496083
for C = 100
Log Loss : 1.828351150388183
```



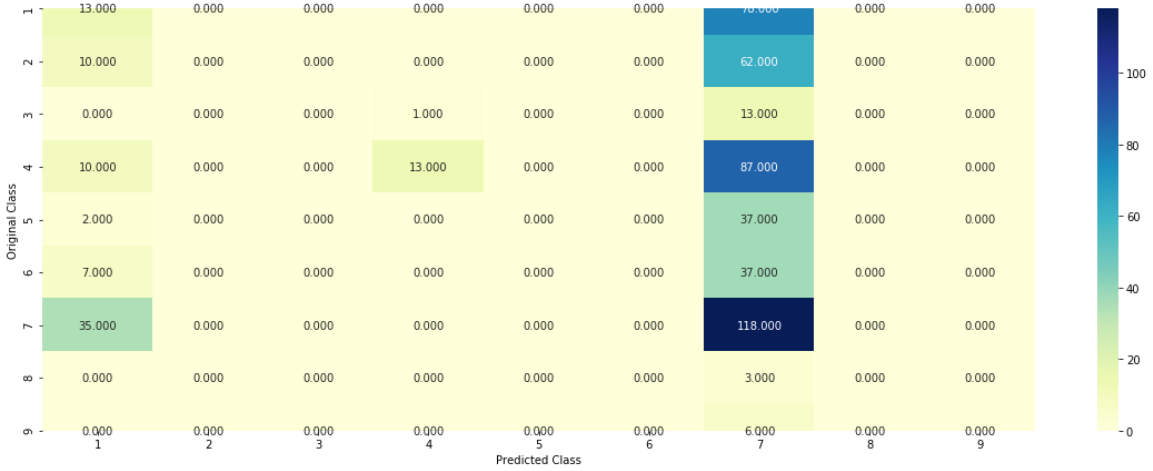

```
For values of best alpha =  0.1 The train log loss is: 1.7365422200835416
For values of best alpha =  0.1 The cross validation log loss is: 1.776376
0067026568
For values of best alpha =  0.1 The test log loss is: 1.758743844500128
```

TESTING WITH BEST HYPERPARAMETERS

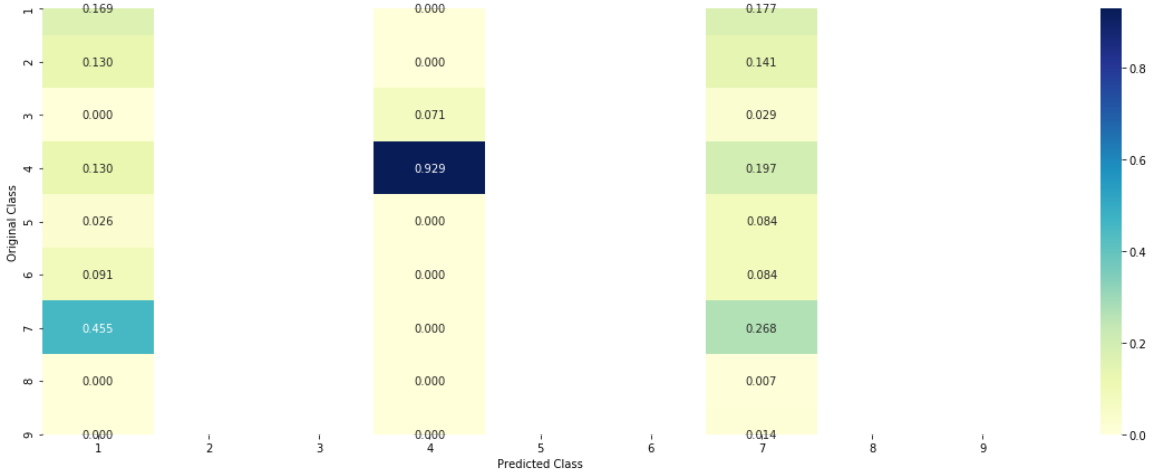
In [172]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
2,class_weight='balanced')
predict_and_plot_confusion_matrix(train_top, train_y,cv_top,cv_y, clf)
```

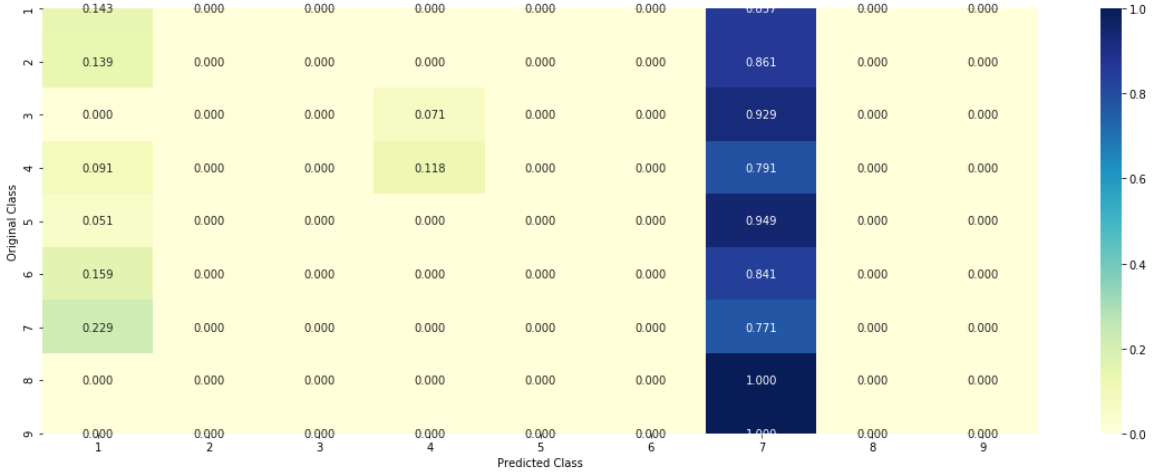
```
Log loss : 1.7763760067026568
Number of mis-classified points : 0.7293233082706767
-------------------- Confusion matrix --------------------
```




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




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



MODEL:5 RANDOM FOREST CLASSIFIER(HYPERPARAMETER TUNING)

In [173]:

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

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

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

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

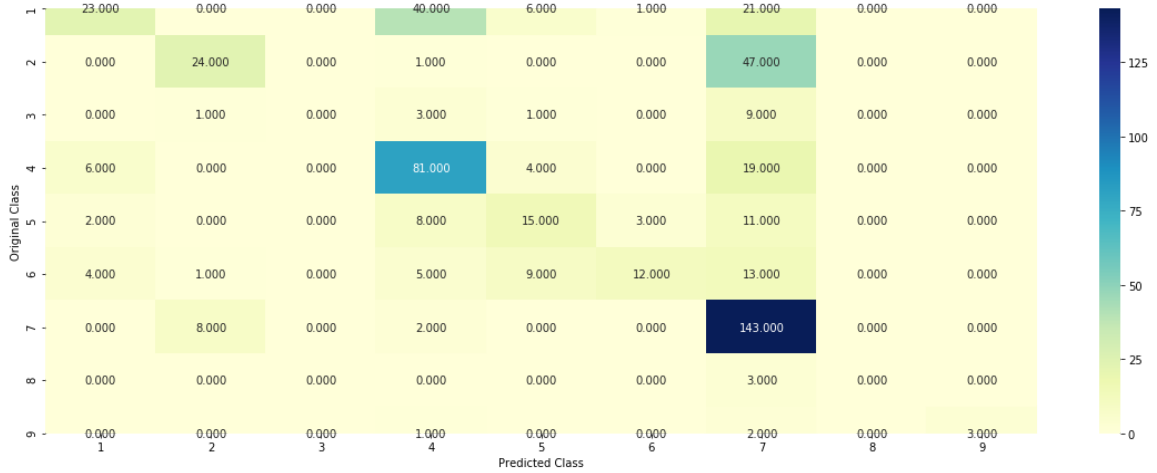
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.4991325709466214
for n_estimators = 100 and max depth =  10
Log Loss : 1.3806287035921145
for n_estimators = 200 and max depth =  5
Log Loss : 1.4900546225657132
for n_estimators = 200 and max depth =  10
Log Loss : 1.3749166562583988
for n_estimators = 500 and max depth =  5
Log Loss : 1.4827372608863887
for n_estimators = 500 and max depth =  10
Log Loss : 1.3674637267280334
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4799845946031727
for n_estimators = 1000 and max depth =  10
Log Loss : 1.3635020141991905
for n_estimators = 2000 and max depth =  5
Log Loss : 1.4779804599942081
for n_estimators = 2000 and max depth =  10
Log Loss : 1.361667953469927
For values of best estimator =  2000 The train log loss is: 1.057718178972
9555
For values of best estimator =  2000 The cross validation log loss is: 1.3
616679534699272
For values of best estimator =  2000 The test log loss is: 1.3563656384017
704
```

TESTING WITH BEST HYPERPARAMETER VALUES

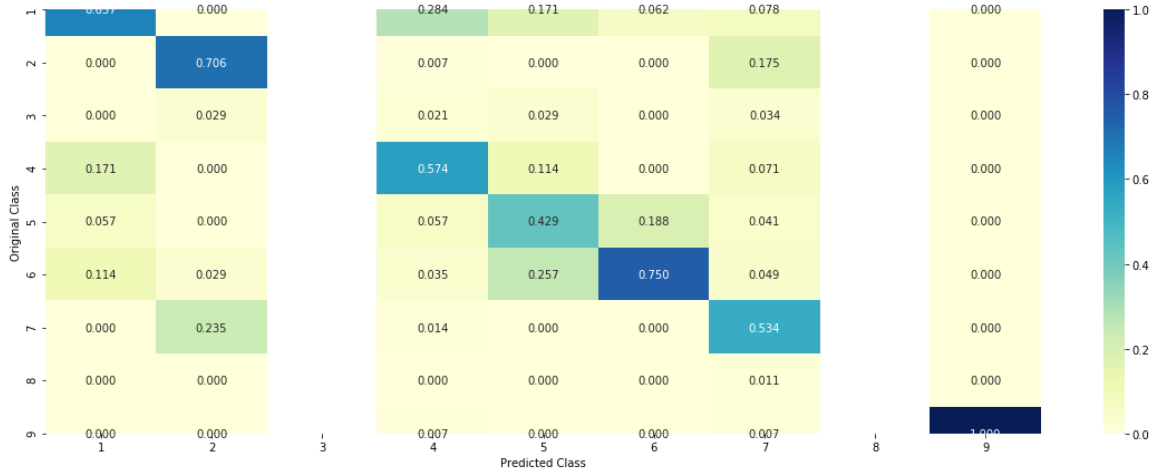
In [174]:

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

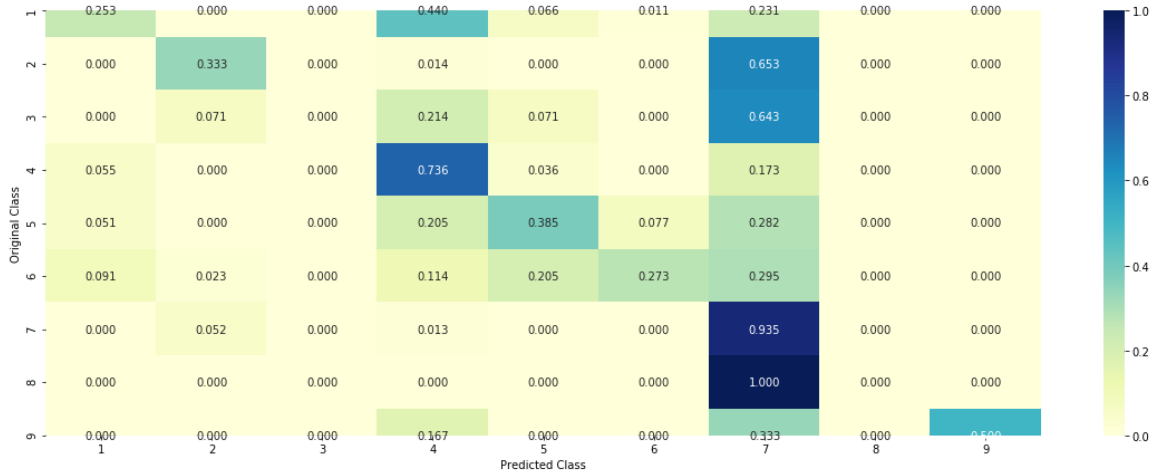
```
Log loss : 1.3616679534699272
Number of mis-classified points : 0.4342105263157895
-------------------- Confusion matrix --------------------
```




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




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



STACKING ALL MODELS(HYPERPARAMETER TUNING)

In [175]:

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

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


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

sig_clf1.fit(train_top, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba
(cv_top))))
sig_clf2.fit(train_top, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pr
oba(cv_top))))
sig_clf3.fit(train_top, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_top
))))
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_classifi
er=lr, use_probas=True)
    sclf.fit(train_top, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_l
oss(cv_y, sclf.predict_proba(cv_top))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_top))
    if best_alpha > log_error:
        best_alpha = log_error
```

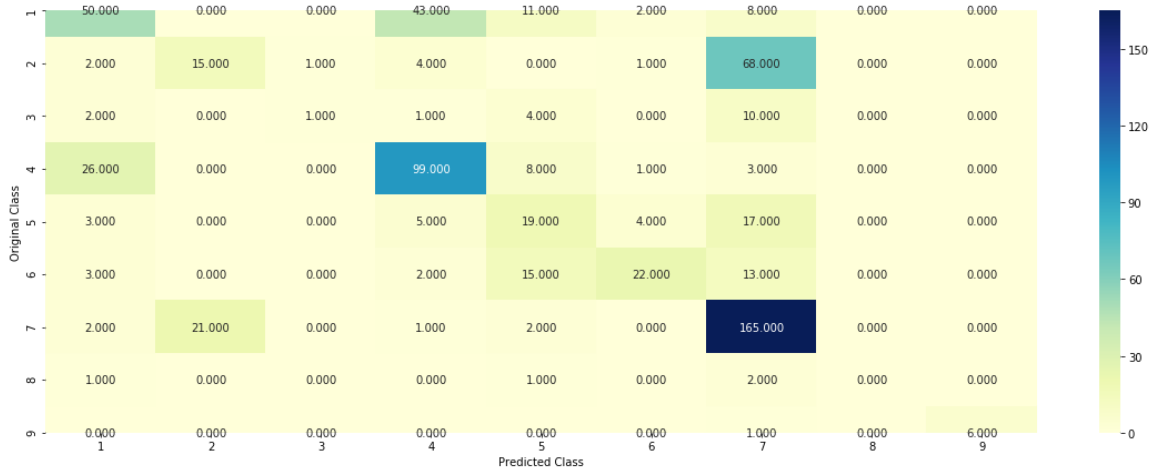
```
Logistic Regression :  Log Loss: 1.77
Support vector machines : Log Loss: 1.83
Naive Bayes : Log Loss: 1.41
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.828
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.808
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.663
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.422
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.524
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.770
```

TESTING WITH BEST HYPERPARAMETERS

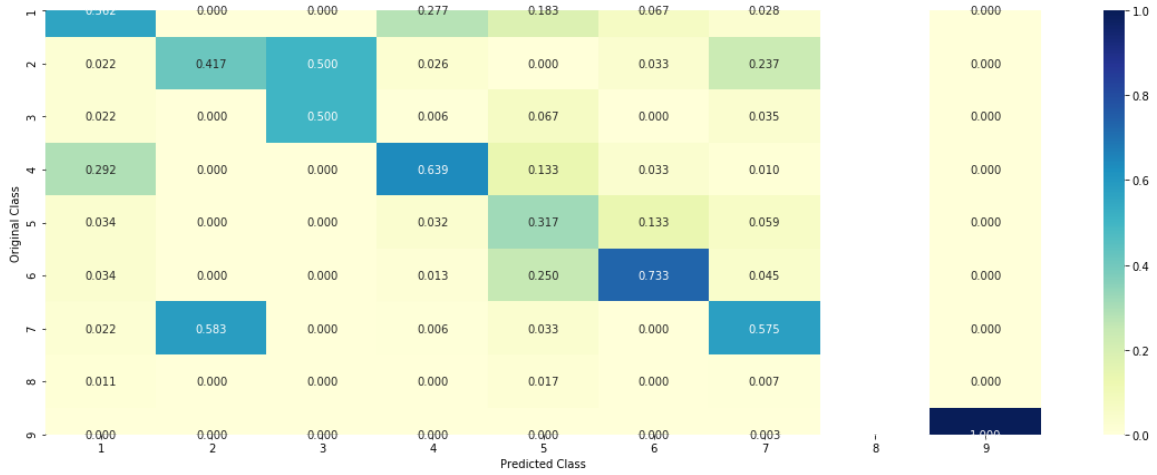
In [176]:

```
lr = LogisticRegression(C=1)

sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
r, use_probas=True)
sclf.fit(train_top, train_y)

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

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

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

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

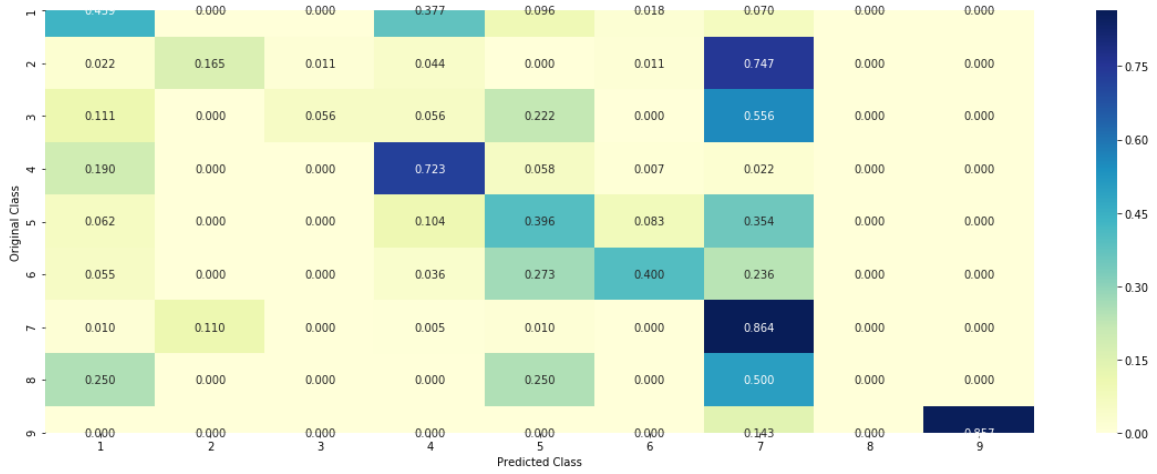
```
Log loss (train) on the stacking classifier : 0.25904623402556054
Log loss (CV) on the stacking classifier : 1.524074093472935
Log loss (test) on the stacking classifier : 1.4964871487192075
Number of missclassified point : 0.4330827067669173
-------------------- Confusion matrix --------------------
```




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




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

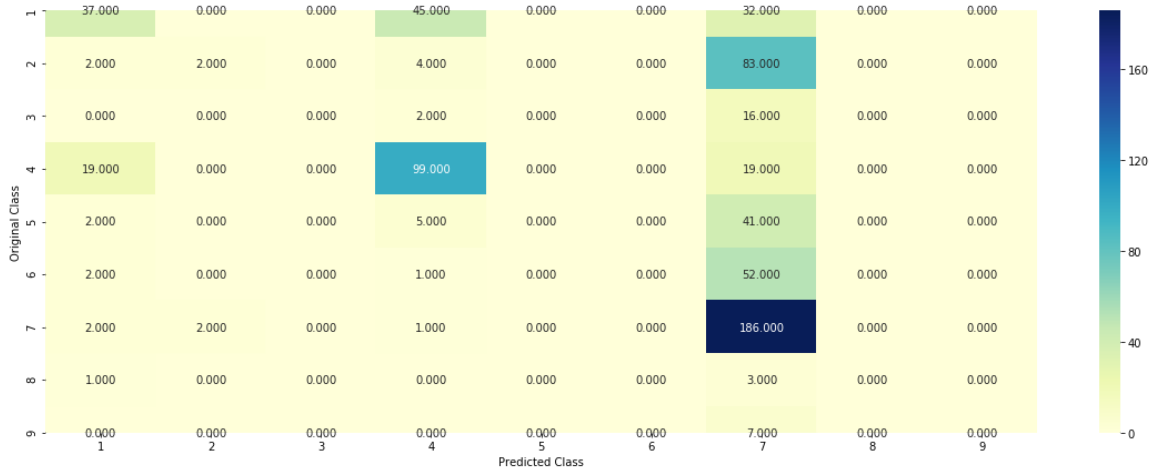


MAJORITY VOTE CLASSIFIER

In [177]:

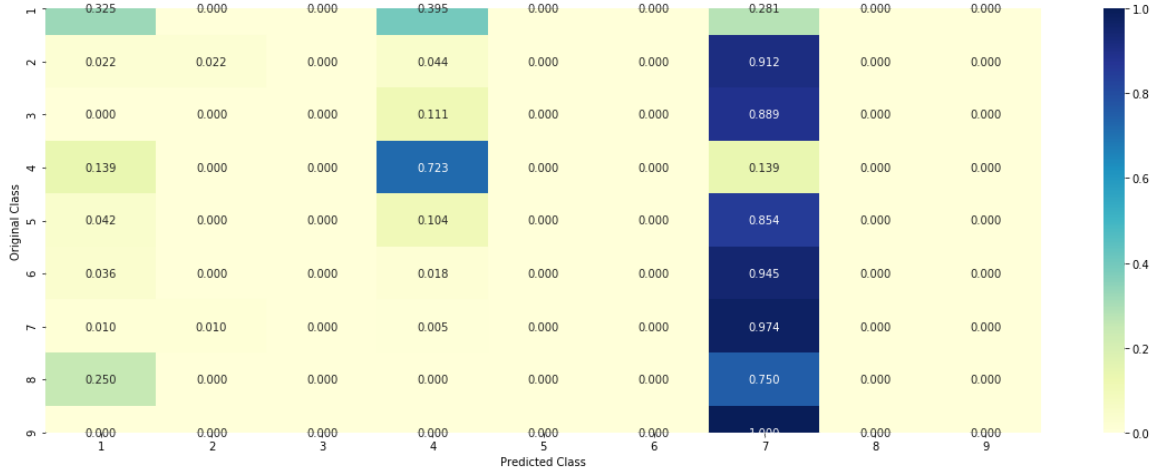
```
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf
3)], voting='soft')
vclf.fit(train_top, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_prob
a(train_top)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_t
op)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(
test_top)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_top)- tes
t_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_top))
```

```
Log loss (train) on the VotingClassifier : 1.3064587461158526
Log loss (CV) on the VotingClassifier : 1.6139657976363797
Log loss (test) on the VotingClassifier : 1.5988380492728378
Number of missclassified point : 0.512781954887218
-------------------- Confusion matrix --------------------
```




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


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



RESULTS:

In [222]:

```
from prettytable import PrettyTable
tfidf_results=PrettyTable(['MODEL','TRAIN LOSS','CV LOSS','TEST LOSS','NO OF MISCLASSIF
IED POINTS'])
tfidf_results.add_row(['NAIVE_BAYES','0.7485','1.4113','1.3947','0.4680'])
tfidf_results.add_row(['KNN','0.9678','1.2473','1.2453','0.3966'])
tfidf_results.add_row(['LR + CLASS BALANCING','1.6907','1.7386','1.7410','0.6466'])
tfidf_results.add_row(['LR WITHOUT CB','1.6753','1.7116','1.7016','0.6296'])
tfidf_results.add_row(['LR SVM','1.7365','1.7763','1.7587','0.7293'])
tfidf_results.add_row(['RANDOM FOREST','1.0577','1.3616','1.3563','0.4342'])
tfidf_results.add_row(['STACKING','0.2590','1.5240','1.4964','0.4330'])
tfidf_results.add_row(['MAX VOTING','1.3064','1.6139','1.5988','0.5127'])


print(tfidf_results.get_string(start=0,end=10))
```

```
+----------------------+------------+---------+-----------+--------------
-------------+
|        MODEL         | TRAIN LOSS | CV LOSS | TEST LOSS | NO OF MISCLASS
IFIED POINTS |
+----------------------+------------+---------+-----------+--------------
-------------+
|     NAIVE_BAYES      |   0.7485   |  1.4113 |   1.3947  |           0.46
80           |
|         KNN          |   0.9678   |  1.2473 |   1.2453  |           0.39
66           |
| LR + CLASS BALANCING |   1.6907   |  1.7386 |   1.7410  |           0.64
66           |
|    LR WITHOUT CB     |   1.6753   |  1.7116 |   1.7016  |           0.62
96           |
|        LR SVM        |   1.7365   |  1.7763 |   1.7587  |           0.72
93           |
|    RANDOM FOREST     |   1.0577   |  1.3616 |   1.3563  |           0.43
42           |
|       STACKING       |   0.2590   |  1.5240 |   1.4964  |           0.43
30           |
|      MAX VOTING      |   1.3064   |  1.6139 |   1.5988  |           0.51
27           |
+----------------------+------------+---------+-----------+--------------
-------------+
```

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

In [183]:

```
vectorizer=CountVectorizer(min_df=1,ngram_range=(1, 2))
vectorizer.fit(train_df['TEXT'].values)

X_train_encoded=vectorizer.transform(train_df['TEXT'].values)
X_cv_encoded=vectorizer.transform(cv_df['TEXT'].values)
X_test_encoded=vectorizer.transform(test_df['TEXT'].values)
print(X_train_encoded.shape)
print(X_cv_encoded.shape)
print(X_test_encoded.shape)
```

```
(2124, 2378985)
(532, 2378985)
(665, 2378985)
```

In [178]:

```
test_df.columns
```

Out[178]:

```
Index(['ID', 'Gene', 'Variation', 'Class', 'TEXT'], dtype='object')
```

In [179]:

```
vectorizer=CountVectorizer(min_df=1,ngram_range=(1, 2))
vectorizer.fit(train_df['Gene'].values)

X_train_gene_encoded=vectorizer.transform(train_df['Gene'].values)
X_cv_gene_encoded=vectorizer.transform(cv_df['Gene'].values)
X_test_gene_encoded=vectorizer.transform(test_df['Gene'].values)
print(X_train_gene_encoded.shape)
print(X_cv_gene_encoded.shape)
print(X_test_gene_encoded.shape)
```

```
(2124, 229)
(532, 229)
(665, 229)
```

In [180]:

```
vectorizer=CountVectorizer(min_df=1,ngram_range=(1, 2))
vectorizer.fit(train_df['Variation'].values)

X_train_var_encoded=vectorizer.transform(train_df['Variation'].values)
X_cv_var_encoded=vectorizer.transform(cv_df['Variation'].values)
X_test_var_encoded=vectorizer.transform(test_df['Variation'].values)
print(X_train_var_encoded.shape)
print(X_cv_var_encoded.shape)
print(X_test_var_encoded.shape)
```

```
(2124, 2033)
(532, 2033)
(665, 2033)
```

In [184]:

```
#creating final data matrix
train_3=hstack((X_train_encoded,X_train_gene_encoded,X_train_var_encoded)).tocsr()
cv_3=hstack((X_cv_encoded,X_cv_gene_encoded,X_cv_var_encoded)).tocsr()
test_3=hstack((X_test_encoded,X_test_gene_encoded,X_test_var_encoded)).tocsr()
print(train_3.shape)
print(cv_3.shape)
print(test_3.shape)
```

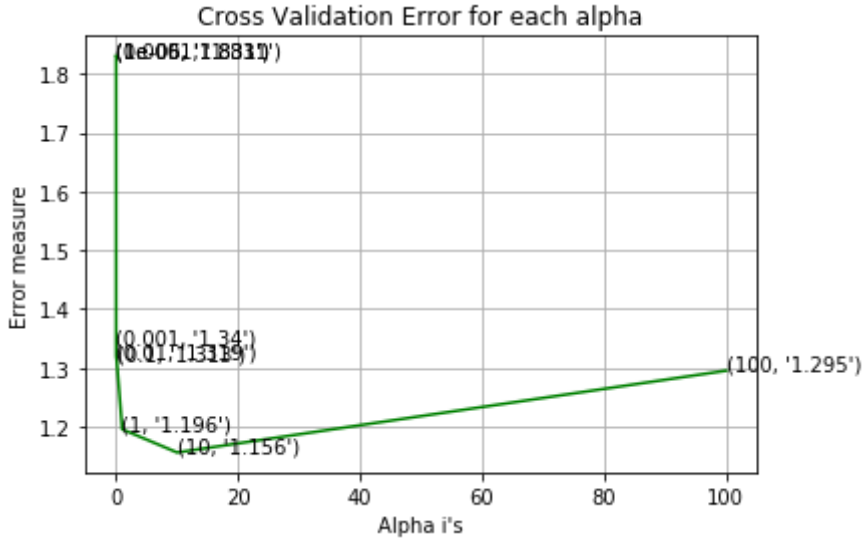
```
(2124, 2381247)
(532, 2381247)
(665, 2381247)
```

APPLYING LOGISTIC REGRESSION(HYPERPARAMETER TUNING)

In [185]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ran
dom_state=42)
    clf.fit(train_3, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_3, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_3)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    # to avoid rounding error while multiplying probabilites we use log-probability est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
s='log', random_state=42)
clf.fit(train_3, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_3, train_y)

predict_y = sig_clf.predict_proba(train_3)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_3)
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_3)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.8308894335840866
for alpha = 1e-05
Log Loss : 1.8308894335840866
for alpha = 0.0001
Log Loss : 1.8308894335840866
for alpha = 0.001
Log Loss : 1.3403574604114323
for alpha = 0.01
Log Loss : 1.3191812504907348
for alpha = 0.1
Log Loss : 1.312993808013908
for alpha = 1
Log Loss : 1.1957216617007367
for alpha = 10
Log Loss : 1.1562598627730973
for alpha = 100
Log Loss : 1.2954065483129464
```




```
For values of best alpha =  10 The train log loss is: 0.8219576049623742
For values of best alpha =  10 The cross validation log loss is: 1.1562598
627730973
For values of best alpha =  10 The test log loss is: 1.1876070391392861
```

TESTING WITH BEST HYPERPARAMETER VALUES

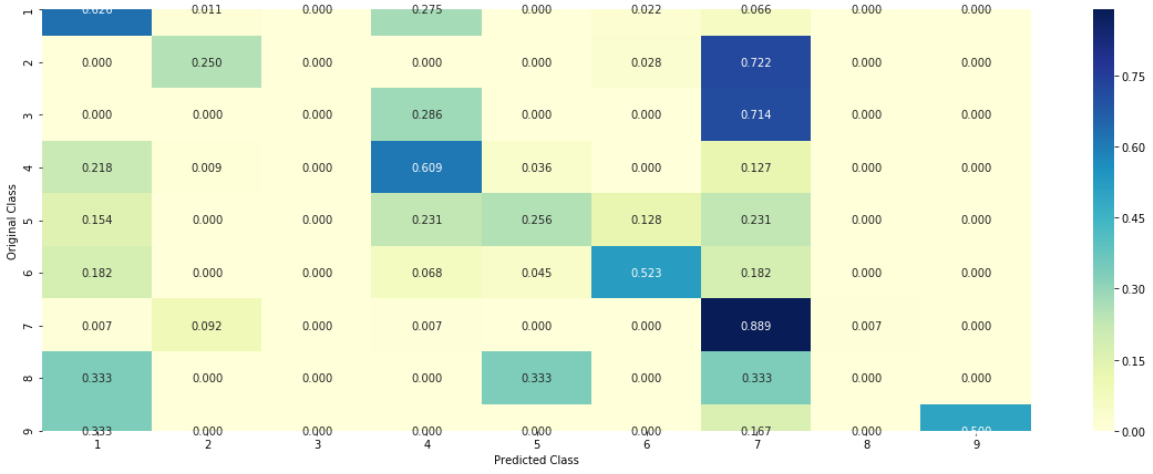
In [186]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
s='log', random_state=42)
predict_and_plot_confusion_matrix(train_3, train_y, cv_3, cv_y, clf)
```

```
Log loss : 1.1562598627730973
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```


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


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



FEATURE ENGINEERING TO REDUCE LOG LOSS TO LESS THAN 1

In [ ]:

```
train_gene_var_tf = hstack((train_gene_feature_tf,train_var_feature_tf))
test_gene_var_tf = hstack((test_gene_feature_tf,test_var_feature_tf))
cv_gene_var_tf = hstack((cv_gene_feature_tf,cv_var_feature_tf))

train_x_tf = hstack((train_gene_var_tf, train_text_feature_tf)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_tf = hstack((test_gene_var_tf, test_text_feature_tf)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_tf = hstack((cv_gene_var_tf, cv_text_feature_tf)).tocsr()
cv_y = np.array(list(cv_df['Class']))


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

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

In [188]:

```
print(train_x_responseCoding.shape)
```

```
(2124, 27)
```

In [199]:

```
train_final_1=hstack((train_x_tf_1,train_x_responseCoding)).tocsr()
cv_final_1=hstack((cv_x_tf_1,cv_x_responseCoding))
test_final_1=hstack((test_x_tf_1,test_x_responseCoding))
print(train_final_1.shape)
print(test_final_1.shape)
print(cv_final_1.shape)
```

```
(2124, 56106)
(665, 56106)
(532, 56106)
```

In [189]:

```
print(train_x_tf.shape)
```

```
(2124, 56079)
```

In [194]:

```
train_final=hstack((train_x_tf,train_x_responseCoding)).tocsr()
cv_final=hstack((cv_x_tf,cv_x_responseCoding))
test_final=hstack((test_x_tf,test_x_responseCoding))
print(train_final.shape)
print(test_final.shape)
print(cv_final.shape)
```

```
(2124, 56106)
(665, 56106)
(532, 56106)
```

In [216]:

```
tfidf=TfidfVectorizer(max_features=4000)
tfidf.fit(train_df['TEXT'])
tfidf_train=tfidf.transform(train_df['TEXT'])
tfidf_cv=tfidf.transform(cv_df['TEXT'])
tfidf_test=tfidf.transform(test_df['TEXT'])
print(tfidf_train.shape)
print(tfidf_cv.shape)
print(tfidf_test.shape)
```

```
(2124, 4000)
(532, 4000)
(665, 4000)
```

In [217]:

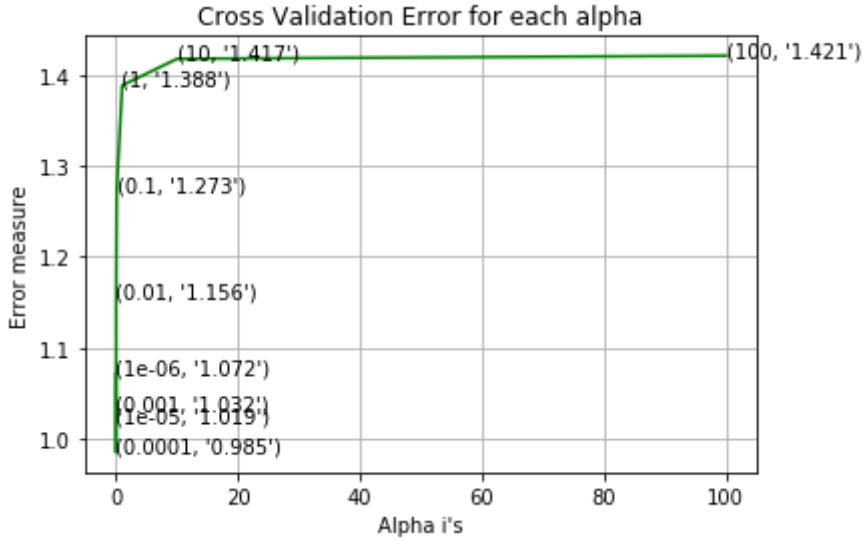
```
train_6=hstack((tfidf_train,train_gene_var_tf,train_x_responseCoding)).tocsr()
cv_6=hstack((tfidf_cv,cv_gene_var_tf,cv_x_responseCoding)).tocsr()
test_6=hstack((tfidf_test,test_gene_var_tf,test_x_responseCoding)).tocsr()
print(train_6.shape)
print(cv_6.shape)
print(test_6.shape)
```

```
(2124, 6197)
(532, 6197)
(665, 6197)
```

In [218]:

```
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_6, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_6, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_6)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
-15))
    # to avoid rounding error while multiplying probabilites we use log-probability est
imates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
)
clf.fit(train_6, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_6, train_y)

predict_y = sig_clf.predict_proba(train_6)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
ss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_6)
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_6)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
s(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.071927914821931
for alpha = 1e-05
Log Loss : 1.018994011285108
for alpha = 0.0001
Log Loss : 0.9850907098474808
for alpha = 0.001
Log Loss : 1.0323058378716714
for alpha = 0.01
Log Loss : 1.1558948338424906
for alpha = 0.1
Log Loss : 1.2730217509562978
for alpha = 1
Log Loss : 1.3882554188431322
for alpha = 10
Log Loss : 1.4173850532540937
for alpha = 100
Log Loss : 1.4207764977712571
```

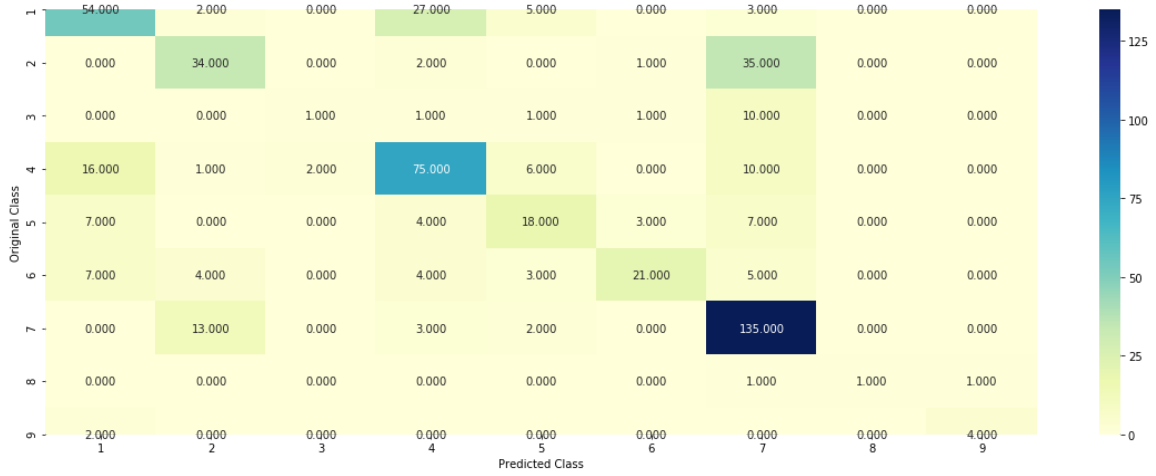



```
For values of best alpha =  0.0001 The train log loss is: 0.36934756282667
91
For values of best alpha =  0.0001 The cross validation log loss is: 0.985
0907098474808
For values of best alpha =  0.0001 The test log loss is: 0.995716930866744
```

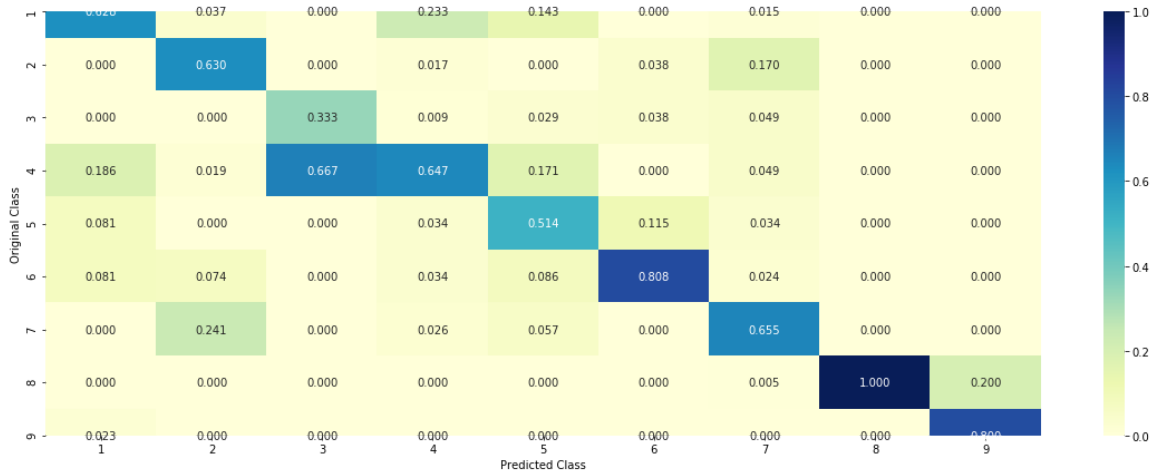
In [219]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_6, train_y, cv_6, cv_y, clf)
```

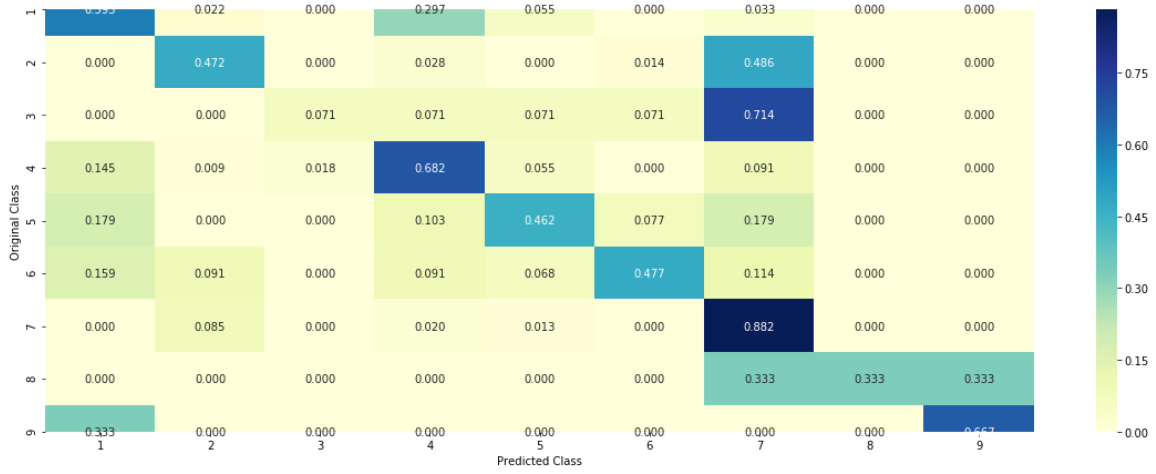
```
Log loss : 0.9850907098474808
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```




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




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



CONCLUSION:

WE USED 4000 FEATURES OF TFIDF ENCODED TEXT AND TFIDF ENCODED GENE AND VARIANCE FEATURE AND RESPONCE CODED FEATURES AND WE GOT TRAIN LOSS 0.3693 AND CV LOSS 0.9850 AND TEST LOSS 0.9957

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: