Personalized cancer diagnosis

Business Problem

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.

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)

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.

Machine Learning Problem Formulation

Data

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)

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). ...

Mapping the real-world problem to an ML problem

Type of Machine Learning Problem

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

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

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.

Train, CV and Test Datasets

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

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
```

```
Using TensorFlow backend.
```

Reading Data

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_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

Reading Text Data

In [3]:

```
# note the seprator in this file is ||
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT
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...

Preprocessing of text

In [4]:

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

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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 355.278724 seconds
```

Mering both dataframes data and data_text on basis of ID

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]:

```
#checking for null values in the data
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]:

```
#replacing null values in text with gene and variation value of the row
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

Test, Train and Cross Validation Split

Splitting data into train, test and cross validation (64:20:16) ie (TRAIN,TEST)(80,20) and then(TRAIN,CV)(80,20)%

In [10]:

```
#creating y label
y_true = result['Class'].values

#replcaing space or  extra spaces in the gene , variation column with_
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 var
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_tru
# split the train data into train and cross validation by maintaining same distribu
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_trai
```

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]:

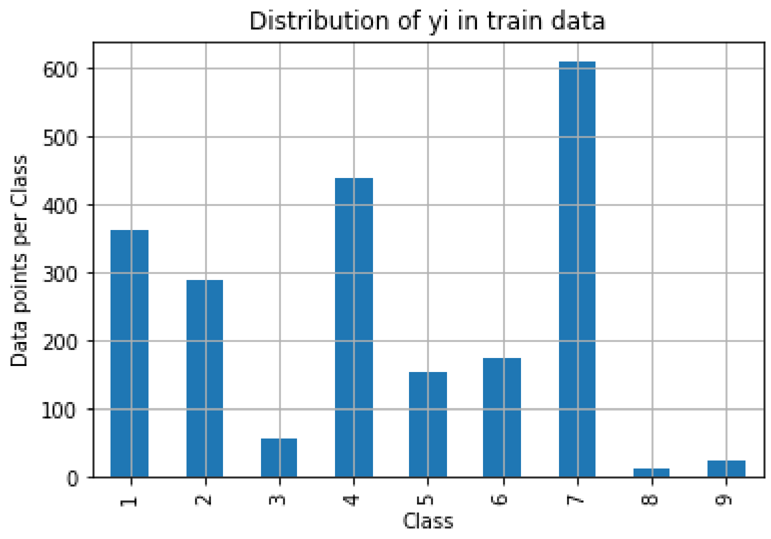
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

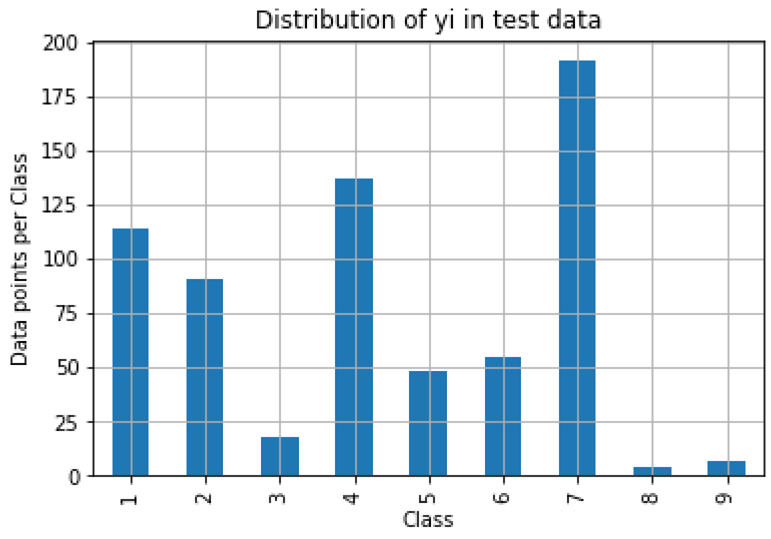
Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:

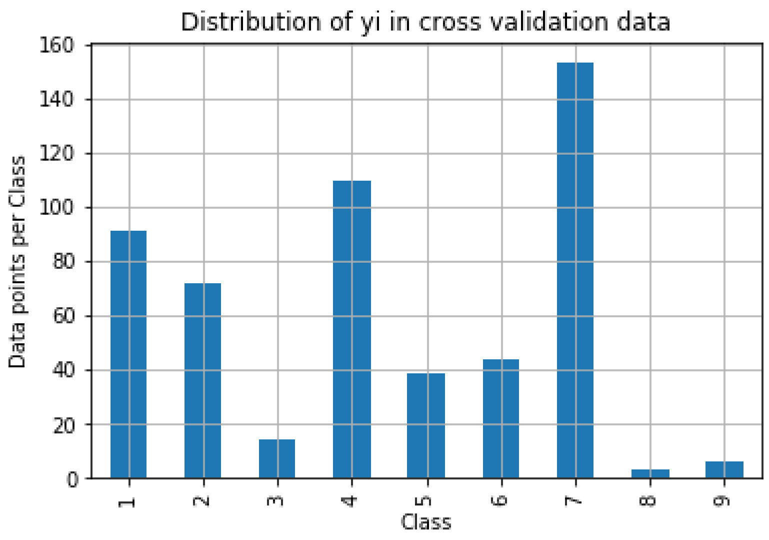
```
# it returns a dict, keys as class labels and values as the number of data points i
#https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.sc
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
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.h
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
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.value


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.h
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
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

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()

#argsort returns indices after sorting the array
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.h
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
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
```



```
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 %)
```

From the above plots it is clear that our dataset is not balanced.
class of 7,4,1,2 are more dominant over other class labels
the labels are equally distributed in the train cv and cross validation sets

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 pr

    # since precision is , of all the points that the model declared to be positive
    #precision=(element in cell)/(sum of elements in the column) each column is pre
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that co

    # 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
    # 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

    # since recall is , of all the points that are actually positive how many of th
    #recall=(element in cell)/(sum of elements in the row) each column is predicted


    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that ro
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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, ytickl
    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, ytickl
    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, ytickl
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

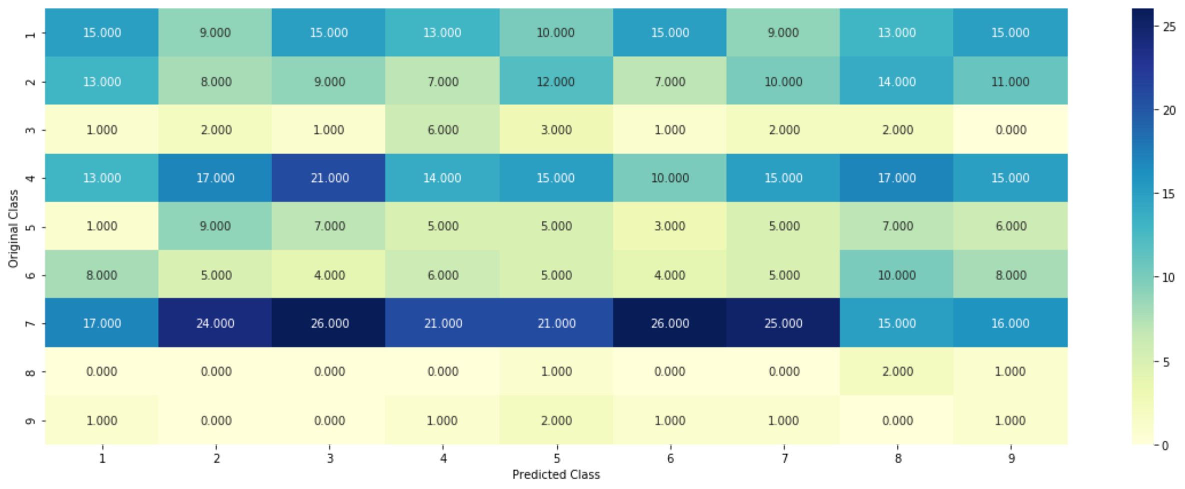
Prediction using a 'Random' Model

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

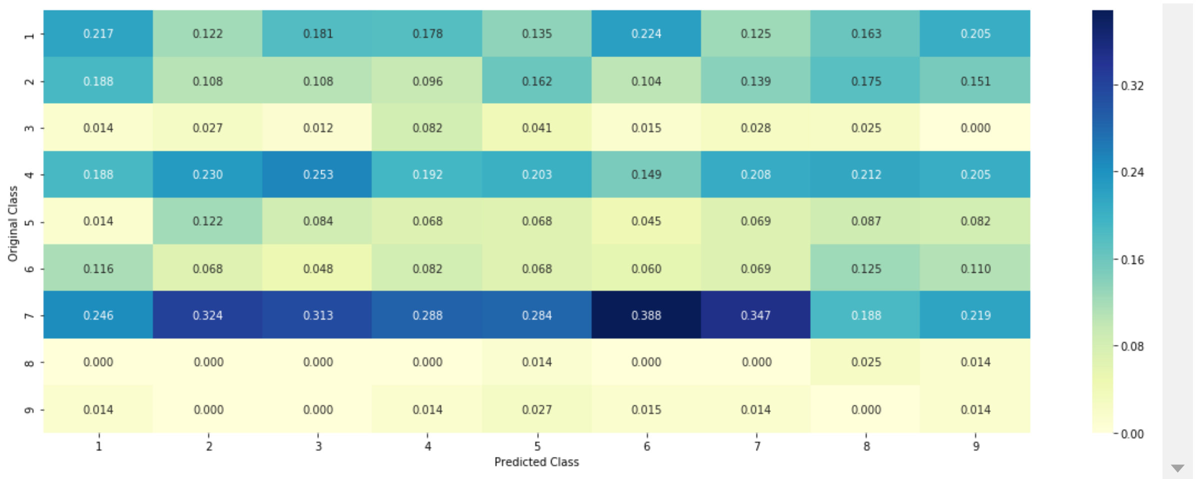
In [14]:

```
# 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_predi


# 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,

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

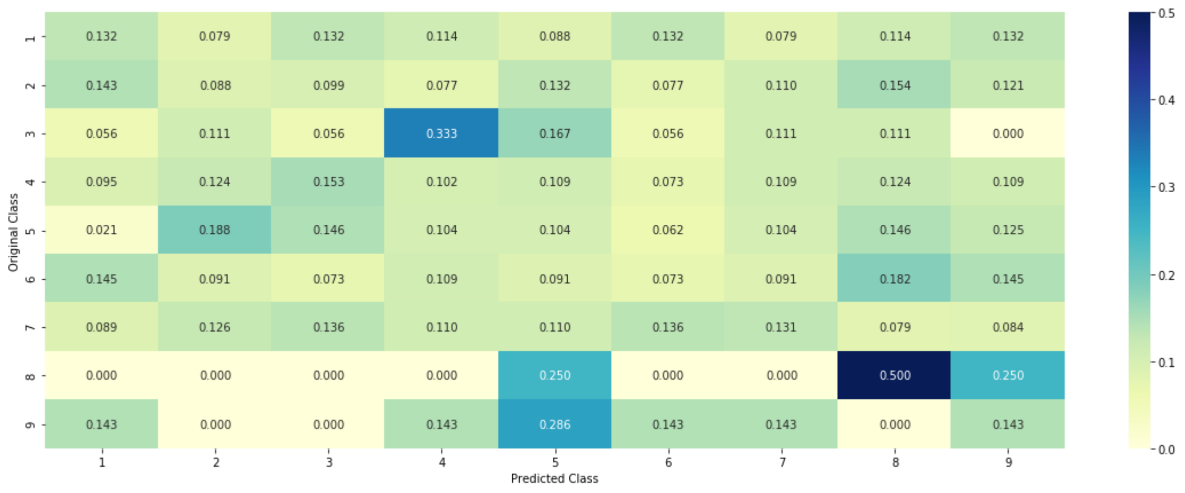
```
Log loss on Cross Validation Data using Random Model 2.526252273477774
6
Log loss on Test Data using Random Model 2.4575948042212996
-------------------- Confusion matrix --------------------
```



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



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



Univariate Analysis

In [15]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing

def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    value_count = train_df[feature].value_counts()

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

    # denominator will contain the number of time that particular feature occured i
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pe
        # vec is 9 dimensional 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 part
            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
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181811
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918238993710691
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334
    #      ...
    #     }
    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
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there
    # 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)

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 : 233
BRCA1     178
TP53      101
EGFR       94
BRCA2      83
PTEN       77
KIT        70
BRAF       64
ERBB2      42
ALK        41
PDGFRA     36
Name: Gene, dtype: int64
```

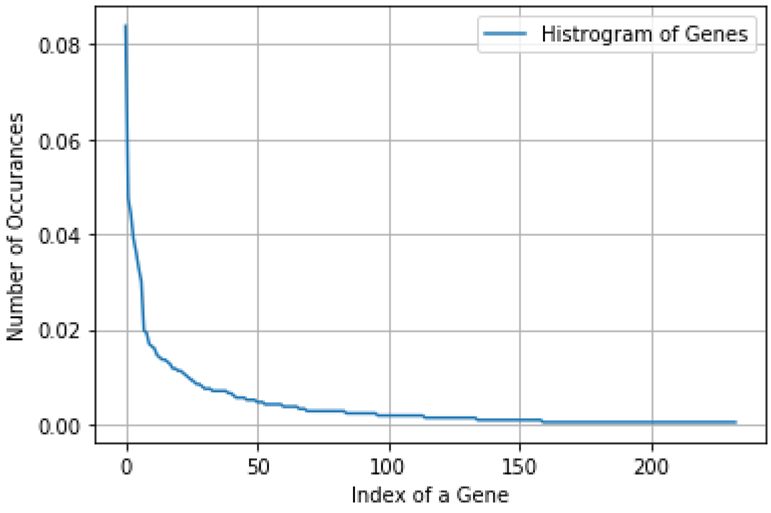
In [17]:

```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in th
```

```
Ans: There are 233 different categories of genes in the train data, an
d they are distibuted as follows
```

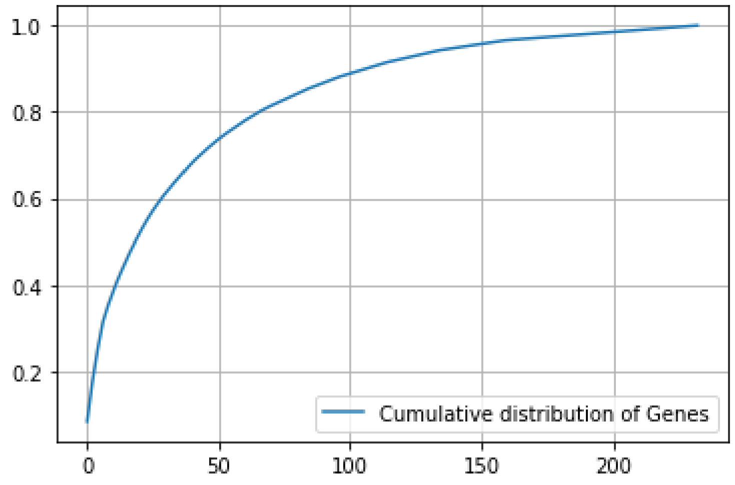
In [18]:

```
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()
```



95% of our data constitue 150 genes which means 150 genes occur more number of times
76 genes occur less number of times

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. Tfidf 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
```

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

tfidf Vectorizing of Gene feature.

In [22]:

```
# tfidf Vectorizering  of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:

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

Out[23]:

```
2685     BRAF
2717     BRAF
506      TP53
2537    BRCA1
359     EP300
Name: Gene, dtype: object
```

In [24]:

```
gene_vectorizer.get_feature_names()
```

Out[24]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'asxl1',
 'atm',
 'atr',
 'aurka',
 'aurkb',
 'axl',
```

In [25]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding
```

```
train_gene_feature_onehotCoding is converted feature using one-hot enc
oding method. The shape of gene feature: (2124, 232)
```

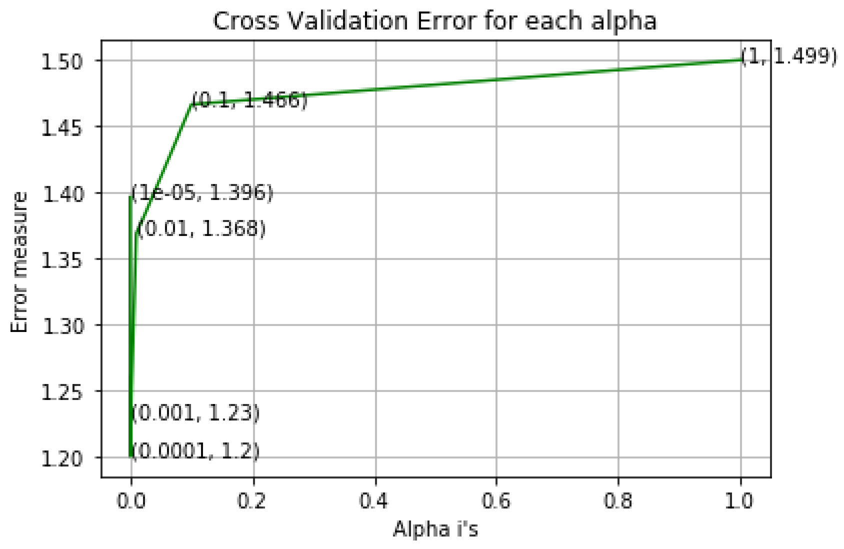
Q4. How good is this gene feature in predicting y_i?

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)] # hyperparameter  for SGD classifier.

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y,

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
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:",lo
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
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
```

```
For values of alpha =  1e-05 The log loss is: 1.3960441958340442
For values of alpha =  0.0001 The log loss is: 1.2001246218982458
For values of alpha =  0.001 The log loss is: 1.2295377021876237
For values of alpha =  0.01 The log loss is: 1.3683615765690793
For values of alpha =  0.1 The log loss is: 1.465805392239332
For values of alpha =  1 The log loss is: 1.499471467602267
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0310717534
540468
For values of best alpha =  0.0001 The cross validation log loss is:
1.2001246218982458
For values of best alpha =  0.0001 The test log loss is: 1.21995434454
709
```

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 ", uniqu

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_co
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_
```

```
Q6. How many data points in Test and CV datasets are covered by the  2
33  genes in train dataset?
Ans
1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 512 out of  532 : 96.2406015037594
```

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 : 1936
Truncating_Mutations    56
Deletion                47
Amplification           46
Fusions                 22
Overexpression           4
Q61H                     3
E17K                     3
V321M                    2
T73I                     2
G12V                     2
Name: Variation, dtype: int64
```

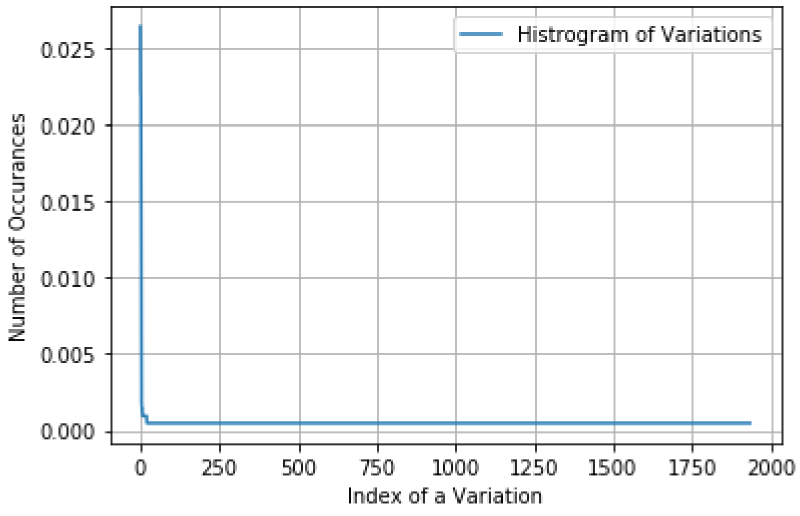
In [29]:

```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variat
```

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

In [30]:

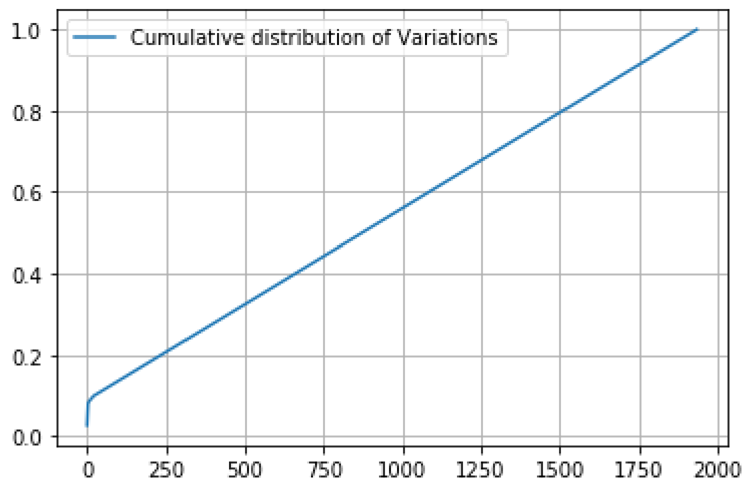
```
#pdf of variants
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:

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

```
[0.02636535 0.04849341 0.07015066 ... 0.99905838 0.99952919 1.
]
```



Most variants occur once,twice or thrice
There are only 4 variations that occur more than thrice

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]:

```
#Response Encoding Variants
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation"
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", c
```

In [33]:

```
print("train_variation_feature_responseCoding is a converted feature using the resp
```

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

In [34]:

```
# TfidfVectorizing of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df[
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Varia
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation
```

In [35]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-ho
```

```
train_variation_feature_onehotEncoded is converted feature using the o
nne-hot encoding method. The shape of Variation feature: (2124, 1966)
```

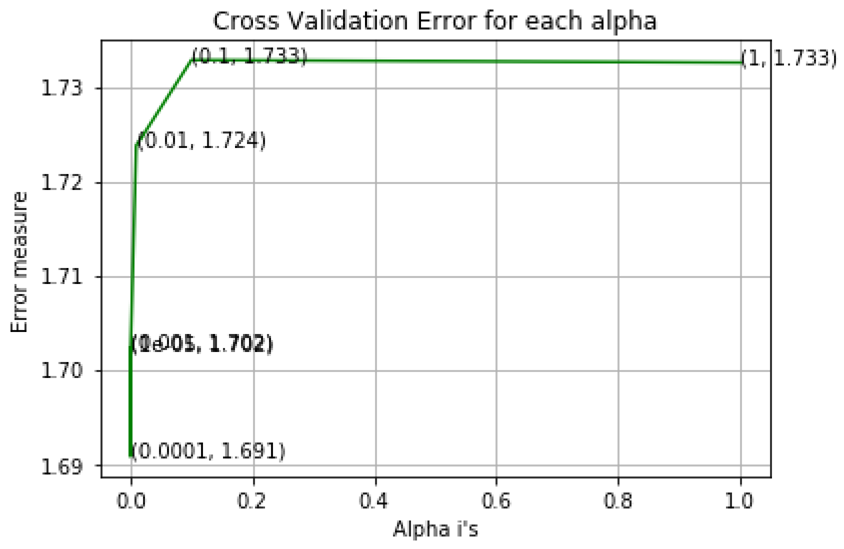
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [36]:

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


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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y,

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
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:",lo
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
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
```

```
For values of alpha =  1e-05 The log loss is: 1.702215214603765
For values of alpha =  0.0001 The log loss is: 1.6907961314504558
For values of alpha =  0.001 The log loss is: 1.7022832956872422
For values of alpha =  0.01 The log loss is: 1.7237807288463658
For values of alpha =  0.1 The log loss is: 1.7328216683137978
For values of alpha =  1 The log loss is: 1.732546066025203
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7120086572
768334
For values of best alpha =  0.0001 The cross validation log loss is:
1.6907961314504558
For values of best alpha =  0.0001 The test log loss is: 1.71451277726
56776
```

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]
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_co
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_
```

```
Q12. How many data points are covered by total  1936  genes in test an
d cross validation data sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 68 out of  532 : 12.781954887218044
```

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 frame 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.ge
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['
            row_index += 1
    return text_feature_responseCoding
```

In [40]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in tra
text_vectorizer = TfidfVectorizer(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*
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
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 : 53684
```

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
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_fe
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_
```

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] , reve
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [46]:

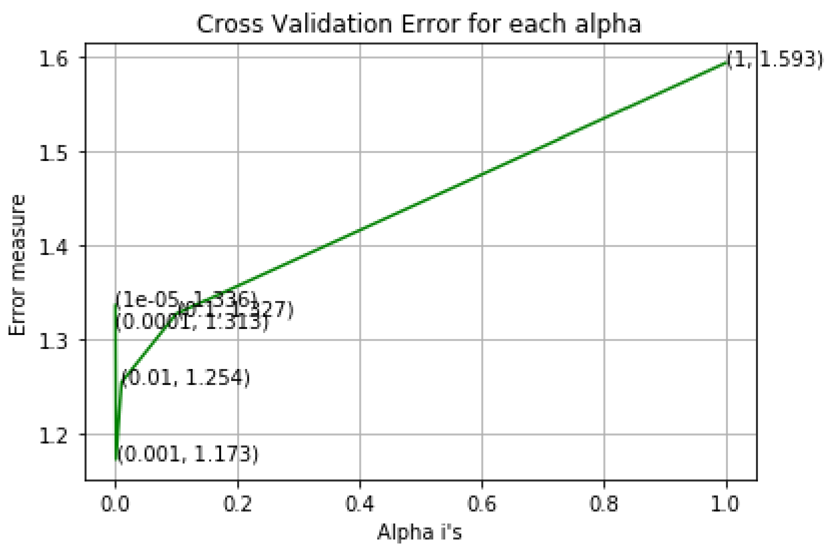
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({0.008574303908560071: 328, 0.015987169306901 46: 317, 0.0420
8240152401138: 239, 0.07658001200561677: 199, 0.014128920562395437:
181, 0.016833598690227602: 158, 0.35303239710566114: 156, 0.25568578
426946303: 128, 0.05623306492154619: 127, 0.014731055150231894: 124,
0.019504620880496842: 117, 0.010597162123374082: 117, 0.037716830559
92814: 114, 0.06673732238373203: 107, 0.06023209965933393: 105, 0.08
487356550593611: 103, 0.06867196994995865: 100, 0.04474694617929618:
96, 0.09761667084406116: 93, 0.0189359797549171: 93, 0.0346314419489
2992: 92, 0.02639673984766501: 87, 0.017598639214930073: 84, 0.01223
509457076854: 84, 0.02600881331314389: 82, 0.04825135199590301: 80,
0.023899786423977987: 80, 0.017404315415231075: 80, 0.01615325424480
463: 80, 0.04541163593463447: 74, 0.019688614901395833: 71, 0.032289
66851952324: 68, 0.05996213648105399 4: 65, 0.04468428359231119: 64,
0.030878180816968186: 64, 0.04353100946001829: 63, 0.013219304598797
523: 63, 0.030103514088748982: 61, 0.02023423182093816: 61, 0.128721
5397124493: 60, 0.01927176588222776: 59, 0.016507829907368814: 59,
0.0905740189557273: 58, 0.014815596432240992: 58, 0.0124488896760965
45: 57, 0.02699034863112965: 56, 0.06631867607959732: 55, 0.01564890
300414269: 55, 0.04806960473261467: 54, 0.013903140924614631: 54, 0.
```

In [47]:

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y,

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
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:",lo
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
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
```

```
For values of alpha =  1e-05 The log loss is: 1.3361315433214243
For values of alpha =  0.0001 The log loss is: 1.3129295283294793
For values of alpha =  0.001 The log loss is: 1.173031773509908
For values of alpha =  0.01 The log loss is: 1.254188768345973
For values of alpha =  0.1 The log loss is: 1.3271259372436093
For values of alpha =  1 The log loss is: 1.592925497564635
```



```
For values of best alpha =  0.001 The train log loss is: 0.68139796732
0624
For values of best alpha =  0.001 The cross validation log loss is: 1.
173031773509908
For values of best alpha =  0.001 The test log loss is: 1.126944747467
9597
```

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 trai
```

```
97.404 % of word of test data appeared in train data
97.332 % of word of Cross Validation appeared in train data
```

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
    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))/t
    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
        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 [{}]".f
        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

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

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_variati
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_featur

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_var
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_featu
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_respon
```

In [54]:

```
print("TfidfVectorizer features :")
print("(number of data points * number of features) in train data = ", train_x_oneh
print("(number of data points * number of features) in test data = ", test_x_onehot
print("(number of data points * number of features) in cross validation data =", cv
```

```
TfidfVectorizer features :
(number of data points * number of features) in train data =  (2124, 5
5882)
(number of data points * number of features) in test data =  (665, 558
82)
(number of data points * number of features) in cross validation data
= (532, 55882)
```

In [55]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_resp
print("(number of data points * number of features) in test data = ", test_x_respon
print("(number of data points * number of features) in cross validation data =", cv
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(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)
```

Base Line Model

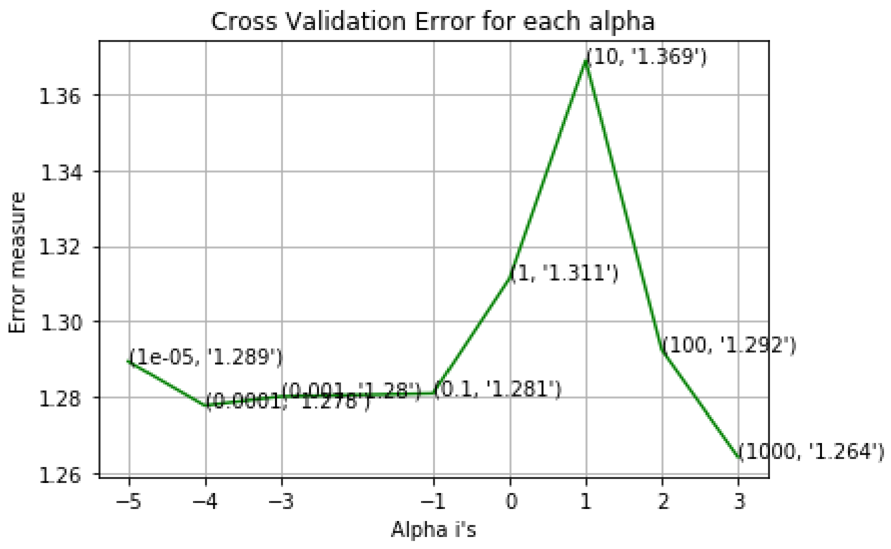
Naive Bayes

Hyper parameter tuning

In [56]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stat

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_, ep
    # to avoid rounding error while multiplying probabilites we use log-probability
    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:",lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
for alpha = 1e-05
Log Loss : 1.2893912644159826
for alpha = 0.0001
Log Loss : 1.2777240598082114
for alpha = 0.001
Log Loss : 1.2801123591541432
for alpha = 0.1
Log Loss : 1.2809329518534558
for alpha = 1
Log Loss : 1.3113861529711135
for alpha = 10
Log Loss : 1.3689257770639751
for alpha = 100
Log Loss : 1.2924746976341848
for alpha = 1000
Log Loss : 1.264262596140312
```



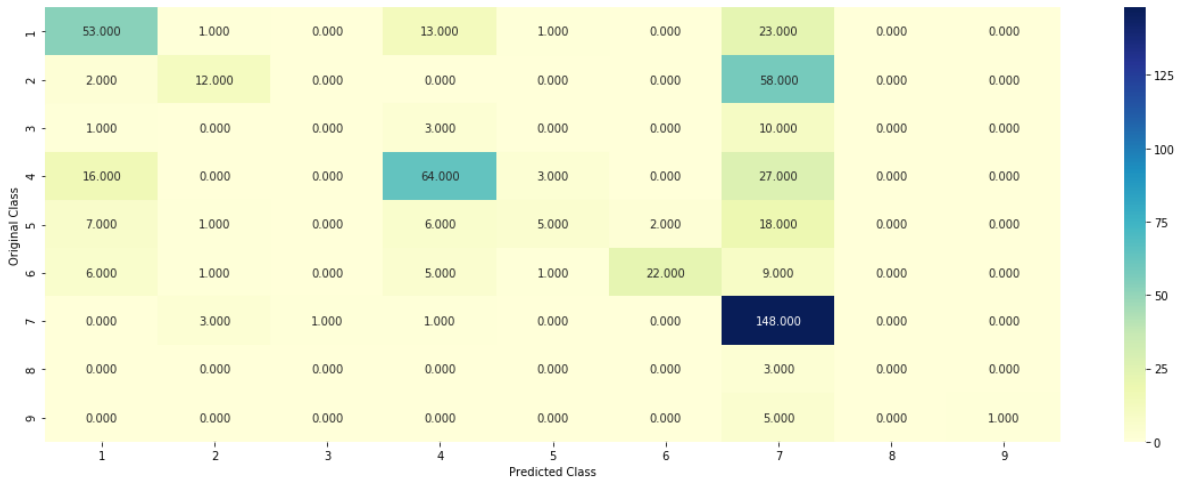
```
For values of best alpha =  1000 The train log loss is: 0.923200531311
3839
For values of best alpha =  1000 The cross validation log loss is: 1.2
64262596140312
For values of best alpha =  1000 The test log loss is: 1.2329253995408
644
```

Testing the model with best hyper paramters

In [57]:

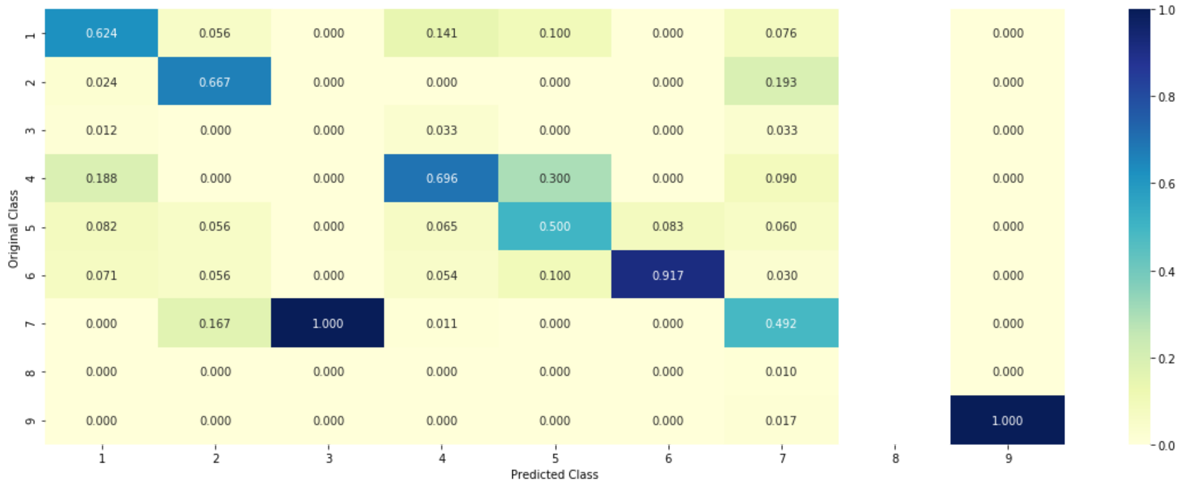
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites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

```
Log Loss : 1.264262596140312
Number of missclassified point : 0.4266917293233083
-------------------- Confusion matrix --------------------
```



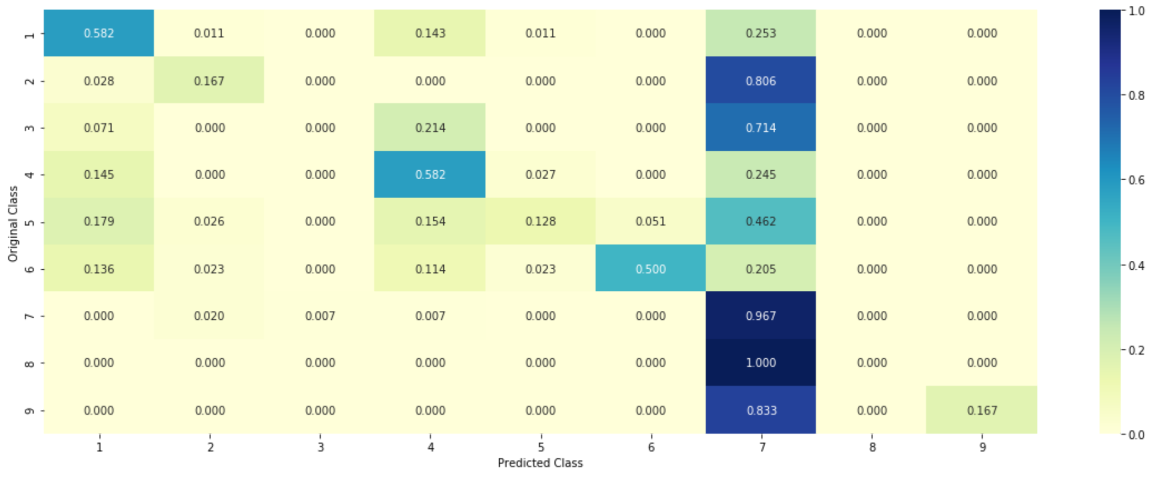
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	53.000	1.000	0.000	13.000	1.000	0.000	23.000	0.000	0.000
2	2.000	12.000	0.000	0.000	0.000	0.000	58.000	0.000	0.000
3	1.000	0.000	0.000	3.000	0.000	0.000	10.000	0.000	0.000
4	16.000	0.000	0.000	64.000	3.000	0.000	27.000	0.000	0.000
5	7.000	1.000	0.000	6.000	5.000	2.000	18.000	0.000	0.000
6	6.000	1.000	0.000	5.000	1.000	22.000	9.000	0.000	0.000
7	0.000	3.000	1.000	1.000	0.000	0.000	148.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	3.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	5.000	0.000	1.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.624	0.056	0.000	0.141	0.100	0.000	0.076		0.000
2	0.024	0.667	0.000	0.000	0.000	0.000	0.193		0.000
3	0.012	0.000	0.000	0.033	0.000	0.000	0.033		0.000
4	0.188	0.000	0.000	0.696	0.300	0.000	0.090		0.000
5	0.082	0.056	0.000	0.065	0.500	0.083	0.060		0.000
6	0.071	0.056	0.000	0.054	0.100	0.917	0.030		0.000
7	0.000	0.167	1.000	0.011	0.000	0.000	0.492		0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.010		0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.017		1.000

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



Feature Importance, Correctly classified point

In [58]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[1.360e-02 3.800e-03 7.000e-04 9.269e-
01 1.100e-02 9.200e-03 3.360e-02
  1.000e-03 2.000e-04]]
Actual Class : 4
--------------------------------------------------
11 Text feature [proteins] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [activity] present in test data point [True]
16 Text feature [experiments] present in test data point [True]
17 Text feature [loss] present in test data point [True]
18 Text feature [function] present in test data point [True]
19 Text feature [acid] present in test data point [True]
20 Text feature [mammalian] present in test data point [True]
21 Text feature [whereas] present in test data point [True]
22 Text feature [indicated] present in test data point [True]
23 Text feature [results] present in test data point [True]
24 Text feature [shown] present in test data point [True]
26 Text feature [type] present in test data point [True]
27 Text feature [whether] present in test data point [True]
28 Text feature [amino] present in test data point [True]
29 Text feature [pten] present in test data point [True]
30 Text feature [important] present in test data point [True]
31 Text feature [determined] present in test data point [True]
32 Text feature [two] present in test data point [True]
33 Text feature [described] present in test data point [True]
34 Text feature [bind] present in test data point [True]
35 Text feature [tagged] present in test data point [True]
36 Text feature [also] present in test data point [True]
37 Text feature [wild] present in test data point [True]
38 Text feature [indicate] present in test data point [True]
39 Text feature [missense] present in test data point [True]
40 Text feature [determine] present in test data point [True]
41 Text feature [retained] present in test data point [True]
42 Text feature [vitro] present in test data point [True]
44 Text feature [ability] present in test data point [True]
45 Text feature [functions] present in test data point [True]
46 Text feature [expressed] present in test data point [True]
47 Text feature [reduced] present in test data point [True]
48 Text feature [may] present in test data point [True]
49 Text feature [purified] present in test data point [True]
50 Text feature [mutations] present in test data point [True]
51 Text feature [abrogate] present in test data point [True]
52 Text feature [containing] present in test data point [True]
53 Text feature [levels] present in test data point [True]
54 Text feature [thus] present in test data point [True]
56 Text feature [analyzed] present in test data point [True]
57 Text feature [although] present in test data point [True]
```

```
59 Text feature [hamartoma] present in test data point [True]
60 Text feature [dephosphorylate] present in test data point [True]
61 Text feature [effects] present in test data point [True]
62 Text feature [suggest] present in test data point [True]
64 Text feature [catalytic] present in test data point [True]
66 Text feature [vivo] present in test data point [True]
67 Text feature [using] present in test data point [True]
68 Text feature [lower] present in test data point [True]
69 Text feature [dic8] present in test data point [True]
70 Text feature [30] present in test data point [True]
71 Text feature [functional] present in test data point [True]
72 Text feature [s170r] present in test data point [True]
73 Text feature [similar] present in test data point [True]
74 Text feature [amount] present in test data point [True]
75 Text feature [standard] present in test data point [True]
77 Text feature [expression] present in test data point [True]
82 Text feature [g129e] present in test data point [True]
83 Text feature [performed] present in test data point [True]
84 Text feature [binding] present in test data point [True]
86 Text feature [critical] present in test data point [True]
88 Text feature [suppressor] present in test data point [True]
89 Text feature [yielded] present in test data point [True]
90 Text feature [tensin] present in test data point [True]
91 Text feature [three] present in test data point [True]
92 Text feature [possible] present in test data point [True]
94 Text feature [previously] present in test data point [True]
95 Text feature [addition] present in test data point [True]
96 Text feature [discussion] present in test data point [True]
97 Text feature [result] present in test data point [True]
98 Text feature [vector] present in test data point [True]
99 Text feature [however] present in test data point [True]
Out of the top  100  features  73 are present in query point
```

Feature Importance, Incorrectly classified point

In [59]:

```
test_point_index =15
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.13e-02 2.52e-02 4.00e-04 4.22e-02
8.80e-03 8.20e-03 8.83e-01 9.00e-04
  1.00e-04]]
Actual Class : 7
--------------------------------------------------
13 Text feature [cells] present in test data point [True]
16 Text feature [activated] present in test data point [True]
17 Text feature [cell] present in test data point [True]
19 Text feature [kinase] present in test data point [True]
20 Text feature [activation] present in test data point [True]
21 Text feature [presence] present in test data point [True]
22 Text feature [phosphorylation] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [shown] present in test data point [True]
27 Text feature [factor] present in test data point [True]
28 Text feature [expressing] present in test data point [True]
29 Text feature [also] present in test data point [True]
30 Text feature [recently] present in test data point [True]
31 Text feature [suggest] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [growth] present in test data point [True]
35 Text feature [treated] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [found] present in test data point [True]
38 Text feature [previously] present in test data point [True]
39 Text feature [addition] present in test data point [True]
40 Text feature [compared] present in test data point [True]
42 Text feature [serum] present in test data point [True]
43 Text feature [well] present in test data point [True]
44 Text feature [increased] present in test data point [True]
45 Text feature [tyrosine] present in test data point [True]
46 Text feature [constitutive] present in test data point [True]
47 Text feature [enhanced] present in test data point [True]
48 Text feature [higher] present in test data point [True]
49 Text feature [1a] present in test data point [True]
50 Text feature [independent] present in test data point [True]
51 Text feature [figure] present in test data point [True]
52 Text feature [potential] present in test data point [True]
53 Text feature [mutations] present in test data point [True]
55 Text feature [followed] present in test data point [True]
62 Text feature [described] present in test data point [True]
63 Text feature [various] present in test data point [True]
64 Text feature [demonstrated] present in test data point [True]
65 Text feature [using] present in test data point [True]
66 Text feature [mechanism] present in test data point [True]
68 Text feature [3b] present in test data point [True]
```

```
69 Text feature [consistent] present in test data point [True]
70 Text feature [showed] present in test data point [True]
71 Text feature [mutant] present in test data point [True]
72 Text feature [constitutively] present in test data point [True]
73 Text feature [inhibitors] present in test data point [True]
74 Text feature [activating] present in test data point [True]
75 Text feature [3a] present in test data point [True]
76 Text feature [interestingly] present in test data point [True]
77 Text feature [obtained] present in test data point [True]
79 Text feature [may] present in test data point [True]
80 Text feature [antibodies] present in test data point [True]
81 Text feature [proliferation] present in test data point [True]
83 Text feature [including] present in test data point [True]
84 Text feature [reported] present in test data point [True]
85 Text feature [2b] present in test data point [True]
86 Text feature [culture] present in test data point [True]
88 Text feature [furthermore] present in test data point [True]
89 Text feature [mutation] present in test data point [True]
90 Text feature [fig] present in test data point [True]
91 Text feature [expression] present in test data point [True]
92 Text feature [induced] present in test data point [True]
93 Text feature [4a] present in test data point [True]
94 Text feature [without] present in test data point [True]
95 Text feature [role] present in test data point [True]
96 Text feature [observed] present in test data point [True]
97 Text feature [absence] present in test data point [True]
98 Text feature [identified] present in test data point [True]
Out of the top  100  features  69 are present in query point
```

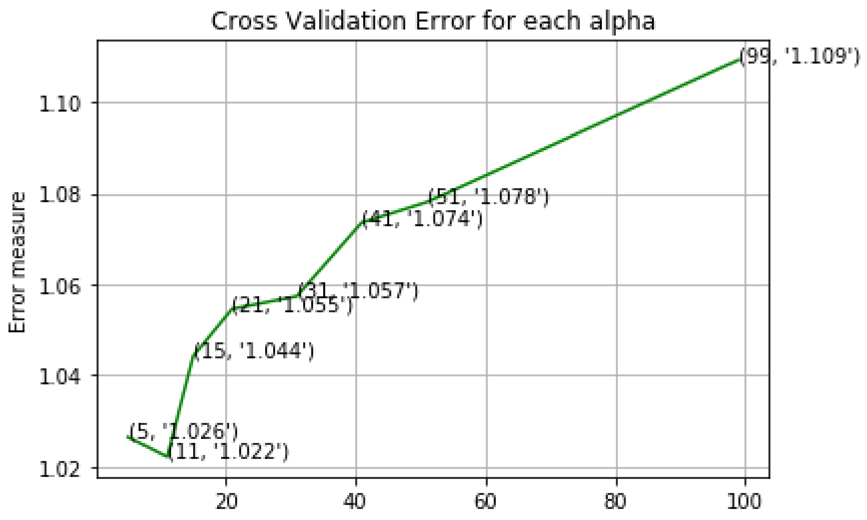
K Nearest Neighbour Classification

Hyper parameter tuning

In [60]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module

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_, ep
    # to avoid rounding error while multiplying probabilites we use log-probability
    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:",lo
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
for alpha = 5
Log Loss : 1.0264514518352434
for alpha = 11
Log Loss : 1.022110228761473
for alpha = 15
Log Loss : 1.0441748966932796
for alpha = 21
Log Loss : 1.0545738195317431
for alpha = 31
Log Loss : 1.0573676043798328
for alpha = 41
Log Loss : 1.0735776929384386
for alpha = 51
Log Loss : 1.0780657910114007
for alpha = 99
Log Loss : 1.1092709419391042
```



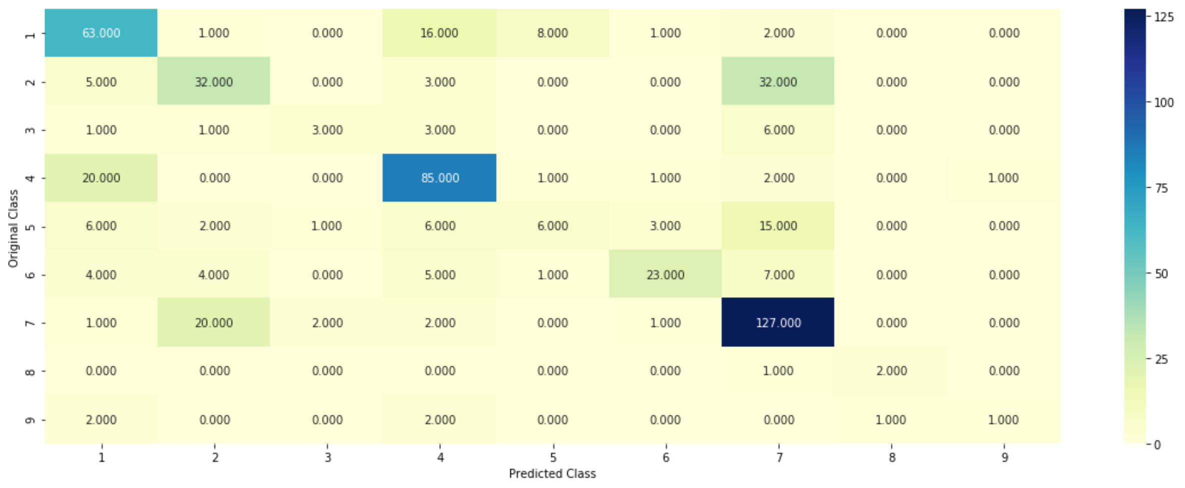
```
For values of best alpha =  11 The train log loss is: 0.64757635393552
1
For values of best alpha =  11 The cross validation log loss is: 1.022
110228761473
For values of best alpha =  11 The test log loss is: 1.104858986152989
```

Testing the model with best hyper paramters

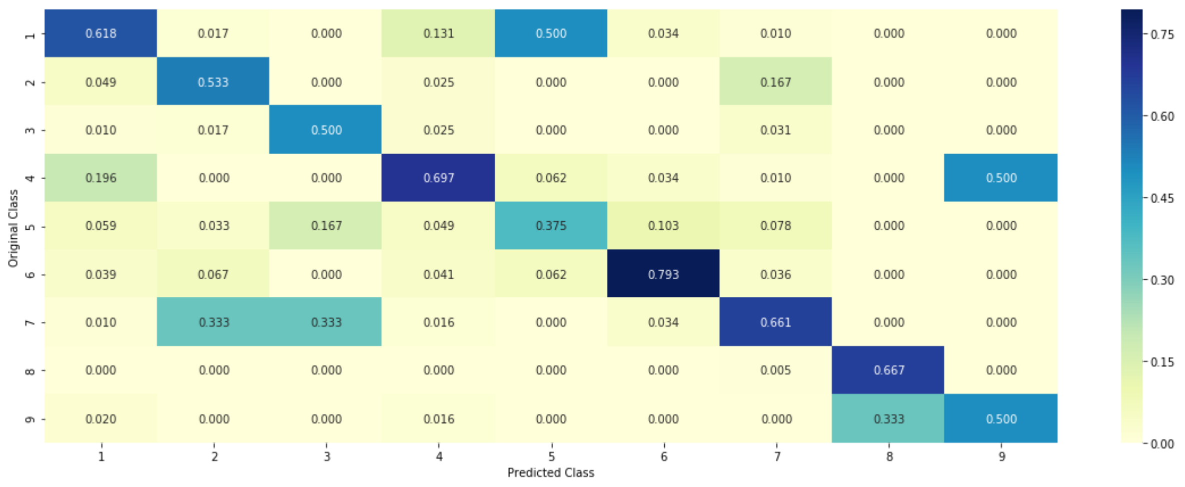
In [61]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module

clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCod
```

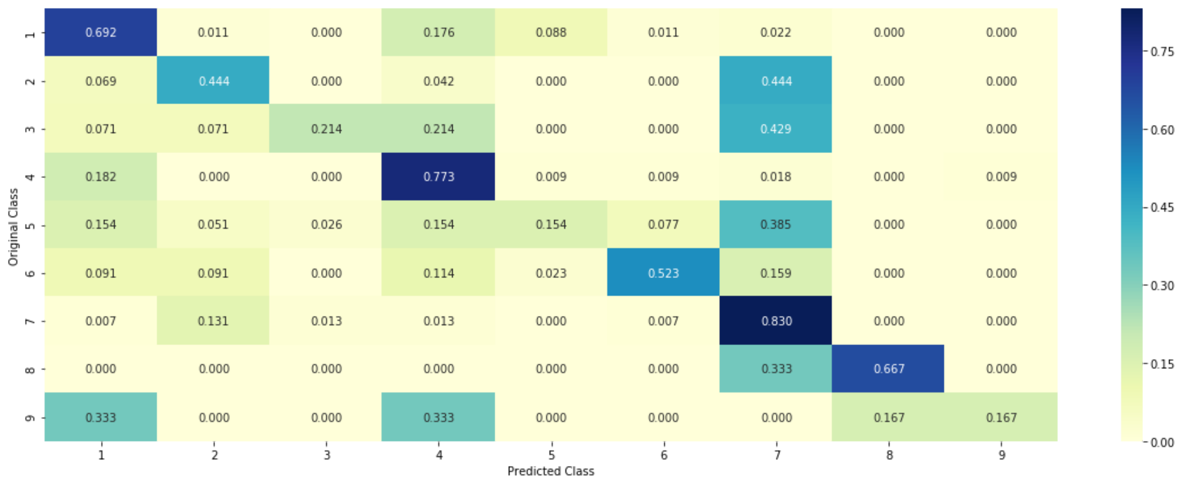
```
Log loss : 1.022110228761473
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```



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



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



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 = 15
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),
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to c
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7
2 7 7 7 7 6 6 2 2]
Fequency of nearest points : Counter({7: 6, 2: 3, 6: 2})
```

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 = 25

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(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),
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test point
s belongs to classes [2 7 7 7 7 7 7 7 2 7 7]
Fequency of nearest points : Counter({7: 9, 2: 2})
```

Logistic Regression

With Class balancing

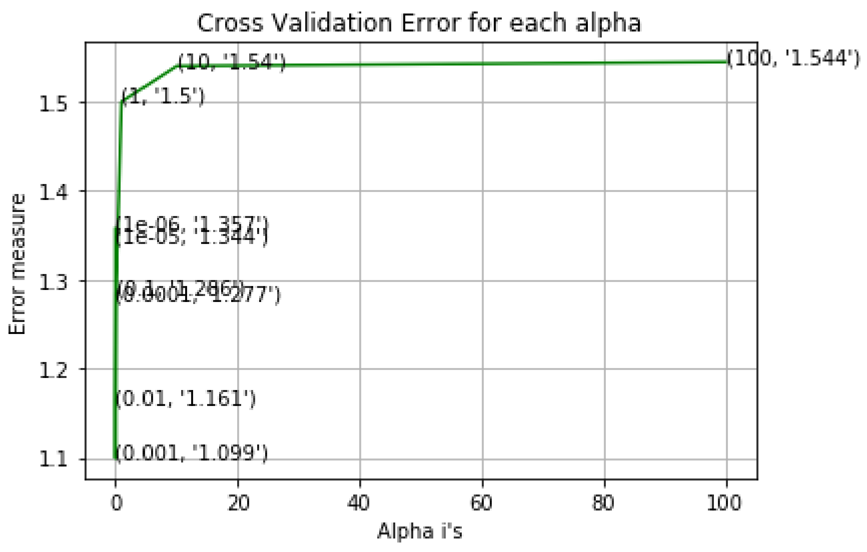
Hyper paramter tuning

In [64]:

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


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',
    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
    # to avoid rounding error while multiplying probabilites we use log-probability
    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',
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:",lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
for alpha = 1e-06
Log Loss : 1.3574918422981772
for alpha = 1e-05
Log Loss : 1.3435812952028592
for alpha = 0.0001
Log Loss : 1.276708405349777
for alpha = 0.001
Log Loss : 1.09924835246161
for alpha = 0.01
Log Loss : 1.16144473406025
for alpha = 0.1
Log Loss : 1.2855183667943961
for alpha = 1
Log Loss : 1.499790075899291
for alpha = 10
Log Loss : 1.5399826228150169
```

```
for alpha = 100
Log Loss : 1.5444981879702213
```



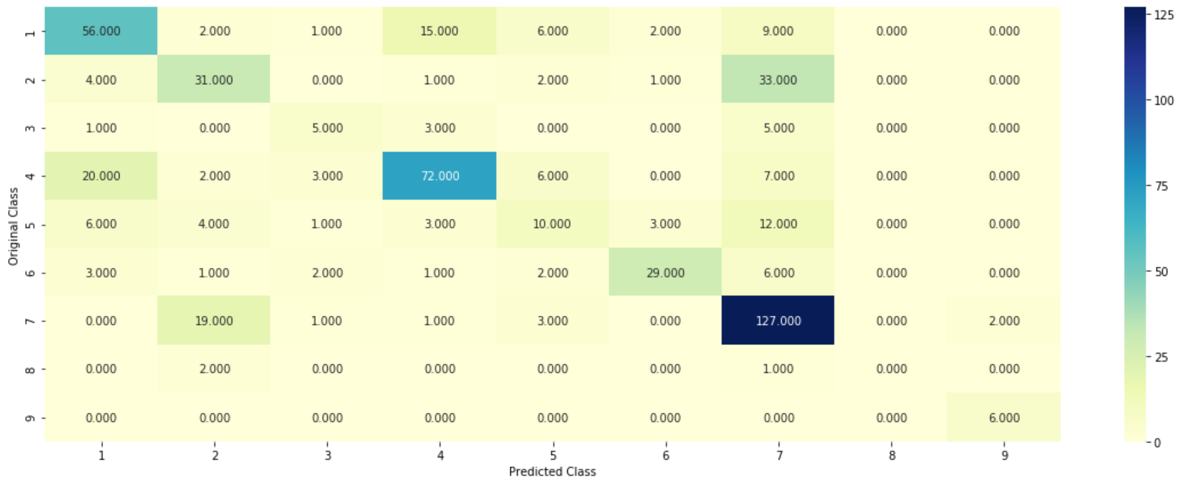
```
For values of best alpha =  0.001 The train log loss is: 0.59613219117
36935
For values of best alpha =  0.001 The cross validation log loss is: 1.
09924835246161
For values of best alpha =  0.001 The test log loss is: 1.084265884007
5965
```

Testing the model with best hyper paramters

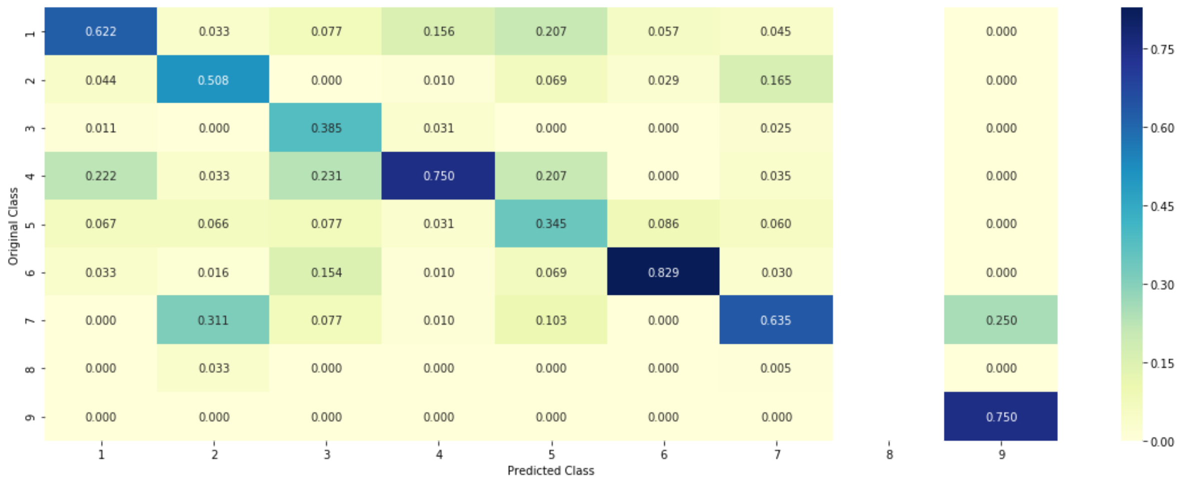
In [65]:

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

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding,
```

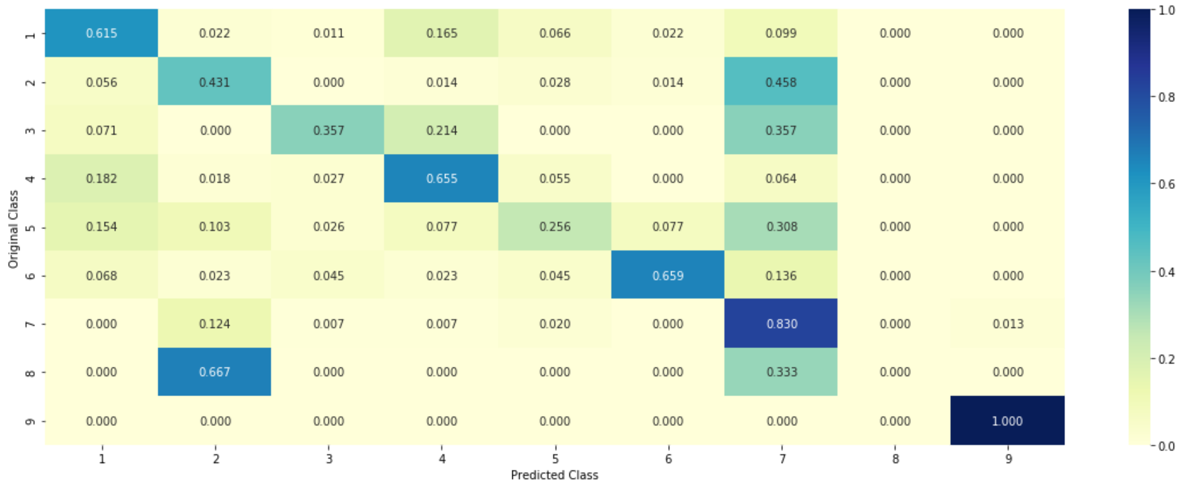
```
Log loss : 1.09924835246161
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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



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



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'
```

Correctly Classified point

In [67]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0051 0.0143 0.0045 0.9416 0.0119 0.
0045 0.009  0.0049 0.0042]]
Actual Class : 4
--------------------------------------------------
0 Text feature [replaced] present in test data point [True]
123 Text feature [00] present in test data point [True]
169 Text feature [aacrjournals] present in test data point [True]
321 Text feature [negatively] present in test data point [True]
428 Text feature [spectrophotometer] present in test data point [True]
448 Text feature [cacl2] present in test data point [True]
455 Text feature [involved] present in test data point [True]
462 Text feature [spreading] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

Incorrectly Classified point

In [68]:

```
test_point_index = 99
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.9256 0.0111 0.0036 0.0109 0.0092 0.
013  0.0164 0.0058 0.0043]]
Actual Class : 1
--------------------------------------------------
16 Text feature [nearly] present in test data point [True]
17 Text feature [complex] present in test data point [True]
41 Text feature [intriguingly] present in test data point [True]
59 Text feature [denzo] present in test data point [True]
65 Text feature [graham] present in test data point [True]
108 Text feature [close] present in test data point [True]
177 Text feature [allowed] present in test data point [True]
213 Text feature [inter] present in test data point [True]
224 Text feature [peak] present in test data point [True]
288 Text feature [compromised] present in test data point [True]
342 Text feature [appearance] present in test data point [True]
396 Text feature [pp2a] present in test data point [True]
409 Text feature [182] present in test data point [True]
421 Text feature [balance] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

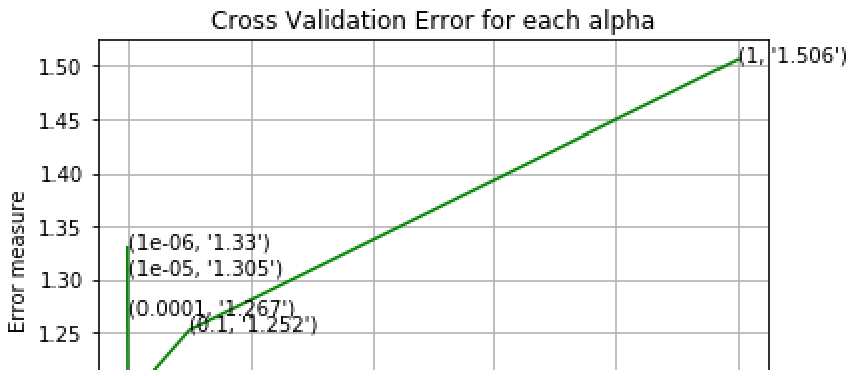
Without Class balancing

Hyper paramter tuning

In [69]:

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

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_, ep
    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
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:",lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
for alpha = 1e-06
Log Loss : 1.3298847440141437
for alpha = 1e-05
Log Loss : 1.3049398437070698
for alpha = 0.0001
Log Loss : 1.2673605146657498
for alpha = 0.001
Log Loss : 1.1334951190938605
for alpha = 0.01
Log Loss : 1.193292839932477
for alpha = 0.1
Log Loss : 1.2521256028106416
for alpha = 1
Log Loss : 1.5060516205626961
```



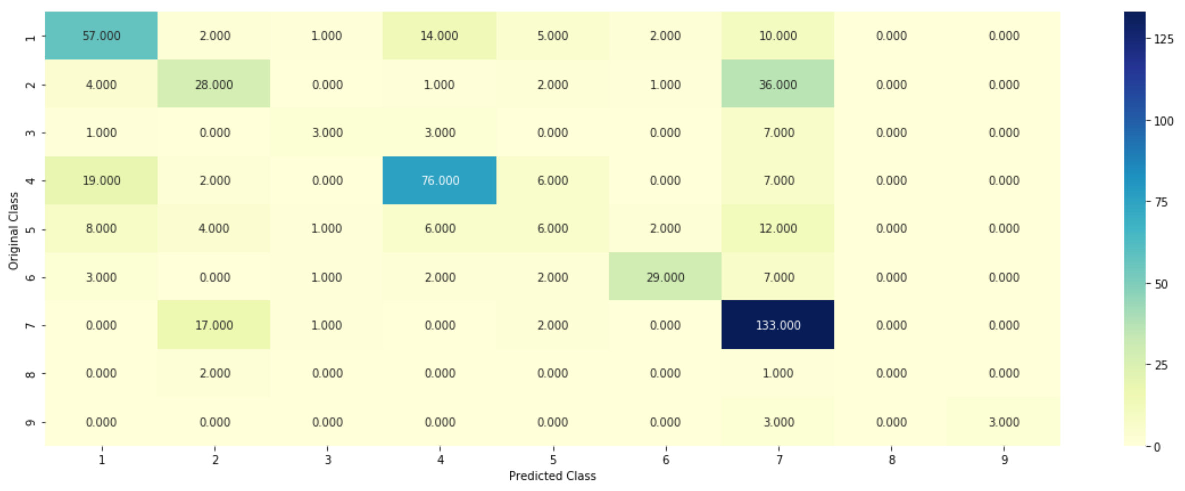
```
For values of best alpha =  0.001 The train log loss is: 0.58455323829
63076
For values of best alpha =  0.001 The cross validation log loss is: 1.
1334951190938605
For values of best alpha =  0.001 The test log loss is: 1.082203484991
387
```

Testing model with best hyper parameters

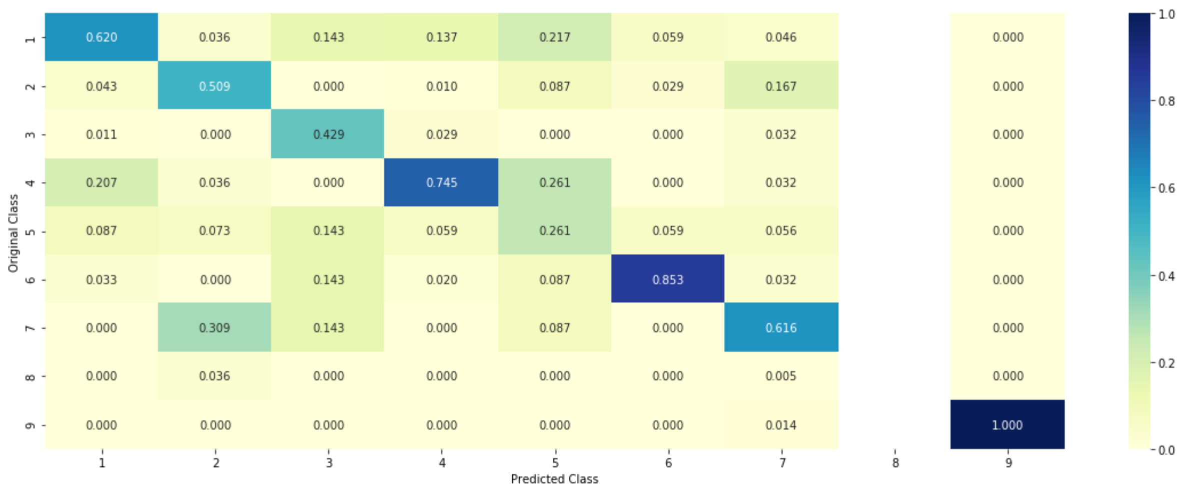
In [70]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generat
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding,
```

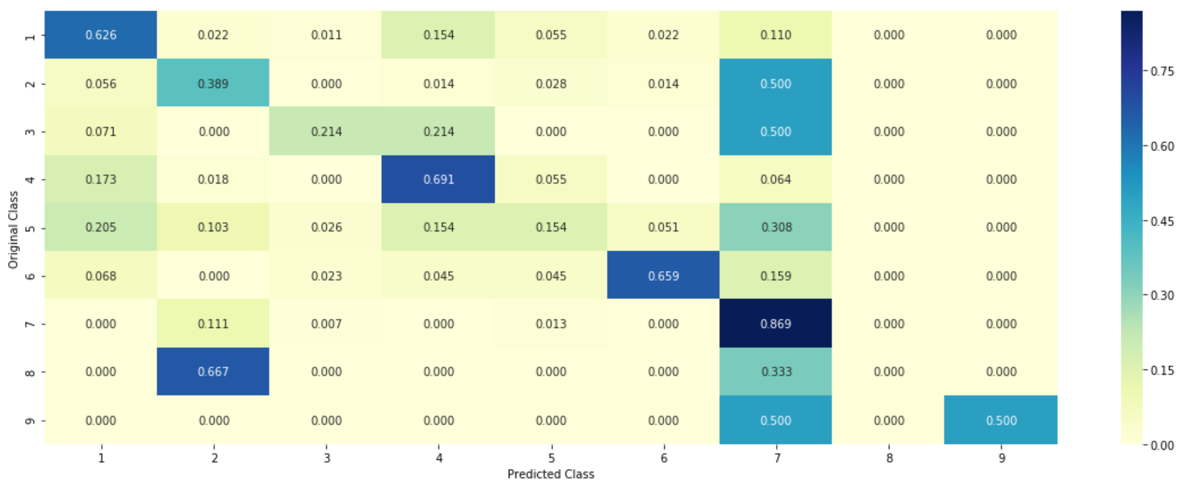
```
Log loss : 1.1334951190938605
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



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



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



Feature Importance, Correctly Classified point

In [71]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 4
Predicted Class Probabilities: [[6.900e-03 1.680e-02 1.500e-03 9.415e-
01 9.300e-03 3.500e-03 1.730e-02
  2.700e-03 4.000e-04]]
Actual Class : 4
--------------------------------------------------
30 Text feature [critical] present in test data point [True]
71 Text feature [determine] present in test data point [True]
337 Text feature [analyses] present in test data point [True]
339 Text feature [mixed] present in test data point [True]
363 Text feature [large] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

Feature Importance, Inorrectly Classified point

In [72]:

```
test_point_index = 99
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.263e-01 1.180e-02 8.000e-04 1.430
e-02 6.200e-03 7.900e-03 3.010e-02
  2.500e-03 2.000e-04]]
Actual Class : 1
--------------------------------------------------
17 Text feature [38] present in test data point [True]
39 Text feature [119] present in test data point [True]
69 Text feature [like] present in test data point [True]
70 Text feature [systematic] present in test data point [True]
78 Text feature [classes] present in test data point [True]
85 Text feature [coincides] present in test data point [True]
91 Text feature [1996] present in test data point [True]
111 Text feature [flexibility] present in test data point [True]
158 Text feature [abundant] present in test data point [True]
182 Text feature [ref] present in test data point [True]
211 Text feature [convex] present in test data point [True]
248 Text feature [appears] present in test data point [True]
268 Text feature [rise] present in test data point [True]
270 Text feature [elongated] present in test data point [True]
```

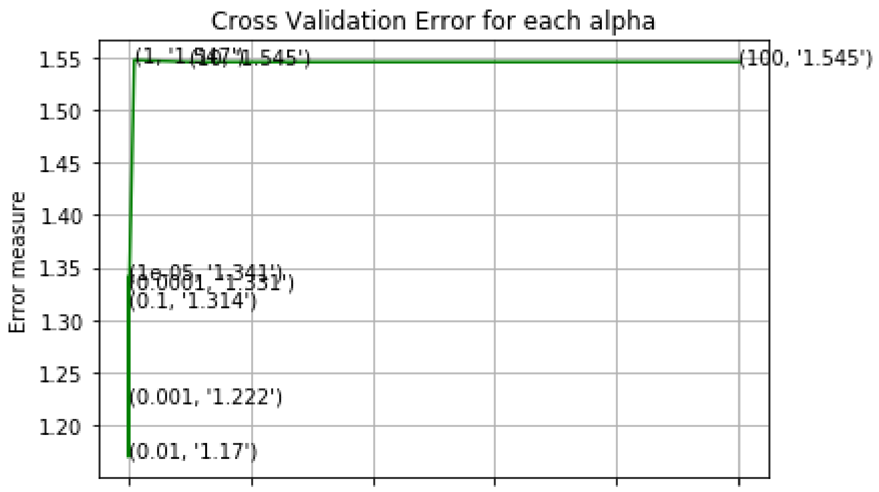
Linear Support Vector Machines

Hyper paramter tuning

In [73]:

```
# read more about support vector machines with linear kernals here http://scikit-le
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='hing
    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
    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',
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:",lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
for C = 1e-05
Log Loss : 1.3408270904508282
for C = 0.0001
Log Loss : 1.3311610891066992
for C = 0.001
Log Loss : 1.2218301274679926
for C = 0.01
Log Loss : 1.1701073258209904
for C = 0.1
Log Loss : 1.314143404461337
for C = 1
Log Loss : 1.5471690559958469
for C = 10
Log Loss : 1.5453307470320574
for C = 100
Log Loss : 1.5453307242281444
```



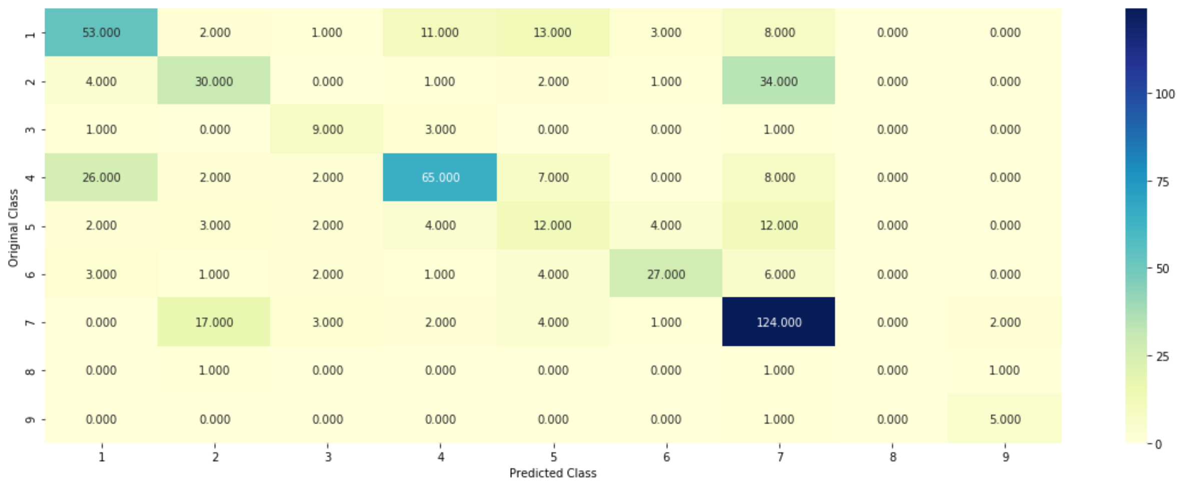
```
For values of best alpha =  0.01 The train log loss is: 0.722158617011
6094
For values of best alpha =  0.01 The cross validation log loss is: 1.1
701073258209904
For values of best alpha =  0.01 The test log loss is: 1.1322502409023
858
```

Testing model with best hyper parameters

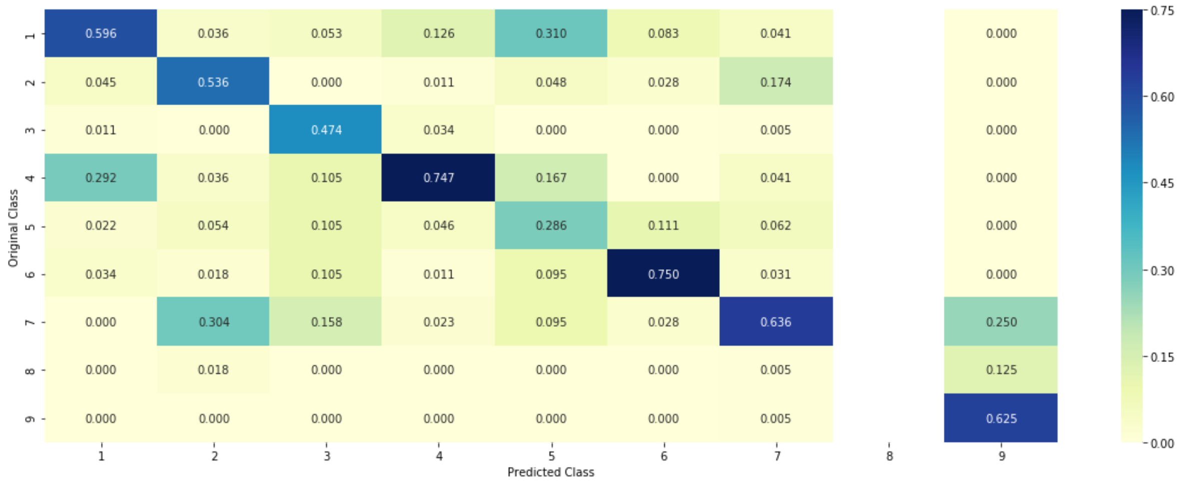
In [74]:

```
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='bal
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_sta
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,c
```

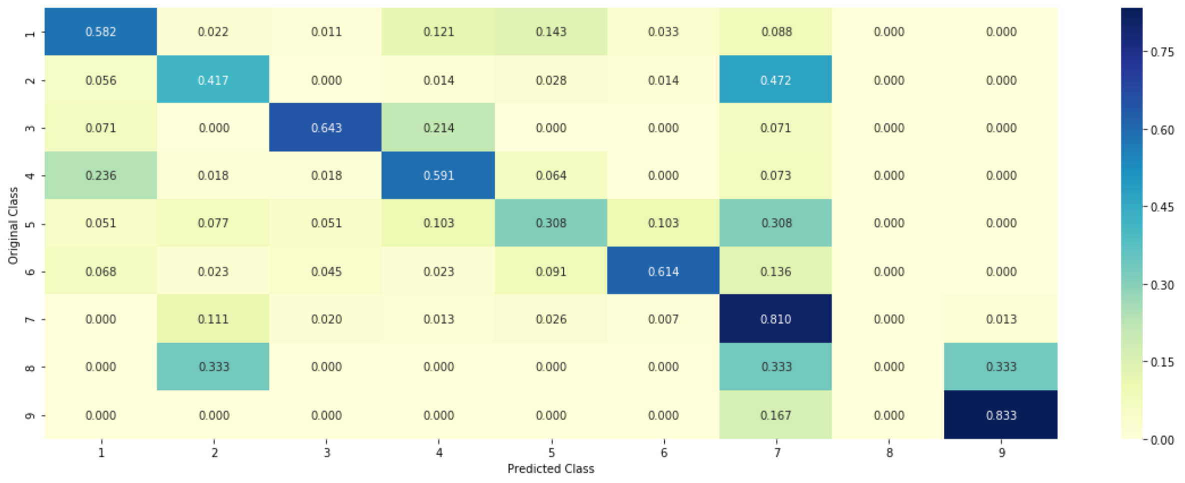
```
Log loss : 1.1701073258209904
Number of mis-classified points : 0.38909774436090223
-------------------- Confusion matrix --------------------
```



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



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



Feature Importance

For Correctly classified point

In [75]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_sta
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0245 0.0283 0.0059 0.8599 0.0229 0.
01   0.0395 0.0042 0.0048]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

For Incorrectly classified point

In [76]:

```
test_point_index = 99
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_oneho
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7649 0.0198 0.0069 0.0623 0.0497 0.
016  0.0675 0.0082 0.0048]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

Random Forest Classifier

Hyper paramter tuning (With TfidfVectorizer)

In [77]:

```
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,
        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_
        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
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
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 l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross valid
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log lo
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2521463278190448
for n_estimators = 100 and max depth =  10
Log Loss : 1.2088014681541068
for n_estimators = 200 and max depth =  5
Log Loss : 1.24714613428645
for n_estimators = 200 and max depth =  10
Log Loss : 1.2020070017934537
for n_estimators = 500 and max depth =  5
Log Loss : 1.2286860782575464
for n_estimators = 500 and max depth =  10
Log Loss : 1.192105886825344
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2251173274847962
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1919522760927803
for n_estimators = 2000 and max depth =  5
Log Loss : 1.222129741702906
for n_estimators = 2000 and max depth =  10
```

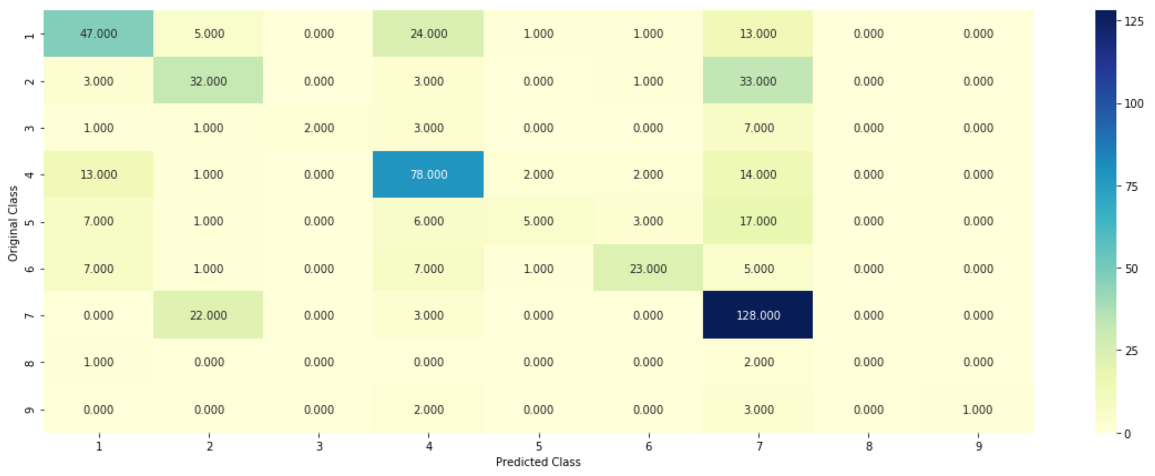
```
Log Loss : 1.1887303024395346
For values of best estimator =  2000 The train log loss is: 0.646037
5266745552
For values of best estimator =  2000 The cross validation log loss i
s: 1.1887303024395344
For values of best estimator =  2000 The test log loss is: 1.1595446
24859724
```

Testing model with best hyper parameters (TfidfVectorizer)

In [78]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,c
```

```
Log loss : 1.1887303024395344
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```



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

Feature Importance

Correctly Classified point

In [79]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini
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_oneho
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],t
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0372 0.018  0.0123 0.8502 0.0318
0.0233 0.0181 0.0036 0.0055]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [phosphorylation] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
8 Text feature [suppressor] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [growth] 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]
19 Text feature [nonsense] present in test data point [True]
20 Text feature [loss] present in test data point [True]
21 Text feature [defective] present in test data point [True]
28 Text feature [cells] present in test data point [True]
32 Text feature [downstream] present in test data point [True]
36 Text feature [treated] present in test data point [True]
37 Text feature [akt] present in test data point [True]
```

Inorrectly Classified point

In [80]:

```
test_point_index = 15
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_oneho
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],t
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0655 0.1482 0.0192 0.0539 0.0436
0.0376 0.6196 0.0051 0.0075]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [phosphorylation] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [activated] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
8 Text feature [suppressor] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [growth] 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]
```

Hyper paramter tuning (With Response Coding)

In [81]:

```
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,
        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_
        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
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
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
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validatio
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss i
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0087653417742977
for n_estimators = 10 and max depth =  3
Log Loss : 1.6758477373257161
for n_estimators = 10 and max depth =  5
Log Loss : 1.4070400151805267
for n_estimators = 10 and max depth =  10
Log Loss : 1.5812340927861825
for n_estimators = 50 and max depth =  2
Log Loss : 1.6678929314897561
for n_estimators = 50 and max depth =  3
Log Loss : 1.4282424247775456
for n_estimators = 50 and max depth =  5
Log Loss : 1.2904853994400503
for n_estimators = 50 and max depth =  10
Log Loss : 1.6747193003901457
for n_estimators = 100 and max depth =  2
Log Loss : 1.5130104408757359
```

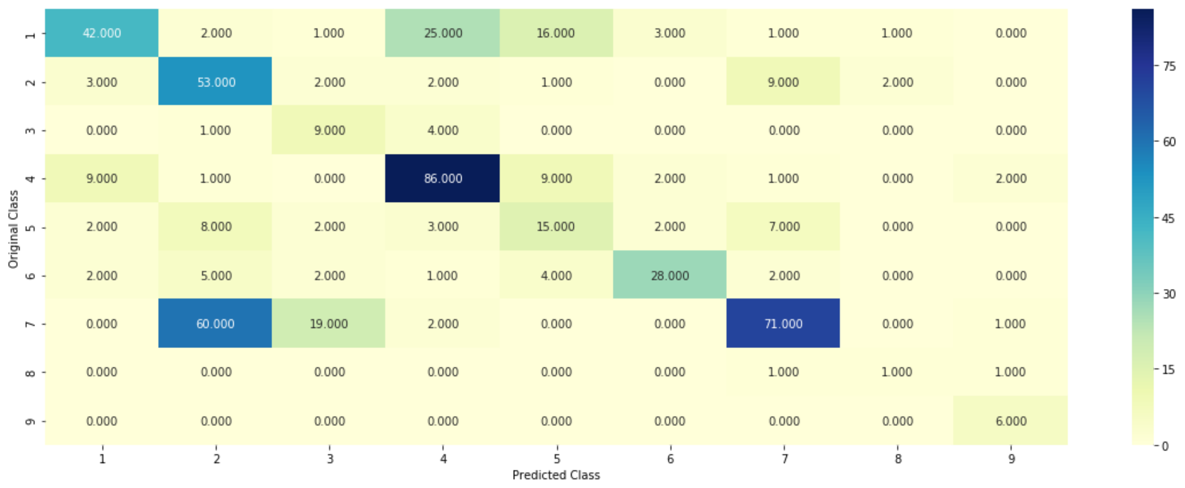
```
for n_estimators = 100 and max depth =  3
```

Testing model with best hyper parameters (Response Coding)

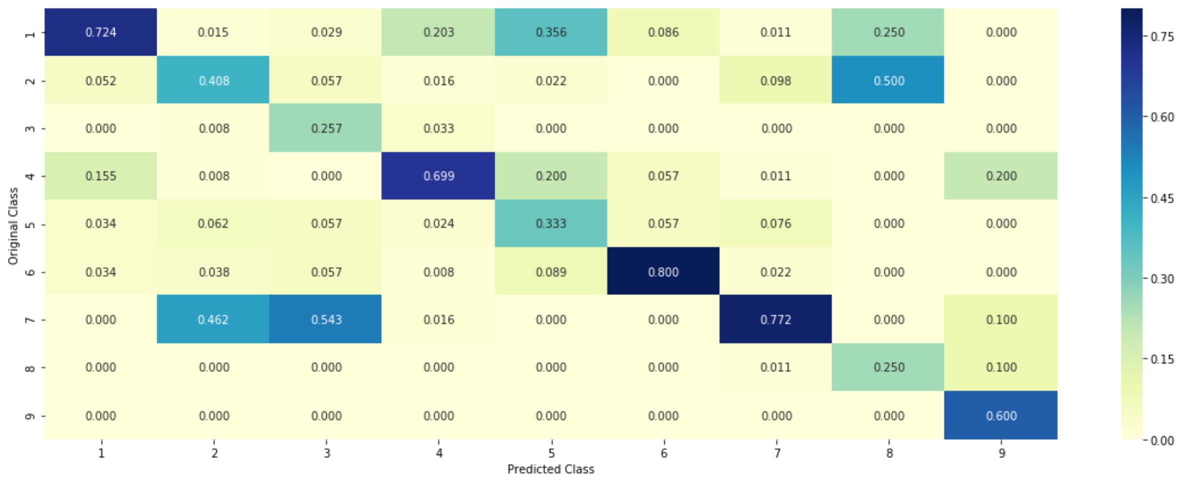
In [82]:

```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=a
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCodi
```

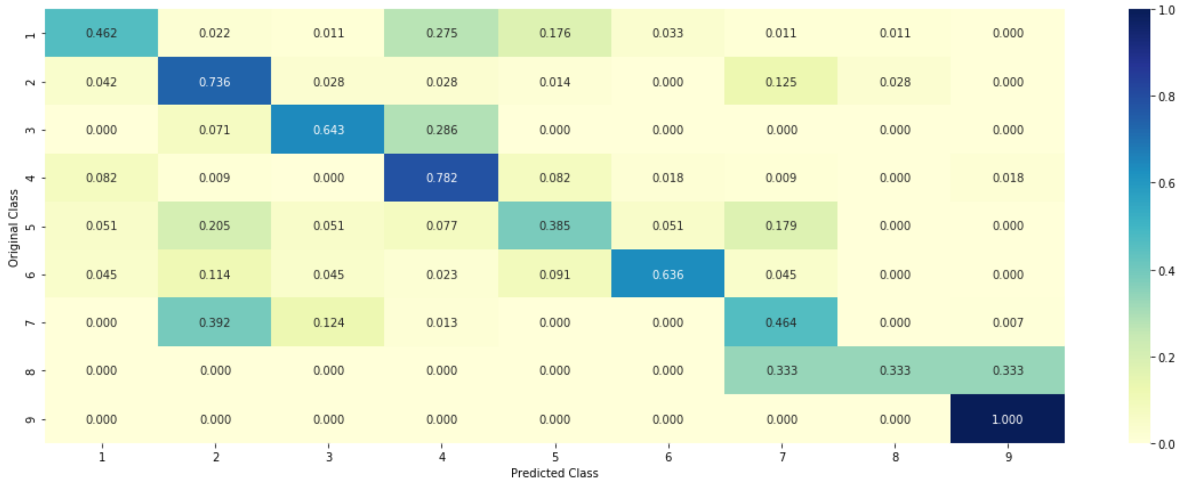
```
Log loss : 1.2715971314637584
Number of mis-classified points : 0.41541353383458646
-------------------- Confusion matrix --------------------
```



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



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



Feature Importance

Correctly Classified point

In [83]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respo
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: [[0.0645 0.0201 0.1567 0.6511 0.0213
0.0335 0.0113 0.0155 0.026 ]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
```

Incorrectly Classified point

In [84]:

```
test_point_index = 13
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respo
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0494 0.1827 0.2462 0.0563 0.0543
0.0762 0.1919 0.0503 0.0927]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
```

Stack the models

testing with hyper parameter tuning

In [85]:

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

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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced',
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_p
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predic
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
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_clas
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, l
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

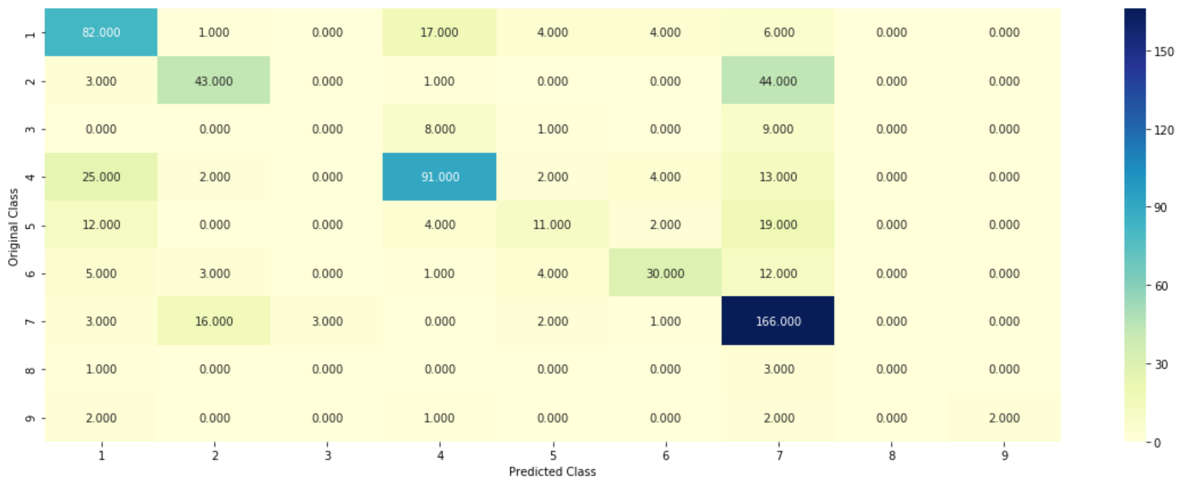
```
Logistic Regression :  Log Loss: 1.10
Support vector machines : Log Loss: 1.55
Naive Bayes : Log Loss: 1.28
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.524
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.148
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.242
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.490
```

testing the model with the best hyper parameters

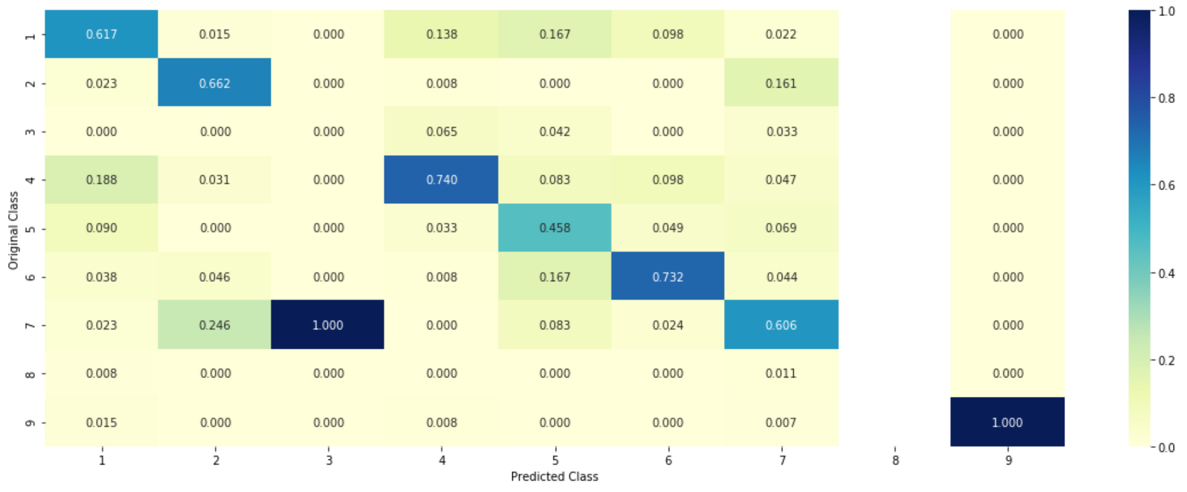
In [86]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifi
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_one
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

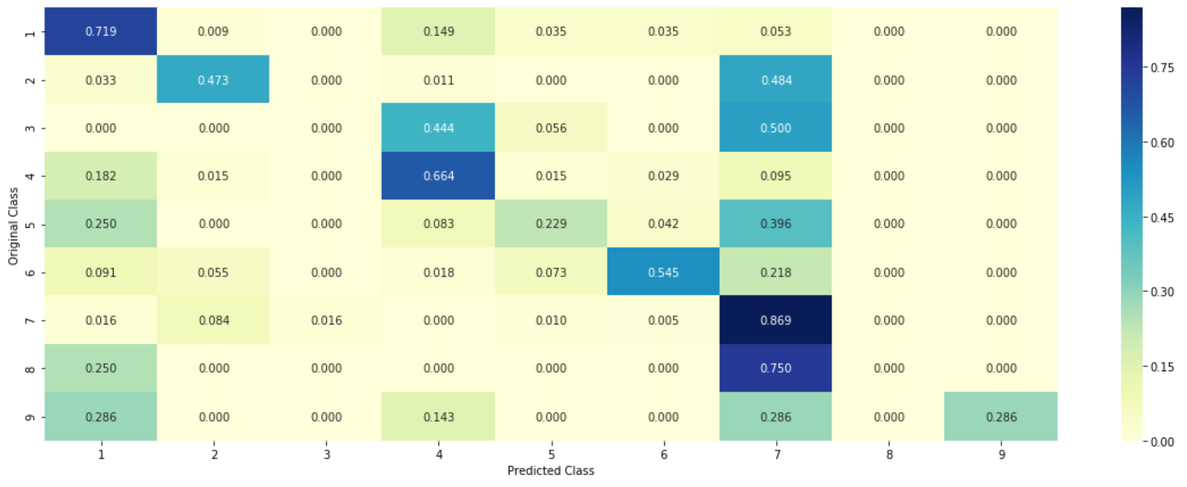
```
Log loss (train) on the stacking classifier : 0.6562941587798546
Log loss (CV) on the stacking classifier : 1.1483030642502194
Log loss (test) on the stacking classifier : 1.12931147835488
Number of missclassified point : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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

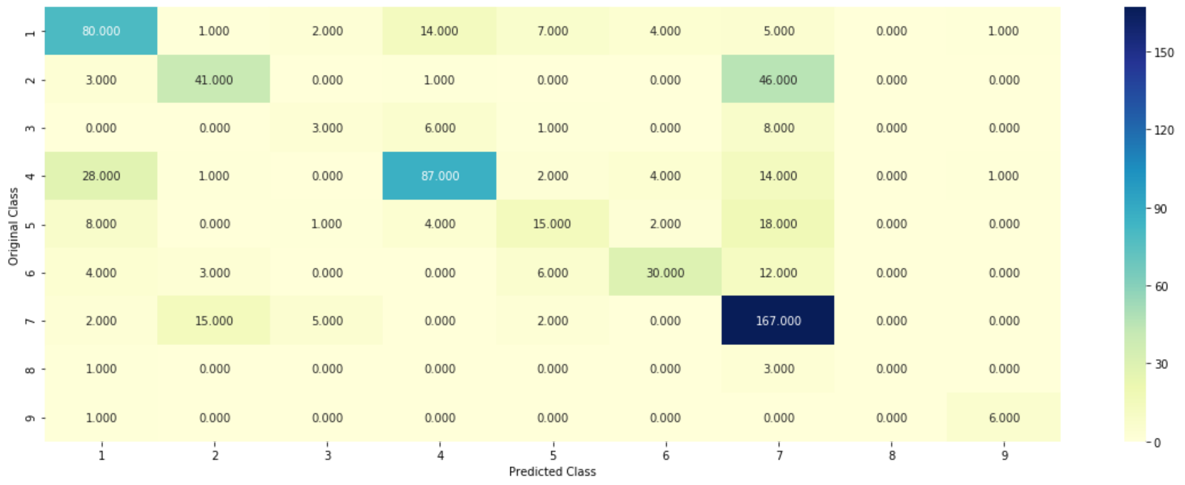


Maximum Voting classifier

In [87]:

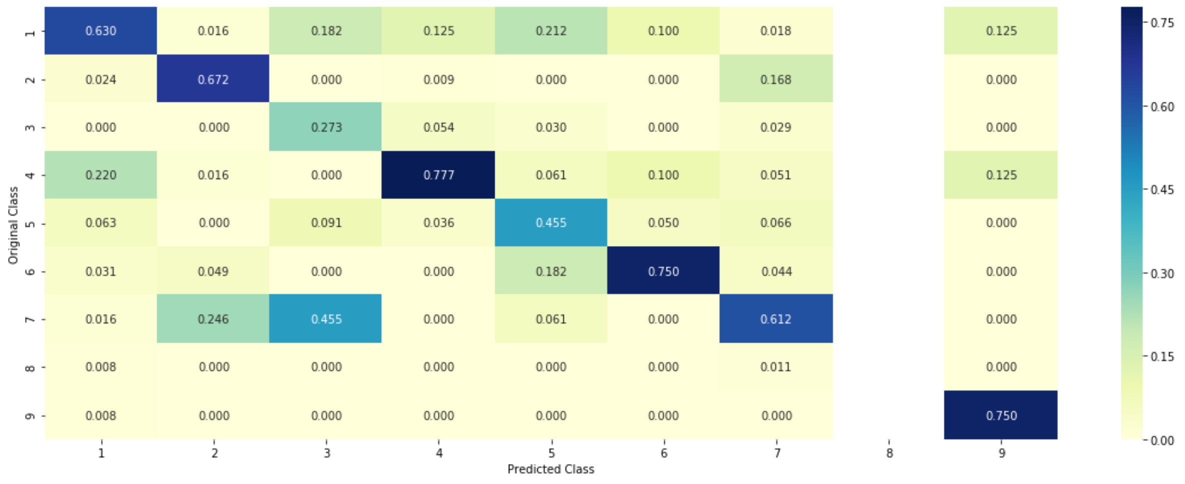
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClas
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier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

```
Log loss (train) on the VotingClassifier : 0.8749189505095902
Log loss (CV) on the VotingClassifier : 1.1787061276474247
Log loss (test) on the VotingClassifier : 1.1437342565752213
Number of missclassified point : 0.3548872180451128
-------------------- Confusion matrix --------------------
```



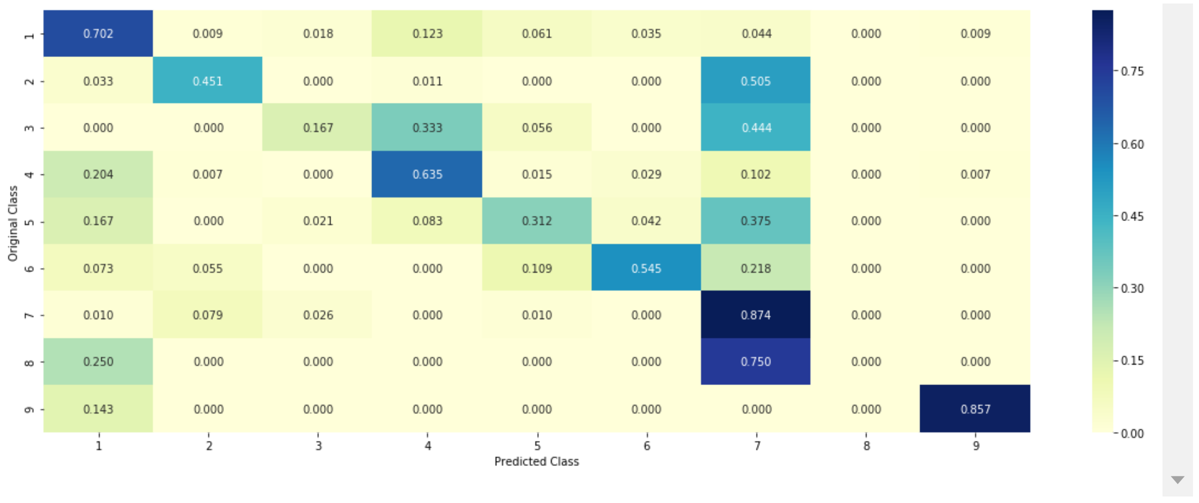
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	80.000	1.000	2.000	14.000	7.000	4.000	5.000	0.000	1.000
2	3.000	41.000	0.000	1.000	0.000	0.000	46.000	0.000	0.000
3	0.000	0.000	3.000	6.000	1.000	0.000	8.000	0.000	0.000
4	28.000	1.000	0.000	87.000	2.000	4.000	14.000	0.000	1.000
5	8.000	0.000	1.000	4.000	15.000	2.000	18.000	0.000	0.000
6	4.000	3.000	0.000	0.000	6.000	30.000	12.000	0.000	0.000
7	2.000	15.000	5.000	0.000	2.000	0.000	167.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	3.000	0.000	0.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.630	0.016	0.182	0.125	0.212	0.100	0.018		0.125
2	0.024	0.672	0.000	0.009	0.000	0.000	0.168		0.000
3	0.000	0.000	0.273	0.054	0.030	0.000	0.029		0.000
4	0.220	0.016	0.000	0.777	0.061	0.100	0.051		0.125
5	0.063	0.000	0.091	0.036	0.455	0.050	0.066		0.000
6	0.031	0.049	0.000	0.000	0.182	0.750	0.044		0.000
7	0.016	0.246	0.455	0.000	0.061	0.000	0.612		0.000
8	0.008	0.000	0.000	0.000	0.000	0.000	0.011		0.000
9	0.008	0.000	0.000	0.000	0.000	0.000	0.000		0.750

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



	1	2	3	4	5	6	7	8	9
1	0.702	0.009	0.018	0.123	0.061	0.035	0.044	0.000	0.009
2	0.033	0.451	0.000	0.011	0.000	0.000	0.505	0.000	0.000
3	0.000	0.000	0.167	0.333	0.056	0.000	0.444	0.000	0.000
4	0.204	0.007	0.000	0.635	0.015	0.029	0.102	0.000	0.007
5	0.167	0.000	0.021	0.083	0.312	0.042	0.375	0.000	0.000
6	0.073	0.055	0.000	0.000	0.109	0.545	0.218	0.000	0.000
7	0.010	0.079	0.026	0.000	0.010	0.000	0.874	0.000	0.000
8	0.250	0.000	0.000	0.000	0.000	0.000	0.750	0.000	0.000
9	0.143	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.857

using only the top 1000 words based of tf-idf values

In [89]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in tra
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding_top = text_vectorizer.fit_transform(train_df['TEXT'
# getting all the feature names (words)
train_text_features_top= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times
text_fea_dict_top = dict(zip(list(train_text_features_top),train_text_fea_counts_to
```

In [90]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding_top = normalize(train_text_feature_onehotCoding_top

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

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

In [91]:

```
# 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_variati
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_featur

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

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

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

Random Forest Classifier on top 1000 words of text feature and on onehot encoding of Genes and Variations features

Hyper paramter tuning (With TfidfVectorizer)

In [92]:

```
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,
        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_
        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
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
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 l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross valid
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log lo
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2440265643350839
for n_estimators = 100 and max depth =  10
Log Loss : 1.2895475198714128
for n_estimators = 200 and max depth =  5
Log Loss : 1.2278996084549505
for n_estimators = 200 and max depth =  10
Log Loss : 1.2746370482149858
for n_estimators = 500 and max depth =  5
Log Loss : 1.215449265212095
for n_estimators = 500 and max depth =  10
Log Loss : 1.2687764556475143
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2146119314092516
```

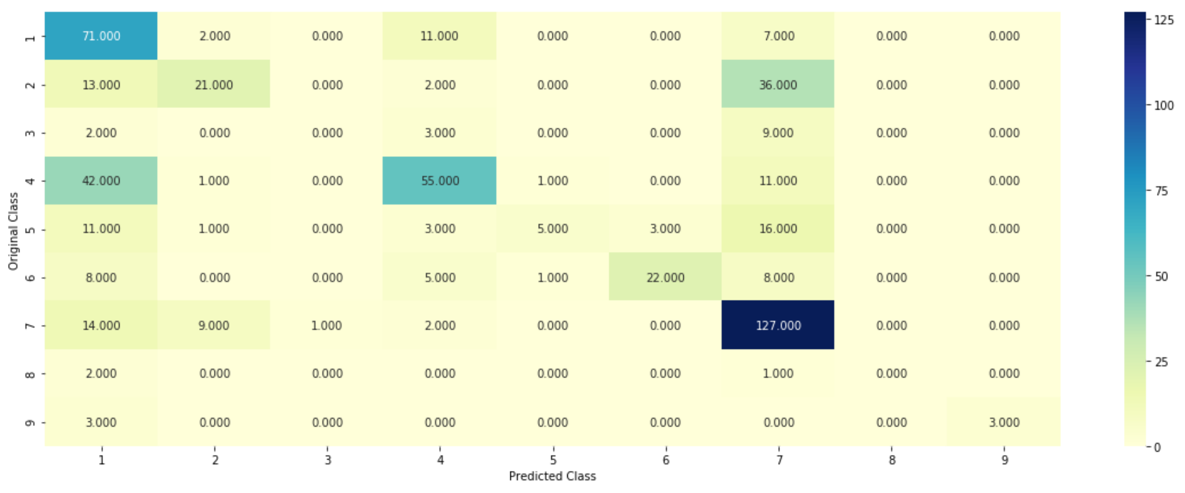
```
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2671528997539183
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2145884735541492
for n_estimators = 2000 and max depth =  10
```

Testing model with best hyper parameters (TfidfVectorizer)

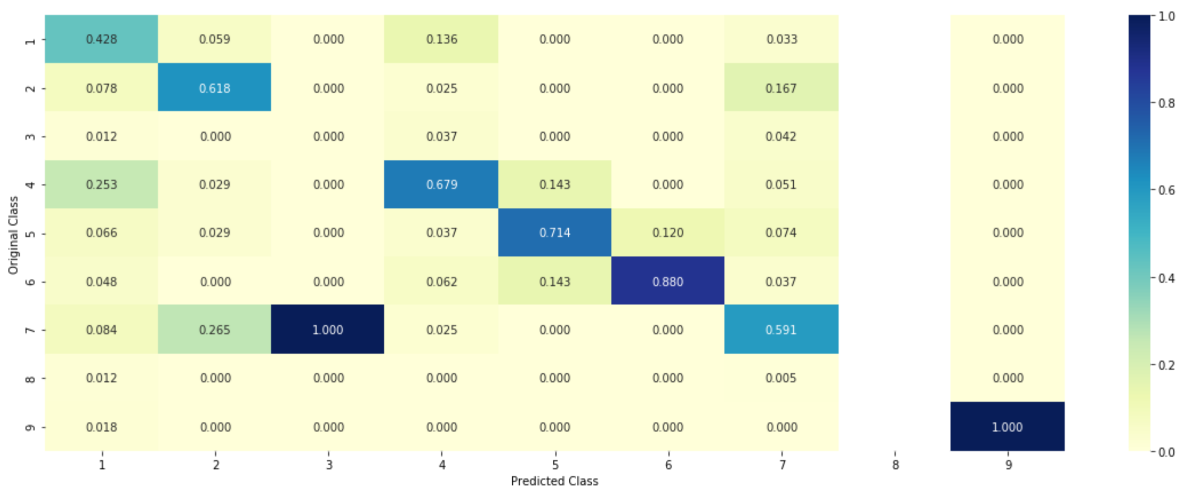
In [93]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,c
```

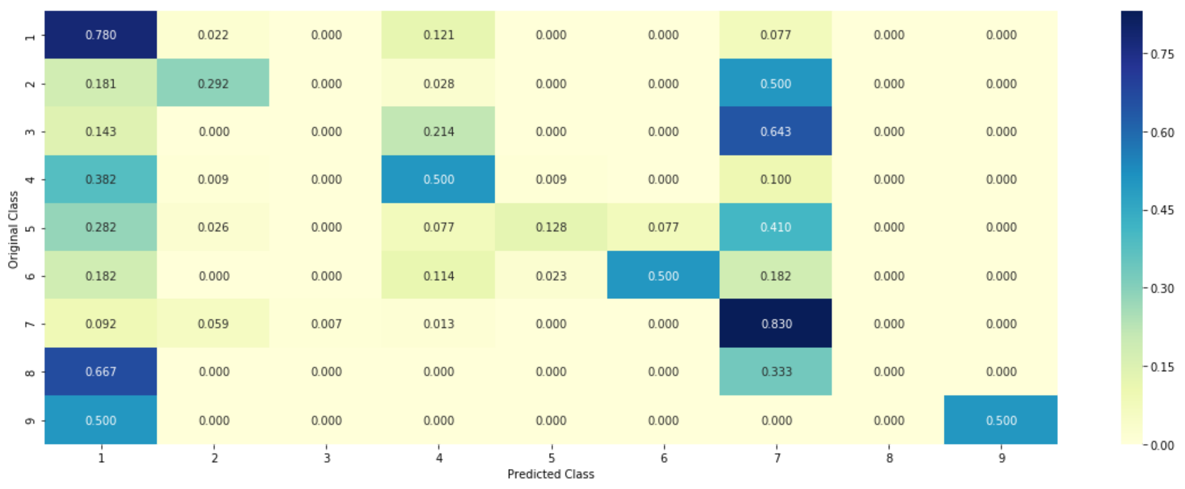
```
Log loss : 1.2145884735541492
Number of mis-classified points : 0.42857142857142855
-------------------- Confusion matrix --------------------
```



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



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



Feature Importance

Correctly Classified point

In [94]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini
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_oneho
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],t
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1046 0.0145 0.0106 0.7662 0.0354 0.
029  0.0314 0.0033 0.005 ]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

for incorrectly classified points

In [95]:

```
test_point_index = 17
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_oneho
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],t
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2299 0.041  0.0672 0.4308 0.0727 0.
0565 0.0851 0.006  0.0108]]
Actuall Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [98]:

```
# building a countvectorizer with all the words that occured minimum 10 times in tr
#considering both unigrams and bi grams
text_vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*
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
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 : 782469
```

In [99]:

```
# 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 [101]:

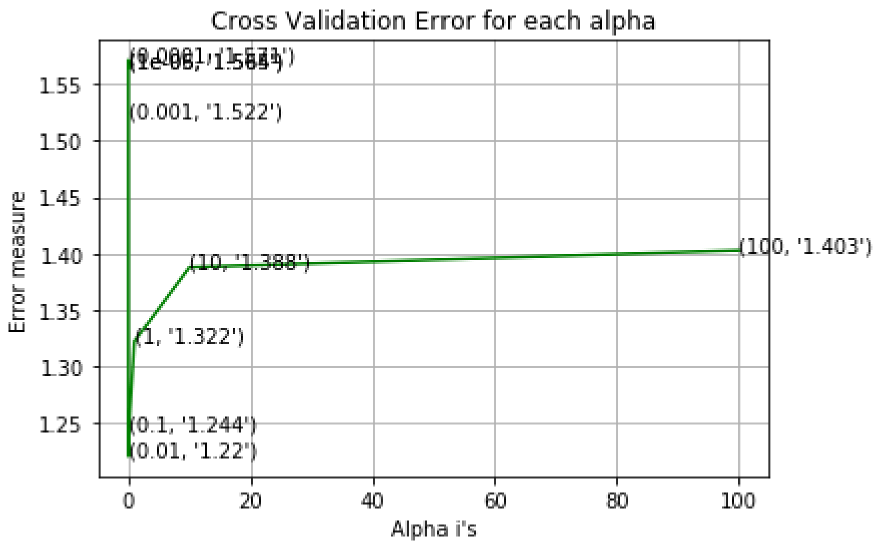
```
# merging gene, variance and text features

# 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_variati
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_featur

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

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

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

LOGISTIC REGRESSION

In [102]:

```
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',
    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
    # to avoid rounding error while multiplying probabilites we use log-probability
    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',
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:",lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log l
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
```

```
for alpha = 1e-06
Log Loss : 1.56445538542886
for alpha = 1e-05
Log Loss : 1.565301863884165
for alpha = 0.0001
Log Loss : 1.5711495725578473
for alpha = 0.001
Log Loss : 1.5215305726379396
for alpha = 0.01
Log Loss : 1.220458621258085
for alpha = 0.1
Log Loss : 1.2437997021924223
for alpha = 1
Log Loss : 1.3219771799509925
for alpha = 10
Log Loss : 1.387746032544035
for alpha = 100
Log Loss : 1.4027061491239279
```

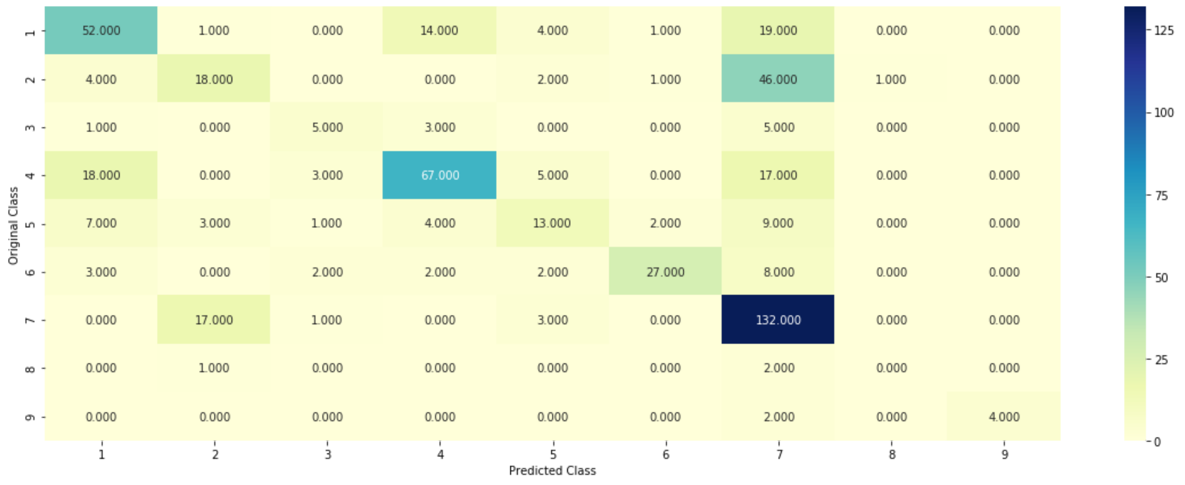


```
For values of best alpha =  0.01 The train log loss is: 0.860444998495
0422
For values of best alpha =  0.01 The cross validation log loss is: 1.2
20458621258085
For values of best alpha =  0.01 The test log loss is: 1.2169956878354
853
```

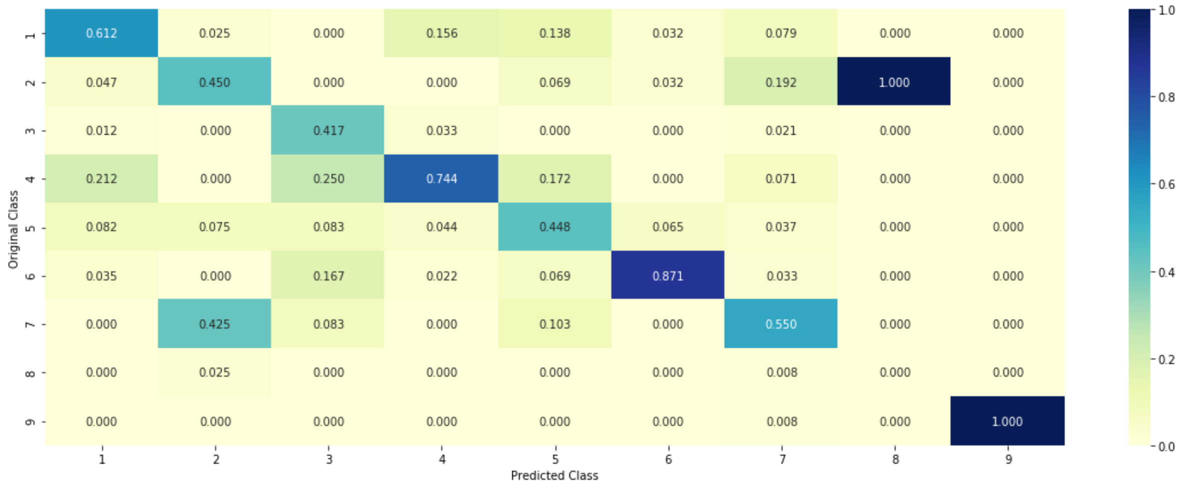
In [104]:

```
Clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding,
```

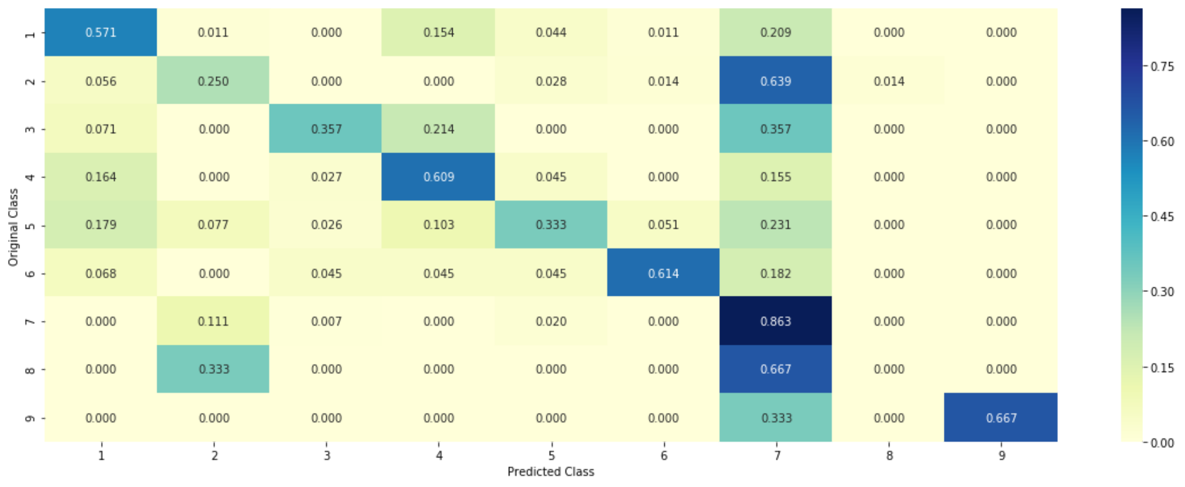
```
Log loss : 1.220458621258085
Number of mis-classified points : 0.40225563909774437
-------------------- Confusion matrix --------------------
```



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



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



Feature Engineering

On Text

In [106]:

```
#https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-
#Number of words

train_df['word_count'] = train_df['TEXT'].apply(lambda x: len(str(x).split(" ")))
cv_df['word_count'] = cv_df['TEXT'].apply(lambda x: len(str(x).split(" ")))
test_df['word_count'] = test_df['TEXT'].apply(lambda x: len(str(x).split(" ")))

#Number of numerics

train_df['numerics'] = train_df['TEXT'].apply(lambda x: len([x for x in x.split() i
cv_df['numerics'] = cv_df['TEXT'].apply(lambda x: len([x for x in x.split() if x.is
test_df['numerics'] = test_df['TEXT'].apply(lambda x: len([x for x in x.split() if
```

On categorical values

In [117]:

```
#gene and variation length
train_df['gene_length'] = train_df['Gene'].apply(lambda x: len(str(x)))
cv_df['gene_length'] = cv_df['Gene'].apply(lambda x: len(str(x)))
test_df['gene_length'] = test_df['Gene'].apply(lambda x: len(str(x)))

train_df['variation_length'] = train_df['Variation'].apply(lambda x: len(str(x)))
cv_df['variation_length'] = cv_df['Variation'].apply(lambda x: len(str(x)))
test_df['variation_length'] = test_df['Variation'].apply(lambda x: len(str(x)))
```

In [129]:

```
#python code to check if string contains a number
#https://stackoverflow.com/questions/19859282/check-if-a-string-contains-a-number
#numeric presence in gene and variation
import re

train_df['gene_numerics'] = train_df['Gene'].apply(lambda x: 1 if bool(re.search(r'
cv_df['gene_numerics'] = cv_df['Gene'].apply(lambda x: 1 if bool(re.search(r'\d', x
test_df['gene_numerics'] = test_df['Gene'].apply(lambda x: 1 if bool(re.search(r'\d

train_df['variation_numerics'] = train_df['Variation'].apply(lambda x: 1 if bool(re
cv_df['variation_numerics'] = cv_df['Variation'].apply(lambda x: 1 if bool(re.searc
test_df['variation_numerics'] = test_df['Variation'].apply(lambda x: 1 if bool(re.s
```

In [131]:

```
train_df[["Gene","Variation","gene_numerics","variation_numerics"]].head(5)
```

Out[131]:

	Gene	Variation	gene_numerics	variation_numerics
2685	BRAF	T599_V600insV	0	1
2717	BRAF	D594Y	0	1
506	TP53	H214Q	1	1
2537	BRCA1	C24R	1	1
359	EP300	Deletion	1	0

In [136]:

```
#how to substract two columns in a dataframe
#https://stackoverflow.com/questions/48350850/subtract-two-columns-in-dataframe
#difference in length of gene and variation

train_df['gene_variation_diff'] = abs(train_df["gene_length"]-train_df["variation_l
cv_df['gene_variation_diff'] = abs(cv_df["gene_length"]-cv_df["variation_length"])
test_df['gene_variation_diff'] = abs(test_df["gene_length"]-test_df["variation_leng
```

In [137]:

```
train_df.head(5)
```

Out[137]:

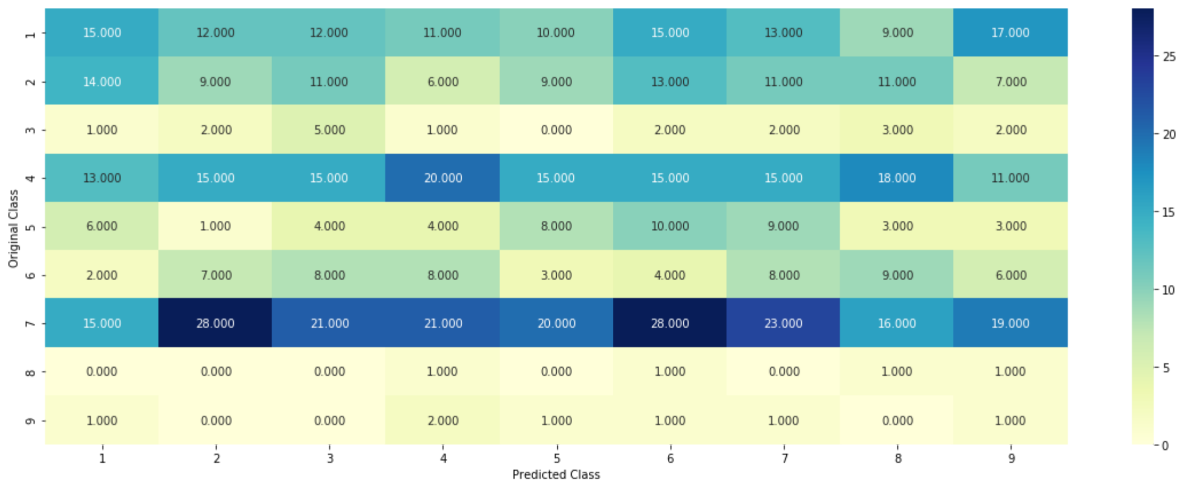
	ID	Gene	Variation	Class	TEXT	word_count	numerics	gene_lengt
2685	2685	BRAF	T599_V600insV	7	abstract activating mutations braf gene common...	2232	139	
2717	2717	BRAF	D594Y	2	braf mutations found subset non small cell lun...	7772	538	
506	506	TP53	H214Q	1	tumor suppressor p53 dependent apoptosis thoug...	8100	759	
2537	2537	BRCA1	C24R	4	published analyses effects missense mutations ...	5389	429	
359	359	EP300	Deletion	1	ep300 protein histone acetyltransferase regula...	4729	333	

Prediction using a 'Random' Model to check max log loss

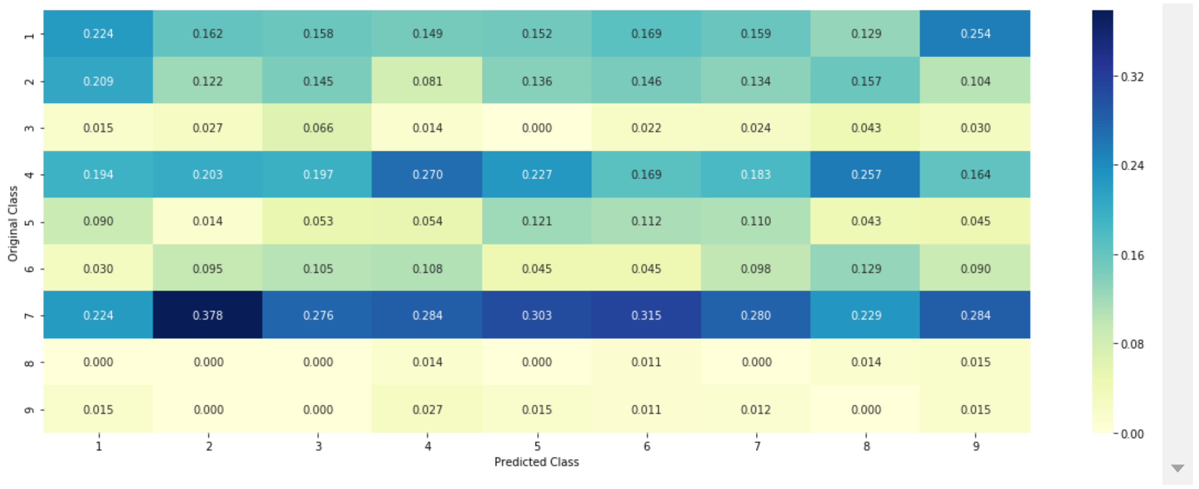
In [138]:

```
# 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_predi


# 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,

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

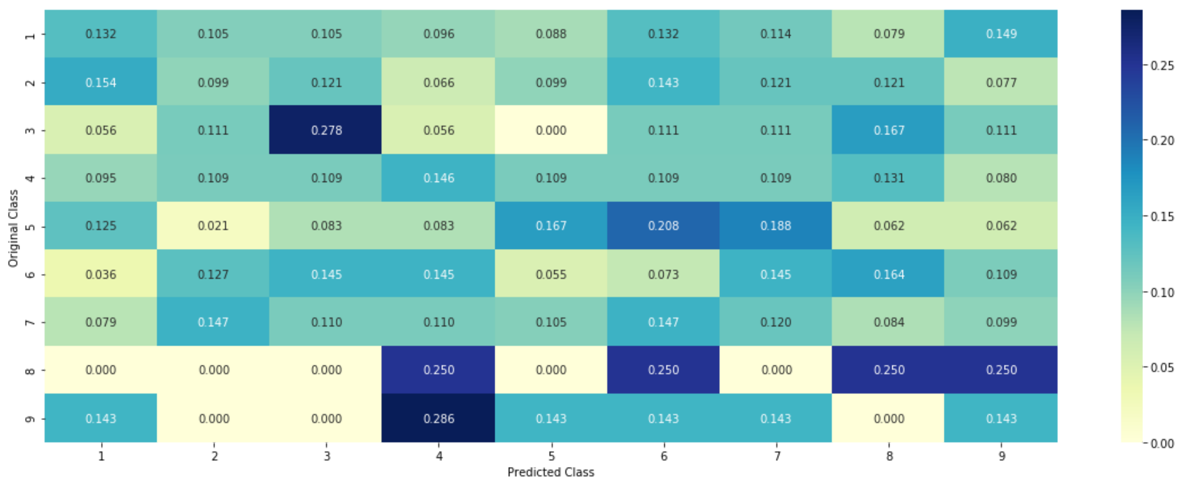
```
Log loss on Cross Validation Data using Random Model 2.442059198735931
2
Log loss on Test Data using Random Model 2.4466246313506983
-------------------- Confusion matrix --------------------
```



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



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



In [140]:

```
train_data=train_df
cv_data = cv_df
test_data = test_df
```

In [144]:

```
train_data.drop(['ID','TEXT','Class','Variation','Gene'],axis=1,inplace=True)
cv_data.drop(['ID','TEXT','Class','Variation','Gene'],axis=1,inplace=True)
test_data.drop(['ID','TEXT','Class','Variation','Gene'],axis=1,inplace=True)
```

In [161]:

```
train_data.columns
```

Out[161]:

```
Index(['word_count', 'numerics', 'gene_length', 'variation_length',
       'gene_numerics', 'variation_numerics', 'gene_variation_diff'],
      dtype='object')
```

Stacking Features

In [176]:

```
#https://github.com/Prakhar-FF13/Personalized-Cancer-Diagnosis/blob/master/Feature%
#https://stackoverflow.com/questions/36967666/transform-scipy-sparse-csr-to-pandas
# scaling the text_count feature
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
train_data["word_count"] = scaler.fit_transform(train_data["word_count"].values.res
test_data["word_count"] = scaler.fit_transform(test_data["word_count"].values.resha
cv_data["word_count"] = scaler.fit_transform(cv_data["word_count"].values.reshape(-

train_data["numerics"] = scaler.fit_transform(train_data["numerics"].values.reshape
test_data["numerics"] = scaler.fit_transform(test_data["numerics"].values.reshape(-
cv_data["numerics"] = scaler.fit_transform(cv_data["numerics"].values.reshape(-1,1)

train_data["gene_length"] = scaler.fit_transform(train_data["gene_length"].values.r
test_data["gene_length"] = scaler.fit_transform(test_data["gene_length"].values.res
cv_data["gene_length"] = scaler.fit_transform(cv_data["gene_length"].values.reshape

train_data["variation_length"] = scaler.fit_transform(train_data["variation_length"
test_data["variation_length"] = scaler.fit_transform(test_data["variation_length"].
cv_data["variation_length"] = scaler.fit_transform(cv_data["variation_length"].valu

train_data["gene_variation_diff"] = scaler.fit_transform(train_data["gene_variation
test_data["gene_variation_diff"] = scaler.fit_transform(test_data["gene_variation_d
cv_data["gene_variation_diff"] = scaler.fit_transform(cv_data["gene_variation_diff"


#onehot encoding

df_gene_var_train = pd.concat([pd.DataFrame(train_gene_feature_onehotCoding.todense
df_gene_var_test = pd.concat([pd.DataFrame(test_gene_feature_onehotCoding.todense()
df_gene_var_cv = pd.concat([pd.DataFrame(cv_gene_feature_onehotCoding.todense()), p

df_train = pd.concat([df_gene_var_train, pd.DataFrame(train_text_feature_onehotCodi
df_test = pd.concat([df_gene_var_test, pd.DataFrame(test_text_feature_onehotCoding_
df_cv = pd.concat([df_gene_var_cv, pd.DataFrame(cv_text_feature_onehotCoding_top.to

df_train["word_count"] = train_data.word_count.values
df_train["numerics"] = train_data.numerics.values
df_train["gene_length"] = train_data.gene_length.values
df_train["variation_length"] = train_data.variation_length.values
df_train["gene_variation_diff"] = train_data.gene_variation_diff.values
df_train["variation_numerics"] = train_data.variation_numerics.values
df_train["gene_numerics"] = train_data.gene_numerics.values


df_cv["word_count"] = cv_data.word_count.values
df_cv["numerics"] = cv_data.numerics.values
df_cv["gene_length"] = cv_data.gene_length.values
df_cv["variation_length"] = cv_data.variation_length.values
df_cv["gene_variation_diff"] = cv_data.gene_variation_diff.values
df_cv["variation_numerics"] = cv_data.variation_numerics.values
df_cv["gene_numerics"] = cv_data.gene_numerics.values


df_test["word_count"] = test_data.word_count.values
df_test["numerics"] = test_data.numerics.values
df_test["gene_length"] = test_data.gene_length.values
df_test["variation_length"] = test_data.variation_length.values
df_test["gene_variation_diff"] = test_data.gene_variation_diff.values
```

```
df_test["variation_numerics"] = test_data.variation_numerics.values
df_test["gene_numerics"] = test_data.gene_numerics.values

train_x_onehotencode=df_train
cv_x_onehotencode=df_cv
test_x_onehotencode=df_test

#response endcoding

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_var
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_featu
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_respon

train_x_responseCoding = np.hstack((train_x_responseCoding,train_data))
cv_x_responseCoding = np.hstack((cv_x_responseCoding,cv_data))
test_x_responseCoding = np.hstack((test_x_responseCoding,test_data))
```

In [179]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_oneh
print("(number of data points * number of features) in test data = ", test_x_onehot
print("(number of data points * number of features) in cross validation data =", cv
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3
205)
(number of data points * number of features) in test data =  (665, 320
5)
(number of data points * number of features) in cross validation data
= (532, 3205)
```

In [180]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_resp
print("(number of data points * number of features) in test data = ", test_x_respon
print("(number of data points * number of features) in cross validation data =", cv
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 3
4)
(number of data points * number of features) in test data =  (665, 34)
(number of data points * number of features) in cross validation data
= (532, 34)
```

Logistic Regression On Engineered Features

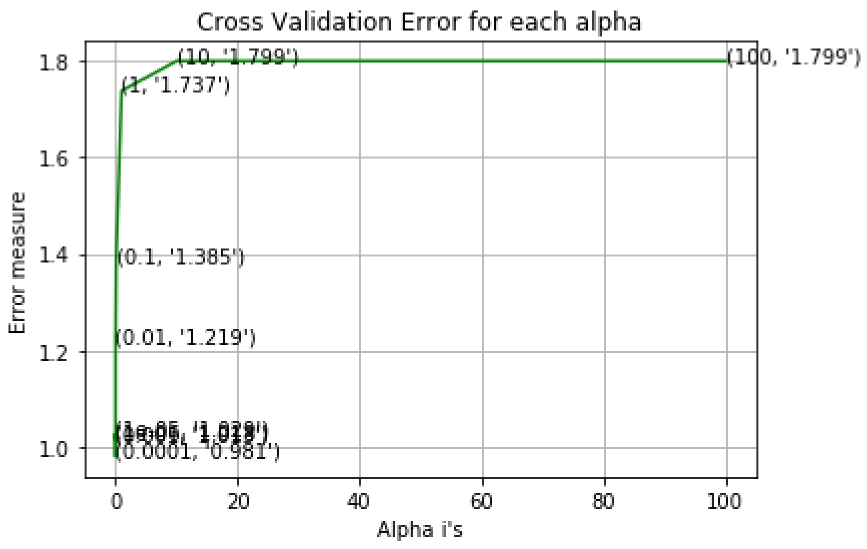
Engineered and onehot encoded features

In [184]:

```
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',
    clf.fit(train_x_onehotencode, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotencode, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotencode)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, ep
    # to avoid rounding error while multiplying probabilites we use log-probability
    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',
clf.fit(train_x_onehotencode, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotencode, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.0179201894202645
for alpha = 1e-05
Log Loss : 1.0288128137639871
for alpha = 0.0001
Log Loss : 0.9805687275739856
for alpha = 0.001
Log Loss : 1.013286283043081
for alpha = 0.01
Log Loss : 1.2190683491438405
```

```
for alpha = 0.1
Log Loss : 1.3850838379188148
for alpha = 1
Log Loss : 1.737377369579716
for alpha = 10
Log Loss : 1.7990321195206689
for alpha = 100
Log Loss : 1.7989750935579738
```

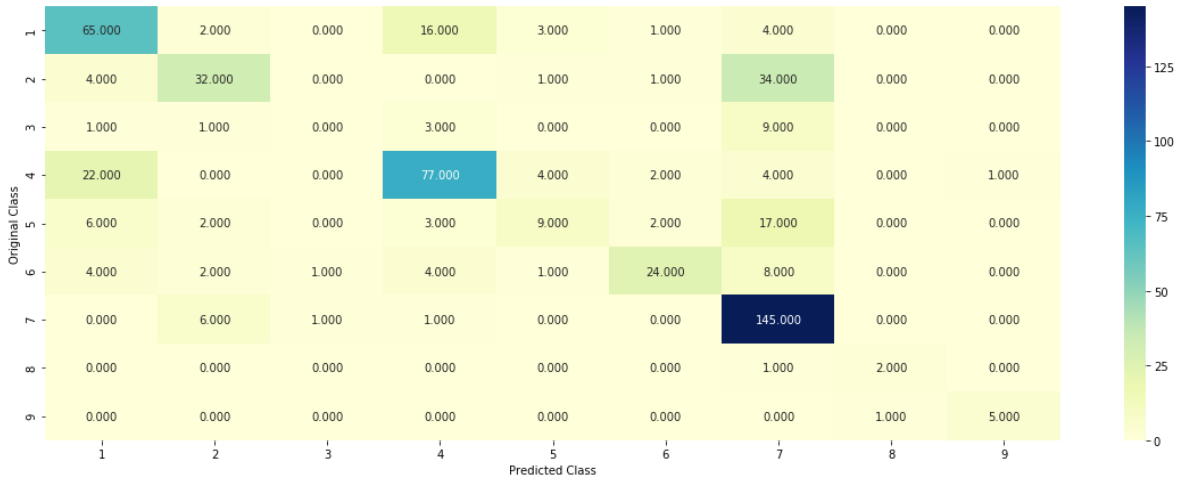


```
For values of best alpha =  0.0001 The train log loss is: 0.4687154598
1574075
For values of best alpha =  0.0001 The cross validation log loss is:
0.9805687275739856
For values of best alpha =  0.0001 The test log loss is: 0.98273517545
01723
```

In [185]:

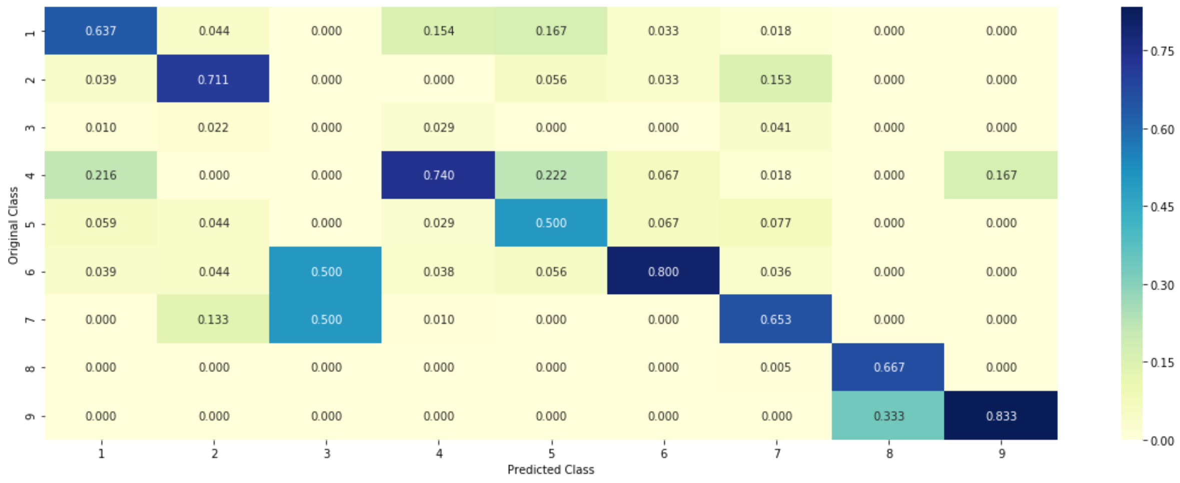
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
predict_and_plot_confusion_matrix(train_x_onehotencode, train_y, cv_x_onehotencode,
```

```
Log loss : 0.98056872757739856
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```



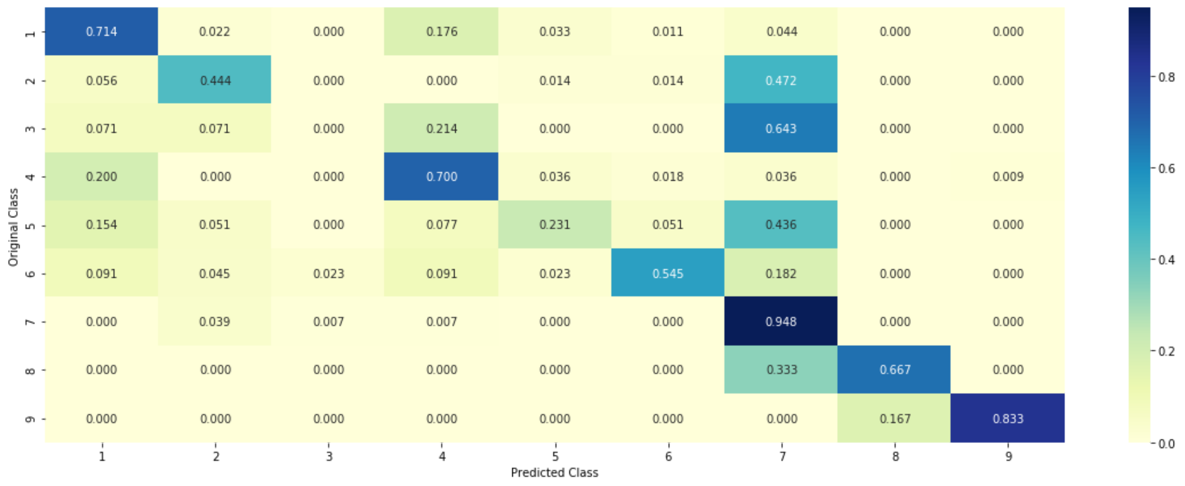
	1	2	3	4	5	6	7	8	9
1	65.000	2.000	0.000	16.000	3.000	1.000	4.000	0.000	0.000
2	4.000	32.000	0.000	0.000	1.000	1.000	34.000	0.000	0.000
3	1.000	1.000	0.000	3.000	0.000	0.000	9.000	0.000	0.000
4	22.000	0.000	0.000	77.000	4.000	2.000	4.000	0.000	1.000
5	6.000	2.000	0.000	3.000	9.000	2.000	17.000	0.000	0.000
6	4.000	2.000	1.000	4.000	1.000	24.000	8.000	0.000	0.000
7	0.000	6.000	1.000	1.000	0.000	0.000	145.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.637	0.044	0.000	0.154	0.167	0.033	0.018	0.000	0.000
2	0.039	0.711	0.000	0.000	0.056	0.033	0.153	0.000	0.000
3	0.010	0.022	0.000	0.029	0.000	0.000	0.041	0.000	0.000
4	0.216	0.000	0.000	0.740	0.222	0.067	0.018	0.000	0.167
5	0.059	0.044	0.000	0.029	0.500	0.067	0.077	0.000	0.000
6	0.039	0.044	0.500	0.038	0.056	0.800	0.036	0.000	0.000
7	0.000	0.133	0.500	0.010	0.000	0.000	0.653	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.667	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.833

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



	1	2	3	4	5	6	7	8	9
1	0.714	0.022	0.000	0.176	0.033	0.011	0.044	0.000	0.000
2	0.056	0.444	0.000	0.000	0.014	0.014	0.472	0.000	0.000
3	0.071	0.071	0.000	0.214	0.000	0.000	0.643	0.000	0.000
4	0.200	0.000	0.000	0.700	0.036	0.018	0.036	0.000	0.009
5	0.154	0.051	0.000	0.077	0.231	0.051	0.436	0.000	0.000
6	0.091	0.045	0.023	0.091	0.023	0.545	0.182	0.000	0.000
7	0.000	0.039	0.007	0.007	0.000	0.000	0.948	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.167	0.833

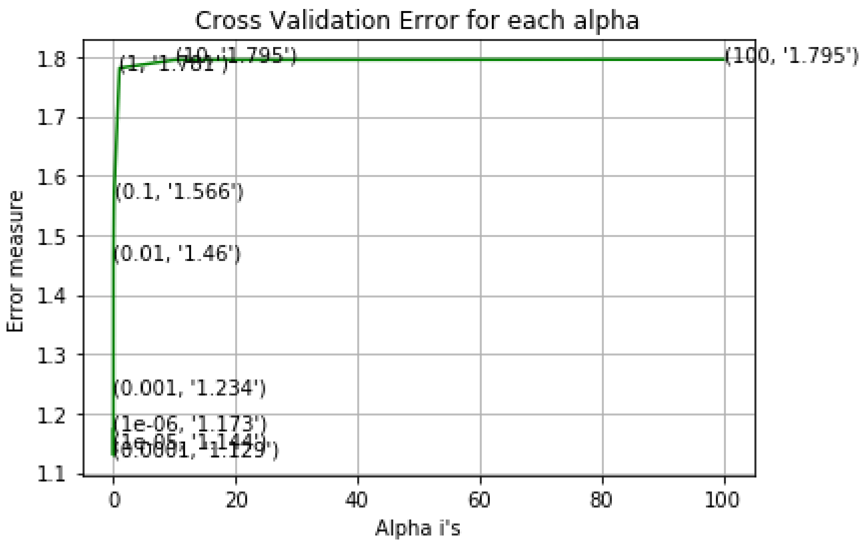
Logistic Regression on engineered and response coded features

In [186]:

```
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',
    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
    # to avoid rounding error while multiplying probabilites we use log-probability
    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',
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_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.1728263407968984
for alpha = 1e-05
Log Loss : 1.1437125743931786
for alpha = 0.0001
Log Loss : 1.1292499624423287
for alpha = 0.001
Log Loss : 1.2340223114089235
for alpha = 0.01
Log Loss : 1.4602751517483918
for alpha = 0.1
```

```
Log Loss : 1.566076314207619
for alpha = 1
Log Loss : 1.7809547492946003
for alpha = 10
Log Loss : 1.7946567649761604
for alpha = 100
Log Loss : 1.795319730122491
```

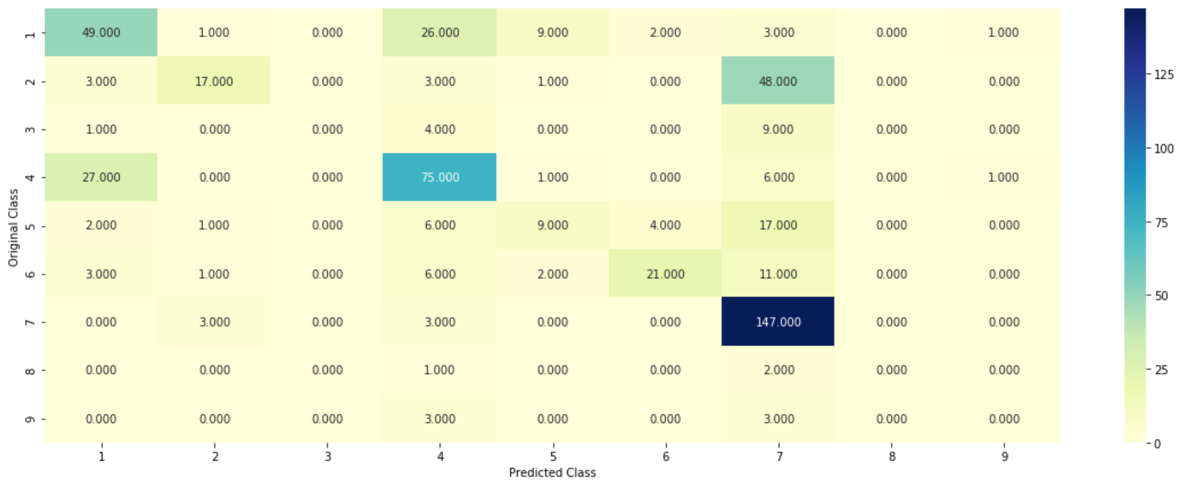


```
For values of best alpha =  0.0001 The train log loss is: 0.9914607246
091152
For values of best alpha =  0.0001 The cross validation log loss is:
1.1292499624423287
For values of best alpha =  0.0001 The test log loss is: 1.17179705603
38686
```

In [187]:

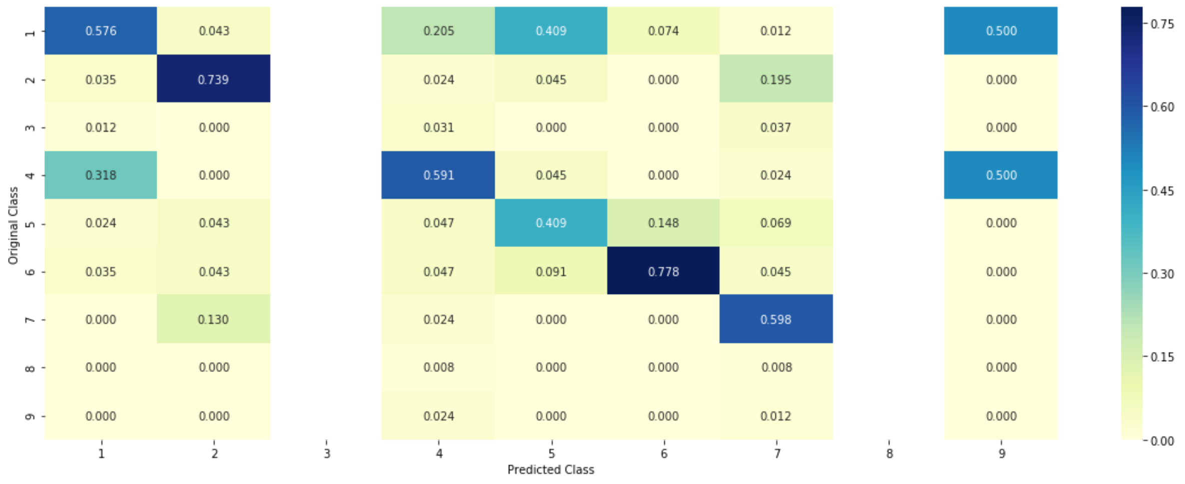
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCod
```

```
Log loss : 1.1292499624423287
Number of mis-classified points : 0.40225563909774437
-------------------- Confusion matrix --------------------
```



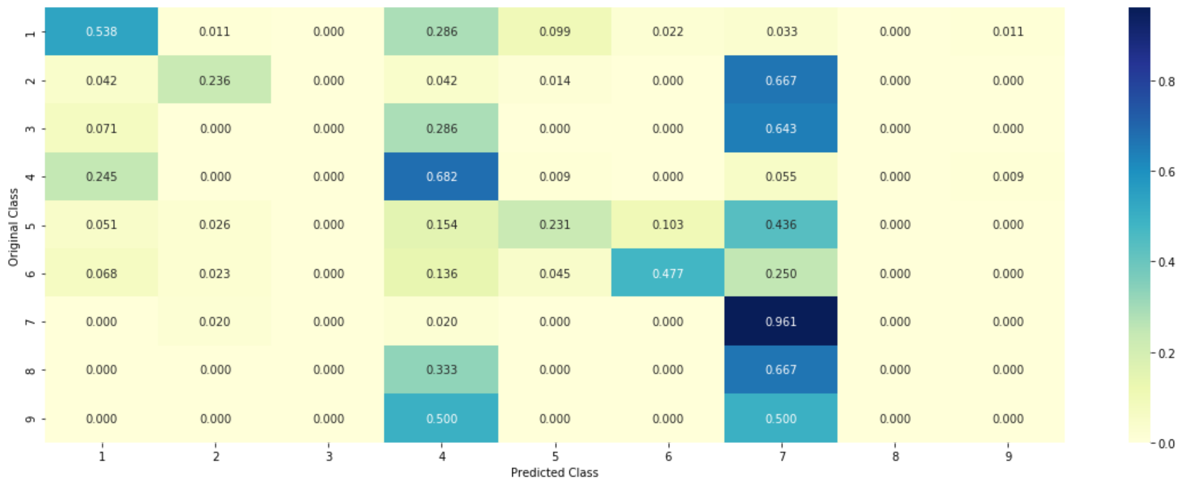
	1	2	3	4	5	6	7	8	9
1	49.000	1.000	0.000	26.000	9.000	2.000	3.000	0.000	1.000
2	3.000	17.000	0.000	3.000	1.000	0.000	48.000	0.000	0.000
3	1.000	0.000	0.000	4.000	0.000	0.000	9.000	0.000	0.000
4	27.000	0.000	0.000	75.000	1.000	0.000	6.000	0.000	1.000
5	2.000	1.000	0.000	6.000	9.000	4.000	17.000	0.000	0.000
6	3.000	1.000	0.000	6.000	2.000	21.000	11.000	0.000	0.000
7	0.000	3.000	0.000	3.000	0.000	0.000	147.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	0.000
9	0.000	0.000	0.000	3.000	0.000	0.000	3.000	0.000	0.000

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



	1	2	3	4	5	6	7	8	9
1	0.576	0.043		0.205	0.409	0.074	0.012		0.500
2	0.035	0.739		0.024	0.045	0.000	0.195		0.000
3	0.012	0.000		0.031	0.000	0.000	0.037		0.000
4	0.318	0.000		0.591	0.045	0.000	0.024		0.500
5	0.024	0.043		0.047	0.409	0.148	0.069		0.000
6	0.035	0.043		0.047	0.091	0.778	0.045		0.000
7	0.000	0.130		0.024	0.000	0.000	0.598		0.000
8	0.000	0.000		0.008	0.000	0.000	0.008		0.000
9	0.000	0.000		0.024	0.000	0.000	0.012		0.000

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



	1	2	3	4	5	6	7	8	9
1	0.538	0.011	0.000	0.286	0.099	0.022	0.033	0.000	0.011
2	0.042	0.236	0.000	0.042	0.014	0.000	0.667	0.000	0.000
3	0.071	0.000	0.000	0.286	0.000	0.000	0.643	0.000	0.000
4	0.245	0.000	0.000	0.682	0.009	0.000	0.055	0.000	0.009
5	0.051	0.026	0.000	0.154	0.231	0.103	0.436	0.000	0.000
6	0.068	0.023	0.000	0.136	0.045	0.477	0.250	0.000	0.000
7	0.000	0.020	0.000	0.020	0.000	0.000	0.961	0.000	0.000
8	0.000	0.000	0.000	0.333	0.000	0.000	0.667	0.000	0.000
9	0.000	0.000	0.000	0.500	0.000	0.000	0.500	0.000	0.000

Process followed:

```
1.importing libraries and Reading dataset
2.EDA on Dataset and Univariate analysis of every feature
3.Encoding catgorical(onehot,response) and text data(tfidf,countvectorizer)
4.Random model for max log-loss
5.Applying models on tfidf text one hot and response encoded catgorical val
ues
6.Taking top 1000 features based on tfidf vectorization
7.Applying Random Forest Model on those features (checking Interpretability
(reasons for prediction))
8.using unigram and bi-gram representation (countvectorizer)
9.Applying Logistic Regression on that text feature
10.Feature Engineering
11.Applying Logistic regression on new  and onehot encoded features
12.Applying Logistic regression on new  and Response encoded features
13.Summary (comparing all model scores)
```

SUMMARY

In [188]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["model","Train Loss","CV Loss","Test Loss","Miss classified%"]
x.add_row(["Random", 0,2.34,2.46,0])
x.add_row(["Naive Bayes(Tfidf)",0.87,1.24,1.27,37.59])
x.add_row(["K Nearest Neighbour(Response)",0.63,1.10,1.08,38.9])
x.add_row(["Logistic Regression(With class balancing)",0.57,1.15,1.09,35.33])
x.add_row(["Logistic Regression(Without class balancing)",0.55,1.16,1.11,36.09])
x.add_row(["Support Vector Mechine(Tfidf)",0.68,1.19,1.15,35.33])
x.add_row(["Random Forest(tfidf)",0.63,1.16,1.11,38.34])
x.add_row(["Random Forest(Response Coding)",0.63,1.16,1.11,38.34])
x.add_row(["Stacking Models(Tfidf)",0.621,1.14,1.11,36.24])
x.add_row(["Maximum Voting Classifier(Tfidf)",0.85,1.15,1.13,37.59])
x.add_row(["Random Forest(tfidf top 1000 +onehot encoded Gene,Variant)",0.85,1.21,1
x.add_row(["Logistic Regression(uni bigram +onehot encoded Gene,Variant)",0.86,1.22
x.add_row(["Logistic Regression(Feature Engineered +onehot encoded Gene,Variant)",0
x.add_row(["Logistic Regression(Feature Engineered +response encoded Gene,Variant)"
print(x)
```

```
+---------------------------------------------------------------------
---+------------+---------+-----------+------------------+
|                                  model
| Train Loss | CV Loss | Test Loss | Miss classified% |
+---------------------------------------------------------------------
---+------------+---------+-----------+------------------+
|                                  Random
|     0      |   2.34  |    2.46   |        0         |
|                            Naive Bayes(Tfidf)
|    0.87    |   1.24  |    1.27   |      37.59       |
|                      K Nearest Neighbour(Response)
|    0.63    |   1.1   |    1.08   |       38.9       |
|                Logistic Regression(With class balancing)
|    0.57    |   1.15  |    1.09   |      35.33       |
|               Logistic Regression(Without class balancing)
|    0.55    |   1.16  |    1.11   |      36.09       |
|                      Support Vector Mechine(Tfidf)
|    0.68    |   1.19  |    1.15   |      35.33       |
|                           Random Forest(tfidf)
|    0.63    |   1.16  |    1.11   |      38.34       |
|                      Random Forest(Response Coding)
|    0.63    |   1.16  |    1.11   |      38.34       |
|                          Stacking Models(Tfidf)
|   0.621    |   1.14  |    1.11   |      36.24       |
|                     Maximum Voting Classifier(Tfidf)
|    0.85    |   1.15  |    1.13   |      37.59       |
|       Random Forest(tfidf top 1000 +onehot encoded Gene,Variant)
|    0.85    |   1.21  |    1.19   |      42.85       |
|      Logistic Regression(uni bigram +onehot encoded Gene,Variant)
|    0.86    |   1.22  |    1.21   |      40.22       |
|  Logistic Regression(Feature Engineered +onehot encoded Gene,Varian
t)  |    0.46    |   0.98  |   0.982   |      32.51       |
| Logistic Regression(Feature Engineered +response encoded Gene,Varian
t) |    0.99    |   1.12  |    1.17   |      40.22       |
+---------------------------------------------------------------------
---+------------+---------+-----------+------------------+
```

Observations:-

```
1.Feature Engineering improves the performance of the models to a greater e
xtent
2.Logistic Regression tends to perform well in case of high dimensionality
3.One hot encoding is better for Logistic Regression where as Random forest
works well with Response encode-d features
```

In [ ]: