

Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>
2. <https://www.youtube.com/watch?v=UwbuW7oK8rk>
3. <https://www.youtube.com/watch?v=qxXRKVompl8>

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/data>
- We have two data files: one contains 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 have a common column called ID
- Data file's information:
 - training_variants (ID , Gene, Variations, Class)
 - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning 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 probabilities => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [0]:

In [0]:

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [0]:

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:

```
link= 'https://drive.google.com/open?id=1gP70Fh4rhyfhQQ9PPVhekW2e-5I9Vu0t'
```

In [5]:

```
fluff, id = link.split('=')
print (id)
```

```
1gP70Fh4rhyfhQQ9PPVhekW2e-5I9Vu0t
```

In [0]:

```
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('training_variants')
```

In [0]:

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:

```
link= 'https://drive.google.com/open?id=1ttR9xZC5lct5CocLP7Ft0lpt7E0-qJkf'
```

In [9]:

```
fluff, id = link.split('=')
print (id)
```

```
1ttR9xZC5lct5CocLP7Ft0lpt7E0-qJkf
```

In [0]:

```
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('training_text')
```

In [11]:

```
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[11]:

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

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

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

3.1.2. Reading Text Data

In [12]:

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

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

Out[12]:

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

3.1.3. Preprocessing of text

In [13]:

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

Out[13]:

```
True
```

In [0]:

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

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

```

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

data_text[column][index] = string

```

In [15]:

```

#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")

```

Time took for preprocessing the text : 272.647463 seconds

In [16]:

```

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

```

Out[16]:

	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 [17]:

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

Out[17]:

	ID	Gene	Variation	Class	TEXT
--	----	------	-----------	-------	------

In [0]:

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

In [19]:

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

Out[19]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	null

3.1.4. Test, Train and Cross Validation Split

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

In [0]:

```

y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
# split the data into test and train by maintaining same distribution of output variable 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output variable 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)

```

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

In [21]:

```

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

```

```

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

```

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

In [22]:

```

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

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

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

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

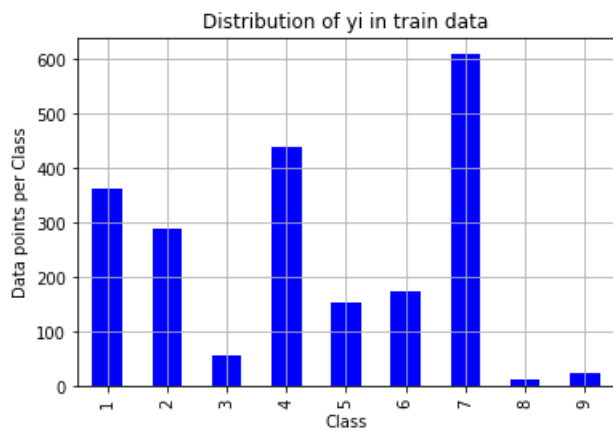
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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

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

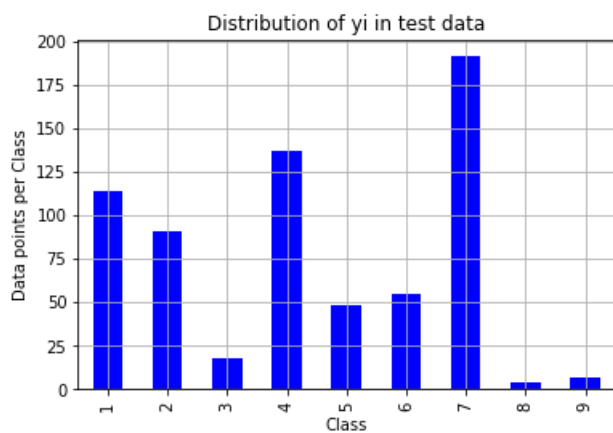
```

```
plt.show()

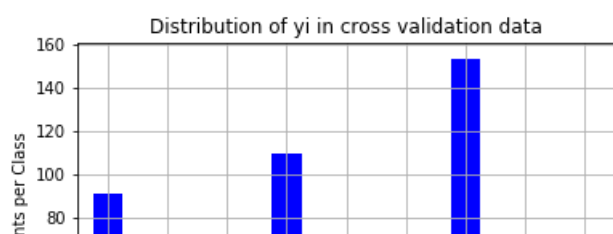
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':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

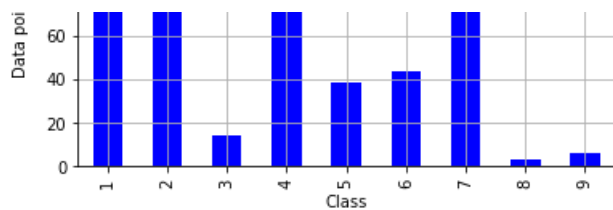


Number of data points in class 7 : 609 (28.672 %)
 Number of data points in class 4 : 439 (20.669 %)
 Number of data points in class 1 : 363 (17.09 %)
 Number of data points in class 2 : 289 (13.606 %)
 Number of data points in class 6 : 176 (8.286 %)
 Number of data points in class 5 : 155 (7.298 %)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



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





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

3.2 Prediction using a 'Random' Model

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

In [0]:

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

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

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

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

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

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

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

    # representing B in heatmap format
```

```

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

```

In [24]:

```

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

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

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

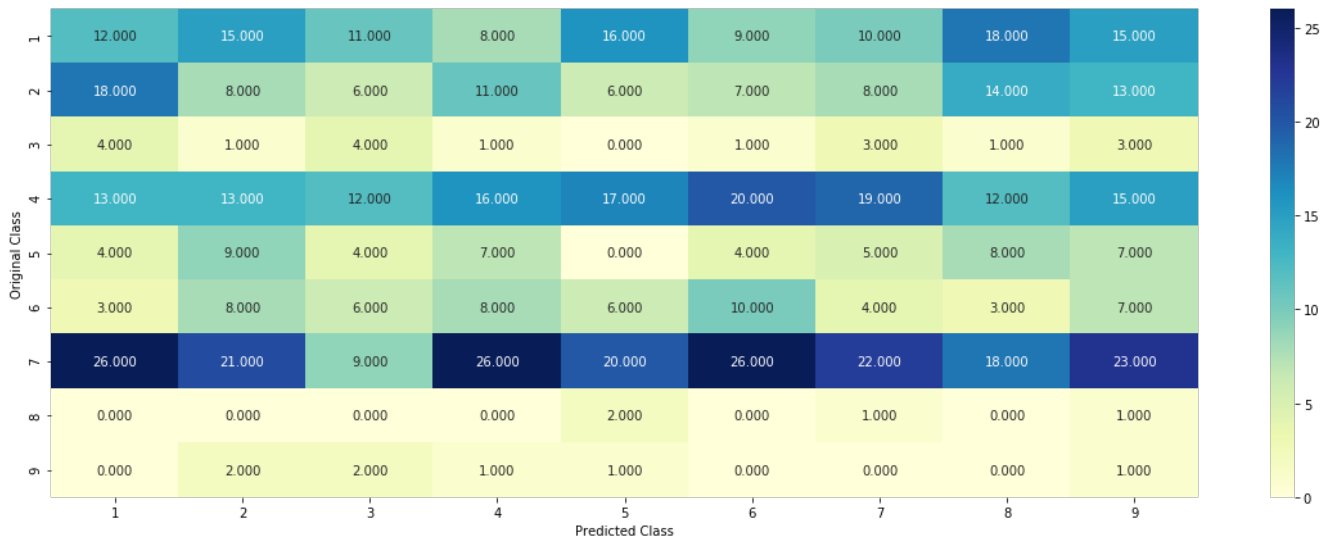
predicted_y = np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

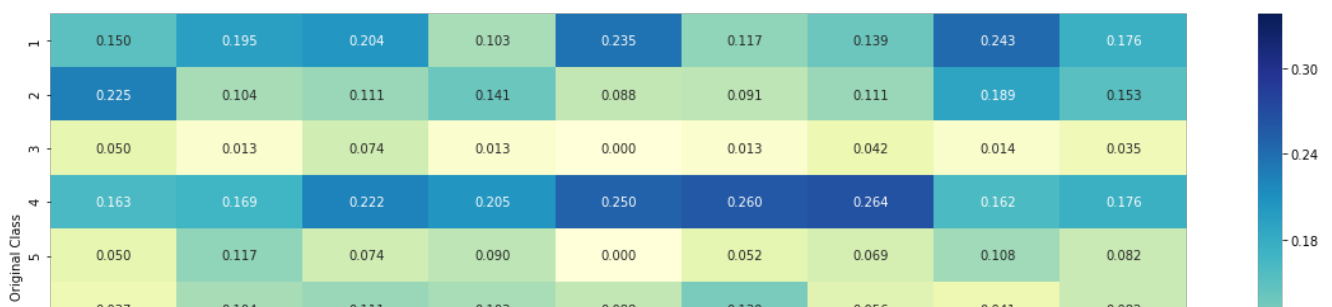
Log loss on Cross Validation Data using Random Model 2.521919391680851

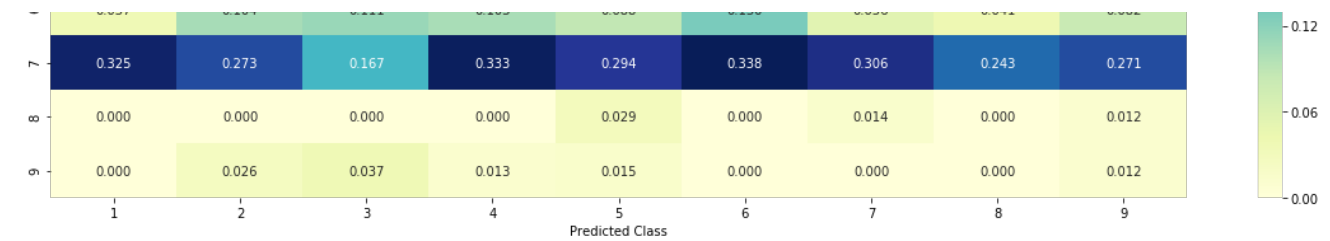
Log loss on Test Data using Random Model 2.4576412291665304

----- Confusion matrix -----

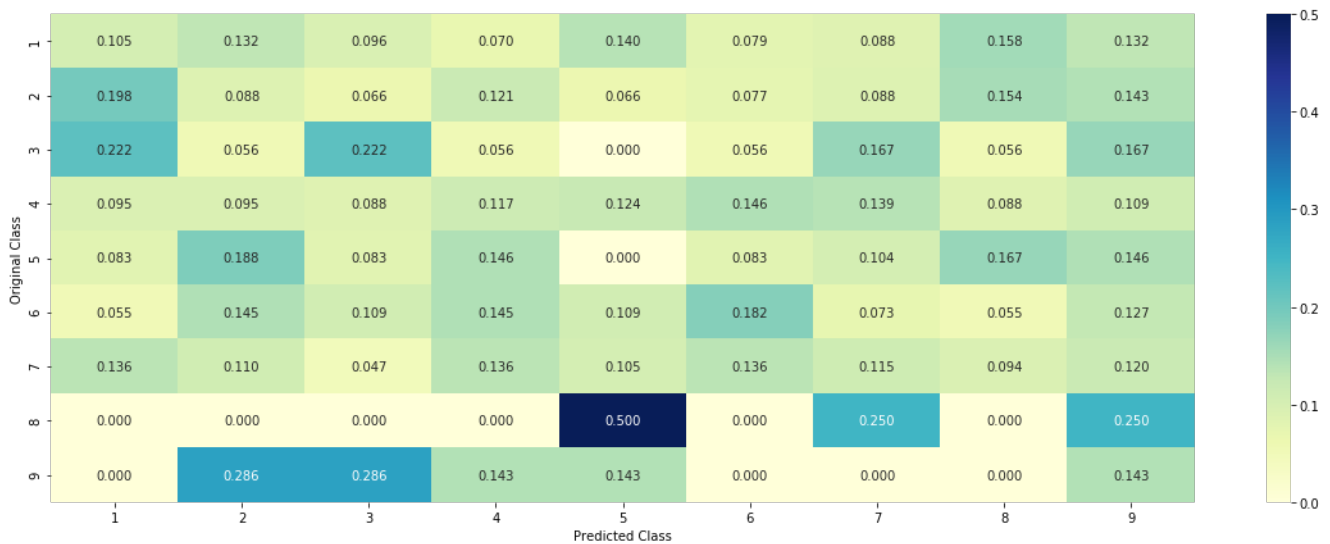


----- Precision matrix (Column Sum=1) -----





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



3.3 Univariate Analysis

In [0]:

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

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

```

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

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

# denominator will contain the number of time that particular feature occurred in whole data
for i, denominator in value_count.items():
    # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
    # 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 particular feature occurred in whole data
        vec.append((cls_cnt.shape[0] + alpha*10) / (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.06818181818181817, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181817, 0.06818181818181817, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.07878787878787878, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    # 'BRAF': [0.06666666666666666, 0.17999999999999999, 0.07333333333333334, 0.07333333333333334, 0.09333333333333338, 0.08000000000000002, 0.29999999999999999, 0.06666666666666666, 0.06666666666666666],
    # ...
    # }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9 1/9 1/9 1/9 1/9 1/9 1/9 1/9 1/9] to gv_fea

```

```
# 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 calculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- $(\text{numerator} + 10 \cdot \alpha) / (\text{denominator} + 90 \cdot \alpha)$

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [26]:

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

```
Number of Unique Genes : 239
BRCA1      169
TP53       107
EGFR        89
PTEN        81
BRCA2       76
KIT          70
BRAF        58
ALK          46
PDGFRA      39
ERBB2       39
Name: Gene, dtype: int64
```

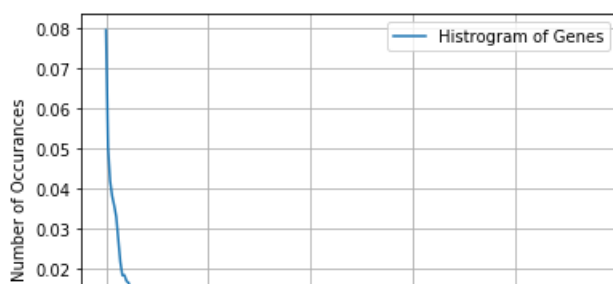
In [27]:

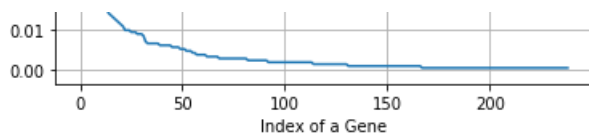
```
print("Ans: There are", unique_genes.shape[0], "different categories of genes in the train data, and they are distributed as follows",)
```

Ans: There are 239 different categories of genes in the train data, and they are distributed as follows

In [28]:

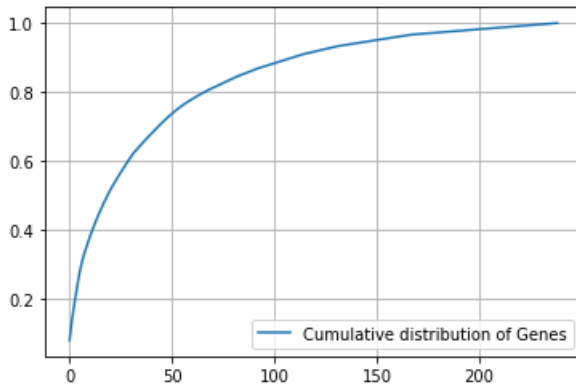
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```





In [29]:

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



Q3. How to featurize this Gene feature ?

Ans. there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [0]:

```
#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 [31]:

```
print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

Using Tfidf Vectorizer

In [0]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
```

```
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [33]:

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

Out[33]:

```
793      ERBB4
1572      ALK
905      PDGFRA
1762      IDH1
2194      PTEN
Name: Gene, dtype: object
```

In [34]:

```
gene_vectorizer.get_feature_names()
```

Out[34]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl2',
 'atm',
 'atr',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
```

'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfil',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'gata3',
'gli1',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnfla',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2c',
'kmt2d',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',

'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
'ncor1',
'nfl',
'nf2',
'nfe2l2',
'nfkb1a',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shoc2'.

```

'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stat3',
'stk11',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'whsc1',
'xpo1',
'xrc2',
'yap1']

```

In [35]:

```

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

```

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

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 [36]:

```

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

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

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

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

```

```

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

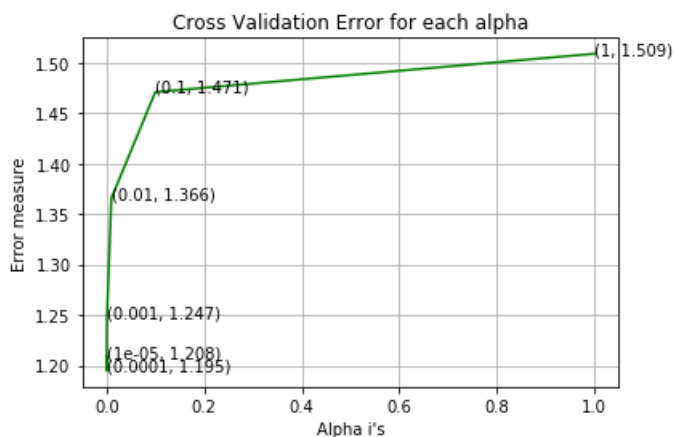
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```

For values of alpha = 1e-05 The log loss is: 1.2082456766821463
 For values of alpha = 0.0001 The log loss is: 1.1946670411978328
 For values of alpha = 0.001 The log loss is: 1.2472342198772528
 For values of alpha = 0.01 The log loss is: 1.3655907759998371
 For values of alpha = 0.1 The log loss is: 1.4707121497035074
 For values of alpha = 1 The log loss is: 1.5087073831885303



For values of best alpha = 0.0001 The train log loss is: 1.0127147944076107
 For values of best alpha = 0.0001 The cross validation log loss is: 1.1946670411978328
 For values of best alpha = 0.0001 The test log loss is: 1.14452987307423

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 [37]:

```

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

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

```

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

Q6. How many data points in Test and CV datasets are covered by the 239 genes in train dataset?

Ans

1. In test data 646 out of 665 : 97.14285714285714

2. In cross validation data 518 out of 532 : 97.36842105263158

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [38]:

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

Number of Unique Variations : 1927

Deletion 52

Amplification 50

Truncating_Mutations 48

Fusions 26

Overexpression 4

G12V 4

E17K 3

I31M 2

G13D 2

T167A 2

Name: Variation, dtype: int64

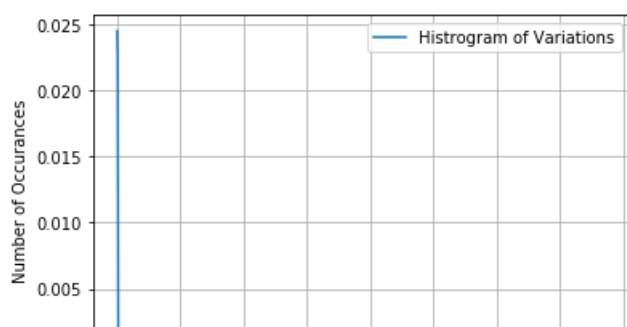
In [39]:

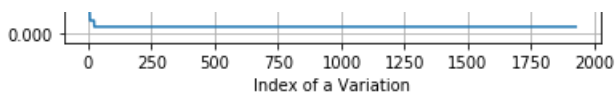
```
print("Ans: There are", unique_variations.shape[0] , "different categories of variations in the train data, and they are distributed as follows",)
```

Ans: There are 1927 different categories of variations in the train data, and they are distributed as follows

In [40]:

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

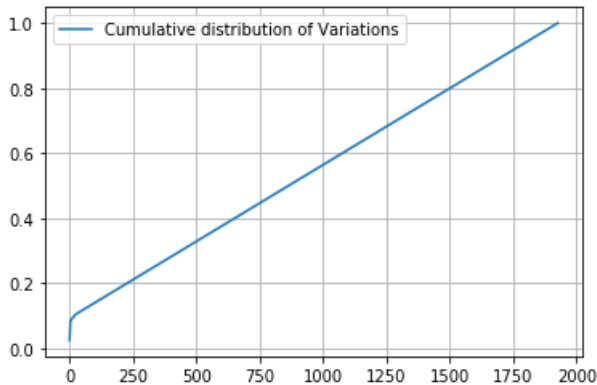




In [41]:

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

```
[0.02448211 0.0480226 0.07062147 ... 0.99905838 0.99952919 1. ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [0]:

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

In [43]:

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

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

In [0]:

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

In [45]:

```
print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 1959)

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

Let's build a model just like the earlier!

In [46]:

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

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

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

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

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

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

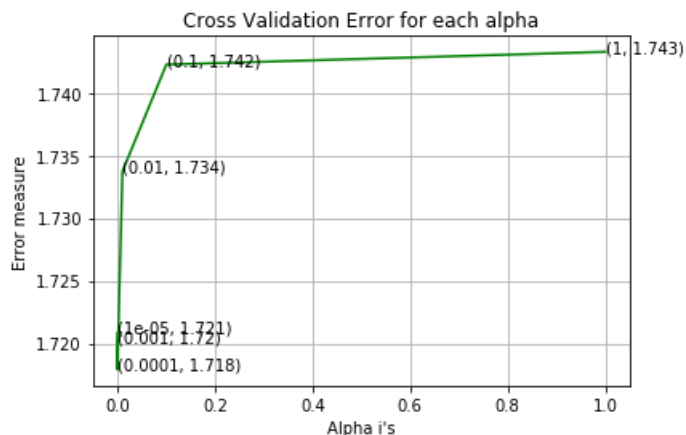
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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

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

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

For values of alpha = 1e-05 The log loss is: 1.7208392835367856
 For values of alpha = 0.0001 The log loss is: 1.7179003292506485
 For values of alpha = 0.001 The log loss is: 1.720099074143756
 For values of alpha = 0.01 The log loss is: 1.7337372678227214
 For values of alpha = 0.1 The log loss is: 1.7422574442994867
 For values of alpha = 1 The log loss is: 1.7432743435419358



For values of best alpha = 0.0001 The train log loss is: 0.7629995506005386
 For values of best alpha = 0.0001 The cross validation log loss is: 1.7179003292506485
 For values of best alpha = 0.0001 The test log loss is: 1.6932538439916989

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 [47]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\1. In test data',test_coverage, 'out of',test_df.shape[0], ":", (test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":", (cv_coverage/cv_df.shape[0])*100)
```

Q12. How many data points are covered by total 1927 genes in test and cross validation data sets?

Ans

1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 54 out of 532 : 10.150375939849624

3.2.3 Univariate Analysis on Text Feature

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

In [0]:

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

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
```

```

        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary

```

In [0]:

```

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

```

In [50]:

```

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

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

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

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

```

Total number of unique words in train data : 1000

In [0]:

```

dict_list = []
# dict_list=[] contains 9 dictionaries 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 build 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 [0]:

```

#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 [0]:

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

In [0]:

```
# 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 [0]:

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

In [56]:

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

```
Counter({252.13121764701262: 1, 182.05898875749324: 1, 133.6575178538359: 1, 133.17699596481765: 1,
129.83974183480487: 1, 119.96322061043313: 1, 119.63664950373052: 1, 113.93647963098371: 1,
113.22997572805873: 1, 108.18518610812625: 1, 105.91154165734054: 1, 90.06896827712725: 1,
89.64475098031308: 1, 88.61938405162934: 1, 81.45071694296391: 1, 81.36199553504551: 1,
79.8461347513574: 1, 79.26769386364384: 1, 78.74765687518907: 1, 77.39969899209521: 1,
77.24970126067815: 1, 74.39223287238022: 1, 69.64336822919198: 1, 69.20300491834465: 1,
68.78924334567776: 1, 67.87915015838448: 1, 66.2274164161208: 1, 65.23385731798344: 1,
64.24379449728924: 1, 64.03882626626006: 1, 63.776716480762126: 1, 63.62247008651857: 1,
62.22714720192915: 1, 59.357890144640514: 1, 59.29476188673443: 1, 58.18031215283741: 1,
56.60885276619782: 1, 56.30463269285229: 1, 54.0240821044073: 1, 53.06225256686257: 1,
51.552233525851165: 1, 50.5809327906934: 1, 49.93259703658693: 1, 49.60288245700827: 1,
49.13835813865119: 1, 48.51515738920927: 1, 48.24090023926921: 1, 46.9126635498478: 1,
46.467383969670834: 1, 46.17130021696102: 1, 45.636668240414615: 1, 44.28868310181699: 1,
43.968399204921006: 1, 43.9676643452848: 1, 43.47703182074203: 1, 43.18128980266401: 1,
43.14794486470538: 1, 42.68244229876238: 1, 42.022613413204596: 1, 41.879423152054464: 1,
41.75865074920074: 1, 41.14290840933409: 1, 40.82032661371804: 1, 40.031515681847985: 1,
39.87373811302046: 1, 39.79727767040212: 1, 39.27876431129272: 1, 39.209639507204386: 1,
39.113856010332555: 1, 38.93908330552918: 1, 38.889574291357704: 1, 38.422767176496265: 1,
38.29235737826823: 1, 37.7643126772316: 1, 37.21929460063124: 1, 37.18715891417655: 1,
37.14738781176538: 1, 37.11594643447873: 1, 36.82823716607767: 1, 36.42621138800891: 1,
36.128460489208585: 1, 36.070481624960706: 1, 35.9466009556808: 1, 35.67563689491884: 1,
35.268607550749174: 1, 35.237463772359945: 1, 34.754966117459745: 1, 34.12672664341926: 1,
34.05502620377925: 1, 33.59564761544674: 1, 33.59083968742792: 1, 33.27158161275426: 1,
33.235726275763156: 1, 32.98867117409903: 1, 32.875807682166055: 1, 32.79498059410589: 1,
32.781696641172005: 1, 32.75148004361944: 1, 32.46577221230045: 1, 32.399250622794966: 1,
32.174151368885504: 1, 32.036739547526395: 1, 31.986824653143596: 1, 31.972266715943658: 1,
31.92235841252026: 1, 31.565806705504105: 1, 31.32525364565513: 1, 30.985550346554266: 1,
30.922437444051237: 1, 30.900875324215342: 1, 30.789013156347416: 1, 30.78749124188596: 1,
30.74795477100428: 1, 30.701419513338475: 1, 30.643267918549917: 1, 30.534104655907367: 1,
30.4161688348333: 1, 30.28515137591406: 1, 30.027545059844584: 1, 29.956996599878202: 1,
29.84052830459993: 1, 29.73665220493253: 1, 29.646596358992277: 1, 29.394501437693044: 1,
28.915956203646665: 1, 28.755915197822134: 1, 28.71190097190299: 1, 28.633949446026673: 1,
28.537179245864134: 1, 28.291962601581233: 1, 28.195994546368453: 1, 28.141175496519587: 1,
28.134123608135077: 1, 27.91522448298007: 1, 27.841941949027106: 1, 27.722172328302662: 1,
27.506867838953294: 1, 27.351639502316335: 1, 27.324789630713305: 1, 27.139282677373266: 1,
27.062081724587816: 1, 26.657867195434026: 1, 26.53704965992687: 1, 26.50566195829096: 1,
26.28366510847649: 1, 26.230705841987216: 1, 26.054309234115216: 1, 25.87763348679646: 1,
```

25.816229157172117: 1, 25.772262332945996: 1, 25.647348190791256: 1, 25.575862919384203: 1, 25.513999101519058: 1, 25.511101082465792: 1, 25.375619481655903: 1, 25.300920401087126: 1, 25.29088432375386: 1, 25.24205558874857: 1, 25.204685136544708: 1, 25.01624681761713: 1, 24.952067192483476: 1, 24.91192353016153: 1, 24.852414497239927: 1, 24.832630296941932: 1, 24.70713080728063: 1, 24.503408229410127: 1, 24.43136525550069: 1, 24.333450390296232: 1, 24.327559077089514: 1, 24.29865949726832: 1, 24.202336610560266: 1, 24.1540249102929: 1, 23.97411831533632: 1, 23.919429443245797: 1, 23.91006965062818: 1, 23.872098439800325: 1, 23.855664873979265: 1, 23.78763122128519: 1, 23.77333106215535: 1, 23.769432810849906: 1, 23.73927343474098: 1, 23.685521786995967: 1, 23.66337695667023: 1, 23.659823070813367: 1, 23.490203573268712: 1, 23.37191358967531: 1, 23.2898212347532: 1, 23.230701630639242: 1, 23.187181558147334: 1, 22.90949326028604: 1, 22.80569354069065: 1, 22.791388574526568: 1, 22.758232028990147: 1, 22.678543652819503: 1, 22.65923276513089: 1, 22.58381991971626: 1, 22.569938531430918: 1, 22.556506471481367: 1, 22.47446783927251: 1, 22.467703278733786: 1, 22.46581261881872: 1, 22.417342939175803: 1, 22.399899623333102: 1, 22.291231038579724: 1, 22.284144452695227: 1, 22.267205178498923: 1, 22.22867353020079: 1, 22.043410762352263: 1, 21.923436470936426: 1, 21.83607831180739: 1, 21.80217828147278: 1, 21.801983977891428: 1, 21.793199453580552: 1, 21.759648437381767: 1, 21.756175793921592: 1, 21.67989744882074: 1, 21.57976256489256: 1, 21.49092964263283: 1, 21.484541539126624: 1, 21.40863905577662: 1, 21.40279342172206: 1, 21.327748591907348: 1, 21.3170780241819: 1, 21.295218868459955: 1, 21.28898538766919: 1, 21.223500888007166: 1, 21.19931960127197: 1, 21.179905609893968: 1, 21.112431456107323: 1, 21.095302705569978: 1, 21.090333816302362: 1, 20.914818285458242: 1, 20.89581111917903: 1, 20.8790033032522: 1, 20.863827404915792: 1, 20.847128218619144: 1, 20.809802470224504: 1, 20.786848558311927: 1, 20.783697773941867: 1, 20.776671840919008: 1, 20.42300784267042: 1, 20.363959570416487: 1, 20.166006853063045: 1, 20.08632052136064: 1, 20.07477472006022: 1, 20.026565298349748: 1, 20.00261687568856: 1, 19.97990902172252: 1, 19.963663511683336: 1, 19.93533694349359: 1, 19.89620183235277: 1, 19.854976281380914: 1, 19.845006408118348: 1, 19.796072537657835: 1, 19.796071888505377: 1, 19.741549477212168: 1, 19.724079845710516: 1, 19.705108469960443: 1, 19.687548018815814: 1, 19.637289901078585: 1, 19.58815281608833: 1, 19.557716671169374: 1, 19.555003949493052: 1, 19.508760808141155: 1, 19.488782278055943: 1, 19.41403646554568: 1, 19.412027125263727: 1, 19.397819730153564: 1, 19.363187550959925: 1, 19.362266357227096: 1, 19.307744842020877: 1, 19.273251608812956: 1, 19.204874942005617: 1, 19.200157647103413: 1, 19.092258052302867: 1, 19.068018349390925: 1, 19.060787648870516: 1, 18.993749013965164: 1, 18.98683107607677: 1, 18.943082702938522: 1, 18.895289034256745: 1, 18.893003499086333: 1, 18.815691510832306: 1, 18.765144325991518: 1, 18.734372854236288: 1, 18.64847785194308: 1, 18.62905756495574: 1, 18.61663095000656: 1, 18.59800317221214: 1, 18.594257071164694: 1, 18.57861198115784: 1, 18.567136002062853: 1, 18.553432689397876: 1, 18.394502821877524: 1, 18.381154555468992: 1, 18.371487687695552: 1, 18.364985126897114: 1, 18.33422798823729: 1, 18.289004511654674: 1, 18.273510347101773: 1, 18.26175786087691: 1, 18.238090622025616: 1, 18.135755330310477: 1, 18.097816684933015: 1, 18.074004343996197: 1, 18.00450362313136: 1, 17.958192390209273: 1, 17.91252399535572: 1, 17.876041628101625: 1, 17.85997061124629: 1, 17.815325928457227: 1, 17.78211940843222: 1, 17.726022381197723: 1, 17.713646086999027: 1, 17.67135649604409: 1, 17.654933712969136: 1, 17.632341923990694: 1, 17.590830841929407: 1, 17.52765501968541: 1, 17.490098279445142: 1, 17.45271903252242: 1, 17.44896422168781: 1, 17.363201193863382: 1, 17.360182241995506: 1, 17.35009535236699: 1, 17.33680310878712: 1, 17.316641502471406: 1, 17.314492989040076: 1, 17.29821340511997: 1, 17.276873680052237: 1, 17.266289677686945: 1, 17.25682079398894: 1, 17.25299090599686: 1, 17.22146038661147: 1, 17.204316035652624: 1, 17.200247332149004: 1, 17.180346308482072: 1, 17.134600097718906: 1, 17.088499768037067: 1, 17.085937171415654: 1, 17.073732763301873: 1, 17.06388286362795: 1, 16.94834955491869: 1, 16.942000759446454: 1, 16.92731634754443: 1, 16.924018220845785: 1, 16.916834898270437: 1, 16.854291518076774: 1, 16.836108609329838: 1, 16.832301142358673: 1, 16.81589504635042: 1, 16.805937396615633: 1, 16.781243154771424: 1, 16.765209996760667: 1, 16.742101703047467: 1, 16.700787492491248: 1, 16.690529614255563: 1, 16.669599596270874: 1, 16.5864237261264: 1, 16.573112259225546: 1, 16.554417246422172: 1, 16.5494910509413: 1, 16.53815145655791: 1, 16.51926357230315: 1, 16.47797417793415: 1, 16.45712105885847: 1, 16.382517499188367: 1, 16.35390337000035: 1, 16.333071316779726: 1, 16.322989674792307: 1, 16.307286421483642: 1, 16.225135765089533: 1, 16.218118537547042: 1, 16.214177536098035: 1, 16.198822609692773: 1, 16.193840241528775: 1, 16.17244723961139: 1, 16.16677926787629: 1, 16.126691126078395: 1, 16.122255172483676: 1, 16.0761660255053: 1, 16.033402969463648: 1, 16.028756107655386: 1, 15.98296354959938: 1, 15.96026103681588: 1, 15.960167284425571: 1, 15.866074885574688: 1, 15.845869257431037: 1, 15.819454465120021: 1, 15.80180444803315: 1, 15.77583475588324: 1, 15.74754816398125: 1, 15.70525225155463: 1, 15.691040970722517: 1, 15.689906569271567: 1, 15.636232507233549: 1, 15.548639343678522: 1, 15.548111173836503: 1, 15.547331707738094: 1, 15.476867279482194: 1, 15.411017349086377: 1, 15.408869447817716: 1, 15.405043087820651: 1, 15.391838323638325: 1, 15.322141564279747: 1, 15.314848040484387: 1, 15.314529776993632: 1, 15.294479095519486: 1, 15.250803614105031: 1, 15.185546122193404: 1, 15.172542915650396: 1, 15.16945239532753: 1, 15.144726813840427: 1, 15.09298091524794: 1, 15.065853136179348: 1, 15.05192883009249: 1, 15.028509600663803: 1, 15.02675195668593: 1, 15.019711111084485: 1, 15.01549620575306: 1, 15.015433423678337: 1, 15.011421582949719: 1, 14.96304433665679: 1, 14.91759042864386: 1, 14.906201277140378: 1, 14.889224425426056: 1, 14.875119452414527: 1, 14.82054767298601: 1, 14.809403763723735: 1, 14.809148046959764: 1, 14.752065765639564: 1, 14.721176968349825: 1, 14.695040199664044: 1, 14.68785886343601: 1, 14.663242517588772: 1, 14.612705584008046: 1, 14.590493041500748: 1, 14.5888000051284: 1, 14.58490541022147: 1, 14.57438599244783: 1, 14.572588790852782: 1, 14.545970881534657: 1, 14.527584865836225: 1, 14.486768011309795: 1, 14.484947781109529: 1, 14.460124281594329: 1, 14.441732532920563: 1, 14.412927729468311: 1, 14.380449641415817: 1, 14.379868663858828: 1, 14.37579943023597: 1, 14.372044489722416: 1, 14.360566352427098: 1, 14.347810484179147: 1, 14.342383026461407: 1, 14.321059424504723: 1,

14.315747724680872: 1, 14.268151106615285: 1, 14.267001441739763: 1, 14.241272373465057: 1,
14.222129352775346: 1, 14.21207911716977: 1, 14.107119362799: 1, 14.104200183892836: 1,
14.06741449122551: 1, 14.049194189734907: 1, 14.048724127994477: 1, 14.02954990948554: 1,
13.998553005963203: 1, 13.949235631074473: 1, 13.898529024873032: 1, 13.88220901473867: 1,
13.857618327715334: 1, 13.848481178060926: 1, 13.819427316821944: 1, 13.814213640646809: 1,
13.782674667620938: 1, 13.717020554514164: 1, 13.705982687912096: 1, 13.702168131941978: 1,
13.677826928828537: 1, 13.674505052029227: 1, 13.671998235933584: 1, 13.64662698816715: 1,
13.59021986974654: 1, 13.575072256670673: 1, 13.57343497895679: 1, 13.544935994496228: 1,
13.502821079337613: 1, 13.479842524574856: 1, 13.45664420939611: 1, 13.408599754215857: 1,
13.39262512514841: 1, 13.389684219547386: 1, 13.322324503922497: 1, 13.308373704928824: 1,
13.275248985334771: 1, 13.271363923897711: 1, 13.2596341965514: 1, 13.216848329746574: 1,
13.20703132354516: 1, 13.175576328328072: 1, 13.16804797400784: 1, 13.159773797754431: 1,
13.138000304565264: 1, 13.068418169028725: 1, 13.052800070769827: 1, 12.992232297514754: 1,
12.982032439999898: 1, 12.956270305287157: 1, 12.944985558648266: 1, 12.934609129787441: 1,
12.930883463313886: 1, 12.924578122531804: 1, 12.911418236400278: 1, 12.908930153788852: 1,
12.885274027797902: 1, 12.862301326749503: 1, 12.832207660114745: 1, 12.830622999218486: 1,
12.815139688071731: 1, 12.80333897373077: 1, 12.77769581421194: 1, 12.753245100201044: 1,
12.74875358909606: 1, 12.738868703046057: 1, 12.716453901447066: 1, 12.713326876518156: 1,
12.706290907398047: 1, 12.697714548943408: 1, 12.689440897345811: 1, 12.681560777961293: 1,
12.674147931199366: 1, 12.664107097527825: 1, 12.658269901219303: 1, 12.61722419754447: 1,
12.61465238282782: 1, 12.605572124556671: 1, 12.57656443958533: 1, 12.548824763281825: 1,
12.526371222418687: 1, 12.504569721284298: 1, 12.47733609339469: 1, 12.456560393131733: 1,
12.441553547808601: 1, 12.430786594312087: 1, 12.424291922314818: 1, 12.42002135319754: 1,
12.352124918984552: 1, 12.341251663797113: 1, 12.339965676205871: 1, 12.335904554787662: 1,
12.332258990526775: 1, 12.31954117787357: 1, 12.297875546007038: 1, 12.292525240714196: 1,
12.269003173347961: 1, 12.259722777558055: 1, 12.259020943447343: 1, 12.256285106365777: 1,
12.217348790080338: 1, 12.21082622800727: 1, 12.208850164825135: 1, 12.205995925976072: 1,
12.204394453705728: 1, 12.203117703205557: 1, 12.195941177909: 1, 12.195522735262314: 1,
12.171424893850984: 1, 12.170332550124325: 1, 12.112500881432082: 1, 12.028271902845761: 1,
12.023166572238255: 1, 11.987024604830625: 1, 11.968822087003469: 1, 11.957571467525963: 1,
11.931485906778388: 1, 11.931110683027686: 1, 11.911501615655142: 1, 11.888711168453046: 1,
11.884096544939801: 1, 11.878732180458334: 1, 11.878102960594871: 1, 11.829611508439033: 1,
11.82520388275743: 1, 11.797414219840725: 1, 11.791388290555847: 1, 11.78264630976602: 1,
11.779502999062263: 1, 11.77073250952162: 1, 11.733277668339905: 1, 11.724673905639847: 1,
11.709558199890996: 1, 11.680998745774446: 1, 11.668938052805963: 1, 11.653943995279358: 1,
11.63005733128032: 1, 11.617236098341548: 1, 11.612557903289117: 1, 11.59667015819938: 1,
11.510070400095785: 1, 11.457253362793473: 1, 11.447005177497815: 1, 11.440069824865724: 1,
11.438281732013145: 1, 11.423972892220895: 1, 11.405280729479609: 1, 11.376650165618045: 1,
11.356635670047455: 1, 11.34134959334716: 1, 11.331756758228728: 1, 11.305138400719045: 1,
11.303438550692045: 1, 11.301793767079392: 1, 11.30158847725428: 1, 11.277714457212873: 1,
11.252829330523662: 1, 11.218982089466051: 1, 11.194849855184: 1, 11.194620191485178: 1,
11.175476081191878: 1, 11.165828593763386: 1, 11.153886058570537: 1, 11.144223181144916: 1,
11.128725272387411: 1, 11.101414092457885: 1, 11.094758116789206: 1, 11.071836224962862: 1,
11.06135532909249: 1, 11.045532599339708: 1, 11.027262961432509: 1, 11.024107493053387: 1,
11.022781950462099: 1, 11.022094392893242: 1, 11.019384965599322: 1, 11.001759782903257: 1,
10.992344756336548: 1, 10.989572882770394: 1, 10.959705817002623: 1, 10.95461203930274: 1,
10.939234815251208: 1, 10.937360409097787: 1, 10.925804716953678: 1, 10.923351335242106: 1,
10.918629589344313: 1, 10.909921753546776: 1, 10.909307461701994: 1, 10.897223545498965: 1,
10.883401065246687: 1, 10.87340337842494: 1, 10.859461578788258: 1, 10.855114975953718: 1,
10.852557712601987: 1, 10.814307395158421: 1, 10.792752585858711: 1, 10.77919696335226: 1,
10.775933031244227: 1, 10.768349734831142: 1, 10.755719234040548: 1, 10.721622504337795: 1,
10.720561190890914: 1, 10.718937047923532: 1, 10.714504312696283: 1, 10.685922153573362: 1,
10.683970829572536: 1, 10.681945953862423: 1, 10.661873343554937: 1, 10.652867416001692: 1,
10.645397046982563: 1, 10.63075017365233: 1, 10.629568002028934: 1, 10.616687101099508: 1,
10.615661169238464: 1, 10.612425391922697: 1, 10.602941708102017: 1, 10.590510901009035: 1,
10.565889273678868: 1, 10.546545219084107: 1, 10.509899765804198: 1, 10.49873769163899: 1,
10.493354142849734: 1, 10.488572698258949: 1, 10.488050467741848: 1, 10.469629982039761: 1,
10.468203394437223: 1, 10.451186261032936: 1, 10.434333819278855: 1, 10.43314462335926: 1,
10.417897569470385: 1, 10.408052294381783: 1, 10.398570401324577: 1, 10.36803520058316: 1,
10.342059202456126: 1, 10.331196434144877: 1, 10.330367538651801: 1, 10.32701943440178: 1,
10.316542772933126: 1, 10.305613994858014: 1, 10.305384006066124: 1, 10.29643553691002: 1,
10.293890102405516: 1, 10.289799406096085: 1, 10.281684326404067: 1, 10.272885197393297: 1,
10.269554059858365: 1, 10.186948266112244: 1, 10.186180107344411: 1, 10.175449329453986: 1,
10.145992714549974: 1, 10.140937344998552: 1, 10.125003921129284: 1, 10.08873444407726: 1,
10.067873446898139: 1, 10.063680981879594: 1, 10.044407610039402: 1, 10.037306653098815: 1,
10.022693357952571: 1, 10.015507641937523: 1, 10.014463235693654: 1, 10.008258683409078: 1,
10.003641165599856: 1, 9.988661185807924: 1, 9.939871422390397: 1, 9.929835885625597: 1, 9.89728577
7473117: 1, 9.893446870555229: 1, 9.865979885945853: 1, 9.857314807252045: 1, 9.84307513138308: 1,
9.826092142730392: 1, 9.8079644363349: 1, 9.800407535245979: 1, 9.785377091352077: 1,
9.782966698534784: 1, 9.767704709305669: 1, 9.757029884989008: 1, 9.75682668664531: 1,
9.755974334367163: 1, 9.737108656298355: 1, 9.73506076896274: 1, 9.718337969803079: 1,
9.697489176701481: 1, 9.688412591693291: 1, 9.676566984224358: 1, 9.671181018502672: 1,
9.648361639818246: 1, 9.6309118608869: 1, 9.620384496718748: 1, 9.613837882905447: 1,
9.60746614947801: 1, 9.597662254734487: 1, 9.594915064562208: 1, 9.593010693671882: 1,
9.586415535466093: 1, 9.566263188436755: 1, 9.562348871191597: 1, 9.550228311624696: 1,
9.5501511981325: 1, 9.534846428401057: 1, 9.531367270273602: 1, 9.51284201319075: 1,
9.495107183831562: 1, 9.494791815739681: 1, 9.478859685510288: 1, 9.45612239621084: 1.

9.454224924773303: 1, 9.452119897016887: 1, 9.447237257865233: 1, 9.41803320049064: 1, 9.415282426841216: 1, 9.414517311687117: 1, 9.409630303627143: 1, 9.39331832801071: 1, 9.385671587641461: 1, 9.379397731433023: 1, 9.372985419632847: 1, 9.364477290201098: 1, 9.360426238205816: 1, 9.359112899898932: 1, 9.356912555869899: 1, 9.343943368860165: 1, 9.327286397959025: 1, 9.31692849968129: 1, 9.31061369489739: 1, 9.308222094523051: 1, 9.3011933419009: 1, 9.298572626499775: 1, 9.294241339699564: 1, 9.292357117101027: 1, 9.260454288727566: 1, 9.248931418265581: 1, 9.247665399320338: 1, 9.235559508050574: 1, 9.233770634788737: 1, 9.212010069563936: 1, 9.210816613596815: 1, 9.203579405340449: 1, 9.193757328791737: 1, 9.193180727277786: 1, 9.193109682183508: 1, 9.18767779129146: 1, 9.178497597091406: 1, 9.173682747662541: 1, 9.143858741721552: 1, 9.13755649063146: 1, 9.132935762501019: 1, 9.130269035951624: 1, 9.129144140658694: 1, 9.117013735414655: 1, 9.102583513790302: 1, 9.096514338954968: 1, 9.09574109853652: 1, 9.09484945894454: 1, 9.07299229417006: 1, 9.066963303937408: 1, 9.05772968256668: 1, 9.049859451209118: 1, 9.035414747804182: 1, 9.034211678416053: 1, 9.021090276112622: 1, 9.014996236073035: 1, 9.014234823798626: 1, 8.999802894527003: 1, 8.98756574848961: 1, 8.983899347933905: 1, 8.970686408942116: 1, 8.962251903919814: 1, 8.961531777938577: 1, 8.953678619276468: 1, 8.926068444529196: 1, 8.916553847591173: 1, 8.909123234377235: 1, 8.902110671698226: 1, 8.891737320614526: 1, 8.891327638393882: 1, 8.887598883677214: 1, 8.881733661173328: 1, 8.872994179826541: 1, 8.866392086926865: 1, 8.866044457727716: 1, 8.84208378287369: 1, 8.812945022141067: 1, 8.80569926165593: 1, 8.802222613108027: 1, 8.801968566597276: 1, 8.77980781913478: 1, 8.765221827475905: 1, 8.76123884116243: 1, 8.753150199650957: 1, 8.736042795512471: 1, 8.727901654687566: 1, 8.72162807716526: 1, 8.69744079299056: 1, 8.685380223363962: 1, 8.682961132886243: 1, 8.671900713448707: 1, 8.640123812563974: 1, 8.632903230857165: 1, 8.632636480078293: 1, 8.625405037465343: 1, 8.602661552348692: 1, 8.602508321190765: 1, 8.60171744583948: 1, 8.552938754158935: 1, 8.54232198788755: 1, 8.536747142279346: 1, 8.519025861871468: 1, 8.506516698408683: 1, 8.503796804870433: 1, 8.46207344386721: 1, 8.449533658906093: 1, 8.443689618340247: 1, 8.434171094978657: 1, 8.410099492175016: 1, 8.388236256617674: 1, 8.384489395257258: 1, 8.378521600981752: 1, 8.352881455264507: 1, 8.350237579873847: 1, 8.323155313268291: 1, 8.306272305836648: 1, 8.305315502188945: 1, 8.302908016997643: 1, 8.293340857565163: 1, 8.28821166557618: 1, 8.286612142146735: 1, 8.286332026282238: 1, 8.280080207063104: 1, 8.267572008098226: 1, 8.265782430757676: 1, 8.259667731705877: 1, 8.242820124744215: 1, 8.236641626995759: 1, 8.221255045503757: 1, 8.211897048489316: 1, 8.204166702000123: 1, 8.193640763644238: 1, 8.191195354077294: 1, 8.169609102282609: 1, 8.15177215706071: 1, 8.128011685630103: 1, 8.121832496348327: 1, 8.12046482604217: 1, 8.108603644451925: 1, 8.103704027032036: 1, 8.085601538043642: 1, 8.082259849266059: 1, 8.073554073504138: 1, 8.063057545927215: 1, 8.062805004431937: 1, 8.052900117295223: 1, 8.027831220295552: 1, 8.015098062635047: 1, 8.002658230805851: 1, 7.998787641936155: 1, 7.990099954256712: 1, 7.981619843414272: 1, 7.975283797779613: 1, 7.9696036414641025: 1, 7.955628928693221: 1, 7.9438632234424595: 1, 7.921002667280391: 1, 7.890398602582164: 1, 7.88383896699531: 1, 7.865380507197981: 1, 7.852210813524375: 1, 7.814981044297964: 1, 7.792832657330623: 1, 7.784152579303389: 1, 7.775324730549827: 1, 7.7684709241647045: 1, 7.764274394391624: 1, 7.76138174644937: 1, 7.737914607081475: 1, 7.7364769795107025: 1, 7.7350931299857555: 1, 7.729096494994043: 1, 7.697711106071845: 1, 7.688097623947434: 1, 7.676102798701102: 1, 7.664954294943623: 1, 7.650530506592734: 1, 7.647209811110372: 1, 7.6408937452633126: 1, 7.627728713676731: 1, 7.626494930952992: 1, 7.625531878574687: 1, 7.615066623924195: 1, 7.613228236075018: 1, 7.603354102690637: 1, 7.602477520187944: 1, 7.5954169536746345: 1, 7.586202955010007: 1, 7.579257366543899: 1, 7.568481380574691: 1, 7.5415283860115725: 1, 7.5165498808614934: 1, 7.516267603413154: 1, 7.514830070226072: 1, 7.466836321612798: 1, 7.422709012207685: 1, 7.419570663086491: 1, 7.408166268866562: 1, 7.397913791986309: 1, 7.383408855992305: 1, 7.38310564220739: 1, 7.379149482511185: 1, 7.34142913650496: 1, 7.330745866016168: 1, 7.317796172957321: 1, 7.311992277073173: 1, 7.289289872425771: 1, 7.25770819691017: 1, 7.2449047934410284: 1, 7.235227850599645: 1, 7.205632491168385: 1, 7.157625183005836: 1, 7.13220674188097: 1, 7.109534242685308: 1, 7.103249215320328: 1, 7.0933504422319995: 1, 7.041709528046589: 1, 7.031374192481042: 1, 6.94590017334012: 1, 6.9366510169516244: 1, 6.933813875079658: 1, 6.9310517419489415: 1, 6.902994488022674: 1, 6.889632227842771: 1, 6.777980344475424: 1, 6.747301092580526: 1, 6.714160677733287: 1, 6.678362588867327: 1, 6.548985437099915: 1, 6.535078575324486: 1, 6.5173693856846215: 1, 6.435725915160505: 1, 6.401195679015439: 1, 6.2917970885178605: 1, 5.9754506438525326: 1})

In [57]:

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

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

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
```

```

# predict(X) Predict class labels for samples in X.

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

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

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

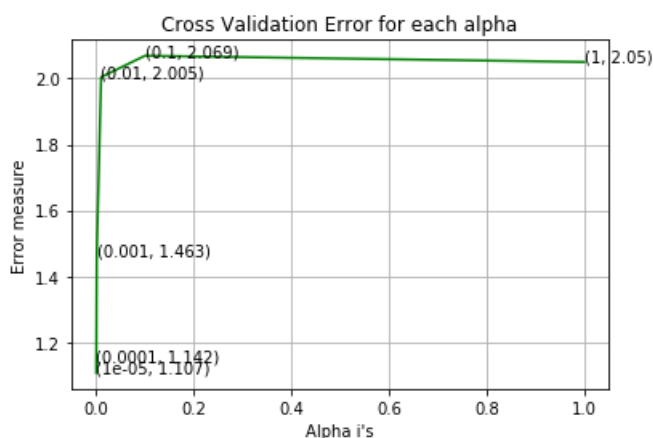
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```

For values of alpha = 1e-05 The log loss is: 1.1069459696999946
 For values of alpha = 0.0001 The log loss is: 1.1420375200398865
 For values of alpha = 0.001 The log loss is: 1.4630570563310146
 For values of alpha = 0.01 The log loss is: 2.0045317035313643
 For values of alpha = 0.1 The log loss is: 2.0692483788387004
 For values of alpha = 1 The log loss is: 2.0502500290445003



For values of best alpha = 1e-05 The train log loss is: 0.7305906615711405
 For values of best alpha = 1e-05 The cross validation log loss is: 1.1069459696999946
 For values of best alpha = 1e-05 The test log loss is: 1.0953012633364543

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

Ans. Yes, it seems like!

In [0]:

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

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

In [59]:

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

94.5 % of word of test data appeared in train data
94.4 % of word of Cross Validation appeared in train data

In [60]:

```
print()
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.field_names=["feature", "Train", "CV", "Test"]
ptable.add_row(["Gene ", "1.00", "1.14", "1.19"])
ptable.add_row(["Variation", "0.7", "1.7", "1.7"])
ptable.add_row(["Text", "0.82", "1.09", "1.124"])
print(ptable)
print()
```

```
+-----+-----+-----+-----+
| feature | Train | CV | Test |
+-----+-----+-----+-----+
| Gene   | 1.00 | 1.14 | 1.19 |
| Variation | 0.7 | 1.7 | 1.7 |
| Text   | 0.82 | 1.09 | 1.124 |
+-----+-----+-----+-----+
```

4. Machine Learning Models

In [0]:

```
#Data preparation for ML models.

#Misc. functionns for ML models

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

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

In [0]:

```
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 [0]:

```
# 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(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]"
                      .format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]"
                      .format(word,yes_no))

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

Stacking the three types of features

In [0]:

```
# merging gene, variance and text features

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

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

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
```

```

r()
train_y = np.array(list(train_df['Class']))

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

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

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

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

```

In [65]:

```

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

```

```

One hot encoding features :
(number of data points * number of features) in train data = (2124, 3197)
(number of data points * number of features) in test data = (665, 3197)
(number of data points * number of features) in cross validation data = (532, 3197)

```

In [66]:

```

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

```

```

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

```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [67]:

```

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

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.
# -----

```



```

# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-
algorithm-1/
# -----

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

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

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

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

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

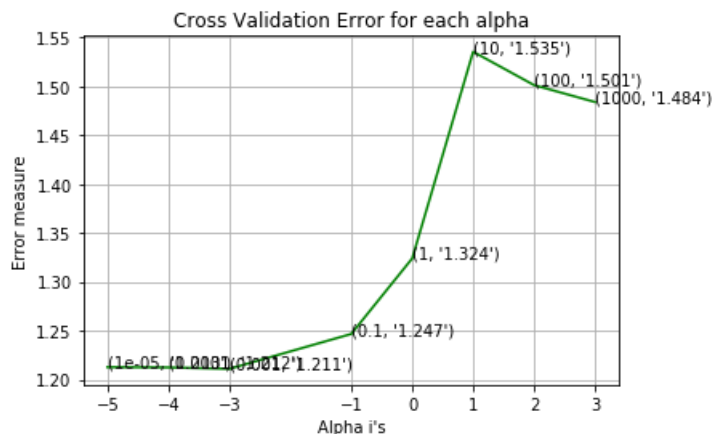
for alpha = 1e-05
Log Loss : 1.2125755929750142
for alpha = 0.0001
Log Loss : 1.2123004902465662
for alpha = 0.001
Log Loss : 1.210844748306353
for alpha = 0.1
Log Loss : 1.2465642825244505
for alpha = 1
Log Loss : 1.3239035850077896
for alpha = 10
Log Loss : 1.5354989466517543

```

```

for alpha = 100
Log Loss : 1.5012957920531036
for alpha = 1000
Log Loss : 1.4839734630756507

```



```

For values of best alpha = 0.001 The train log loss is: 0.5434329610130213
For values of best alpha = 0.001 The cross validation log loss is: 1.210844748306353
For values of best alpha = 0.001 The test log loss is: 1.144397692835139

```

4.1.1.2. Testing the model with best hyper paramters

In [68]:

```

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

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

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

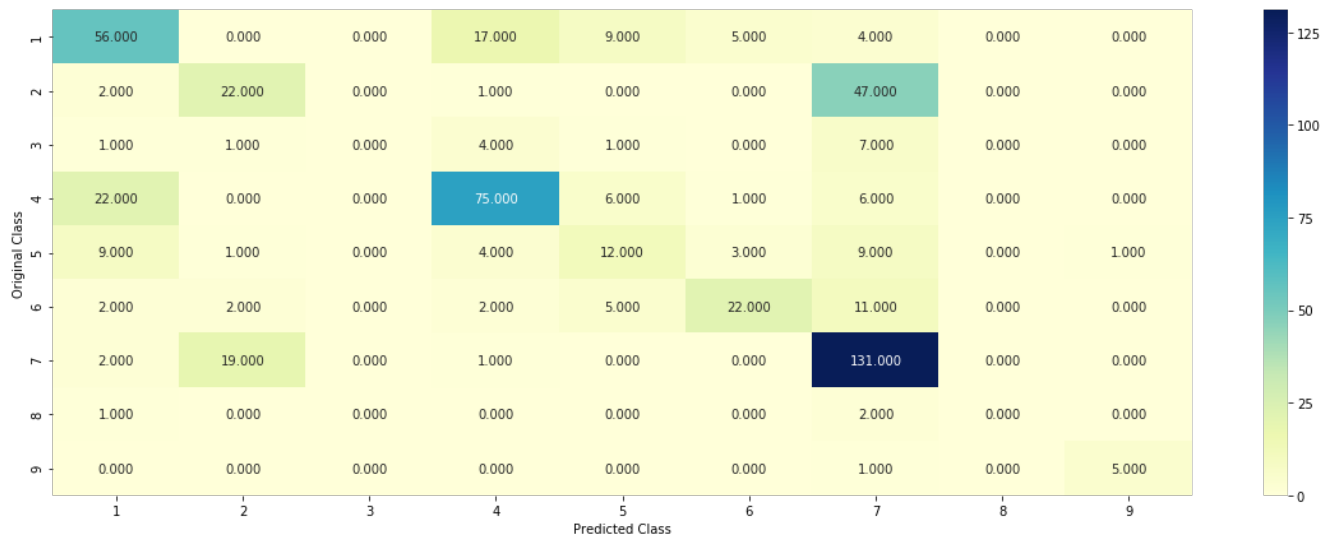
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilies we use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

```

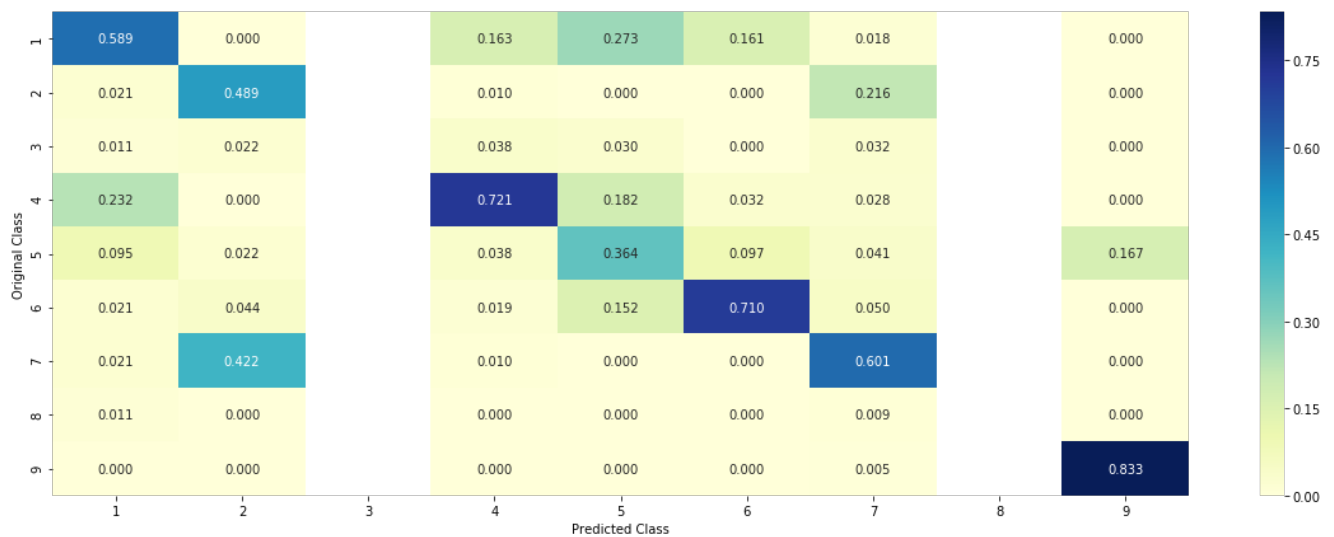
```

Log Loss : 1.210844748306353
Number of missclassified point : 0.39285714285714285
----- Confusion matrix -----

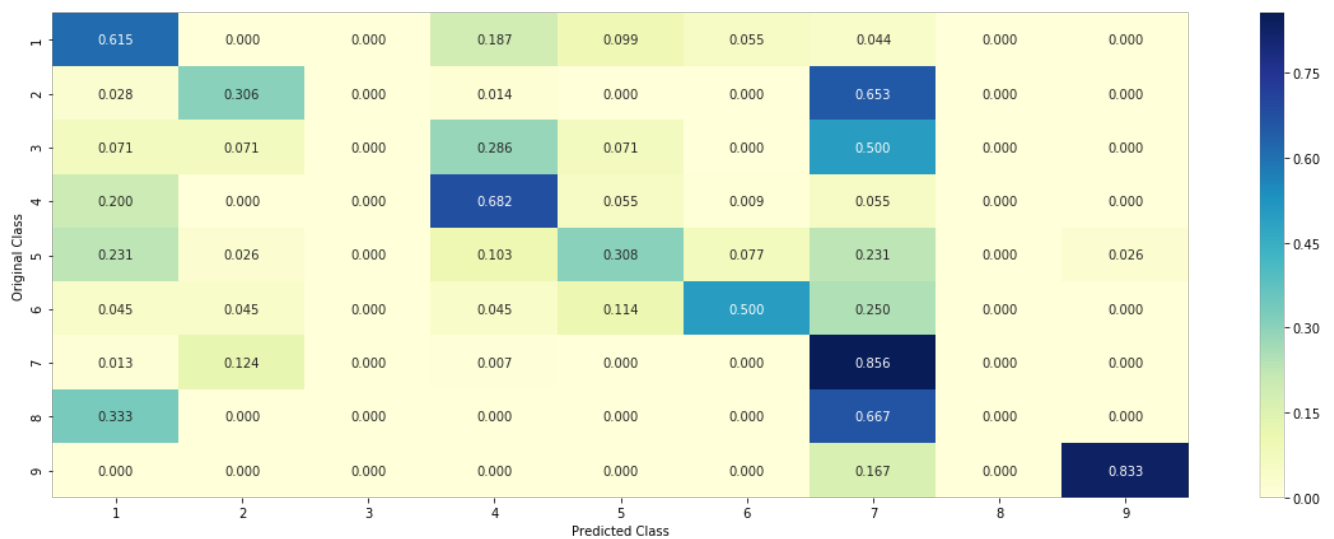
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [69]:

```
test_point_index = 1
```

```

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

```

Predicted Class : 6
Predicted Class Probabilities: [[0.0862 0.0513 0.0141 0.1126 0.0565 0.5613 0.111 0.0035 0.0035]]
Actual Class : 6

21 Text feature [1093] present in test data point [True]
29 Text feature [13] present in test data point [True]
34 Text feature [000] present in test data point [True]
Out of the top 100 features 3 are present in query point

4.1.1.4. Feature Importance, Incorrectly classified point

In [70]:

```

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

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0649 0.135 0.011 0.0689 0.035 0.0351 0.644 0.0031 0.003]]
Actual Class : 2

54 Text feature [100] present in test data point [True]
Out of the top 100 features 1 are present in query point

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [71]:

```

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

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

# find more about CalibratedClassifierCV here at http://scikit-

```

```

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

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

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

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

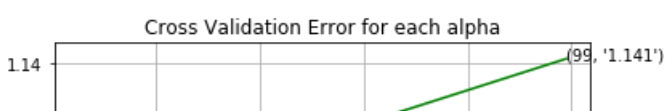
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha],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

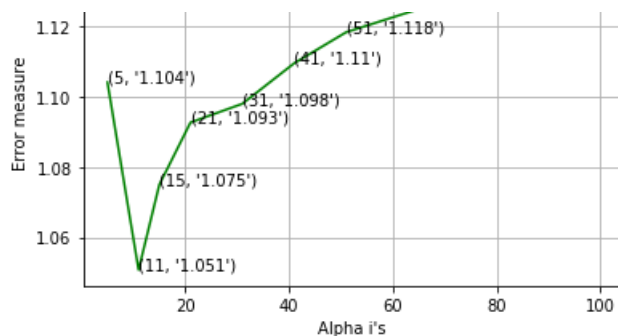
```

```

for alpha = 5
Log Loss : 1.1041895517146412
for alpha = 11
Log Loss : 1.050662310152841
for alpha = 15
Log Loss : 1.0749793348560823
for alpha = 21
Log Loss : 1.0927206671680691
for alpha = 31
Log Loss : 1.0980682733516993
for alpha = 41
Log Loss : 1.1096475806100037
for alpha = 51
Log Loss : 1.1184911601659036
for alpha = 99
Log Loss : 1.1411770041583293

```





For values of best alpha = 11 The train log loss is: 0.640073218674863

For values of best alpha = 11 The cross validation log loss is: 1.050662310152841

For values of best alpha = 11 The test log loss is: 0.9543879907068362

4.2.2. Testing the model with best hyper paramters

In [72]:

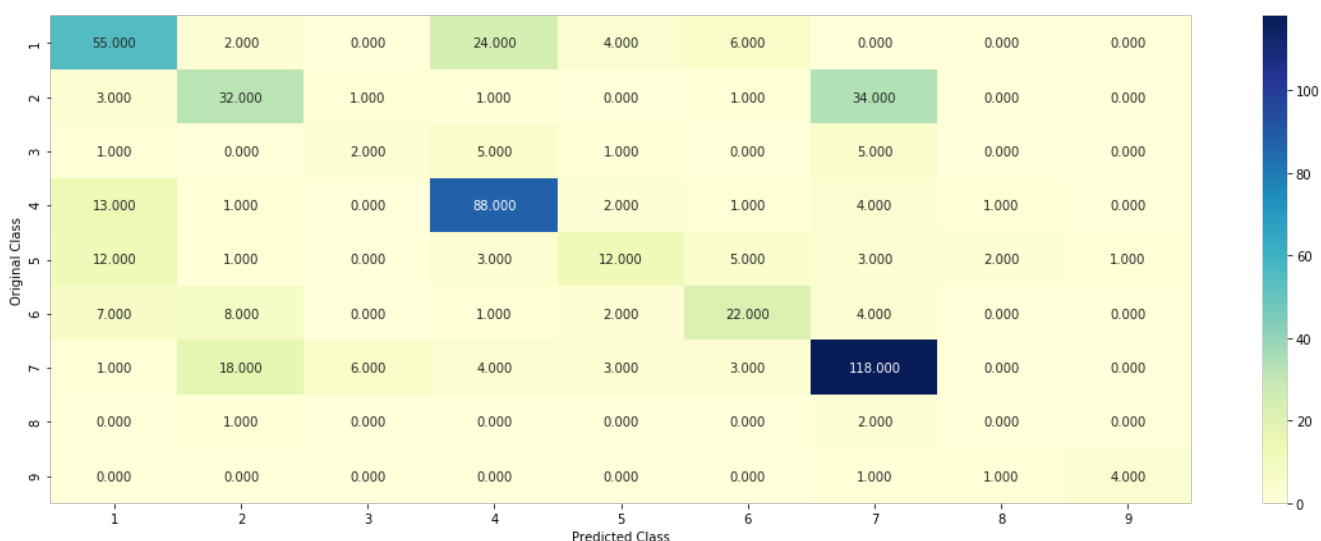
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None,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

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

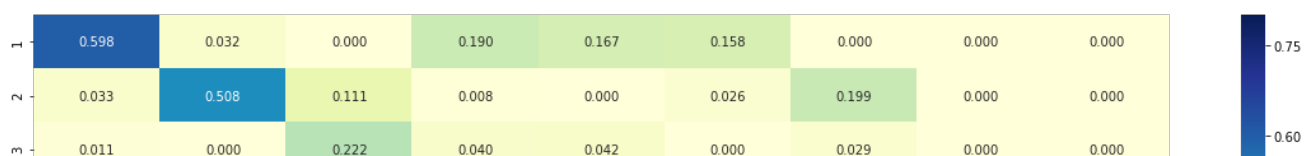
Log loss : 1.050662310152841

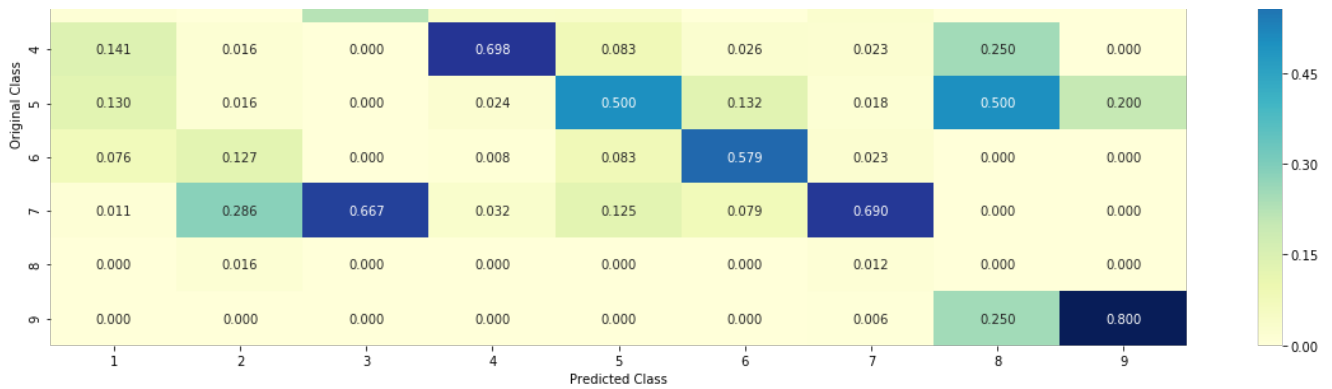
Number of mis-classified points : 0.37406015037593987

----- Confusion matrix -----

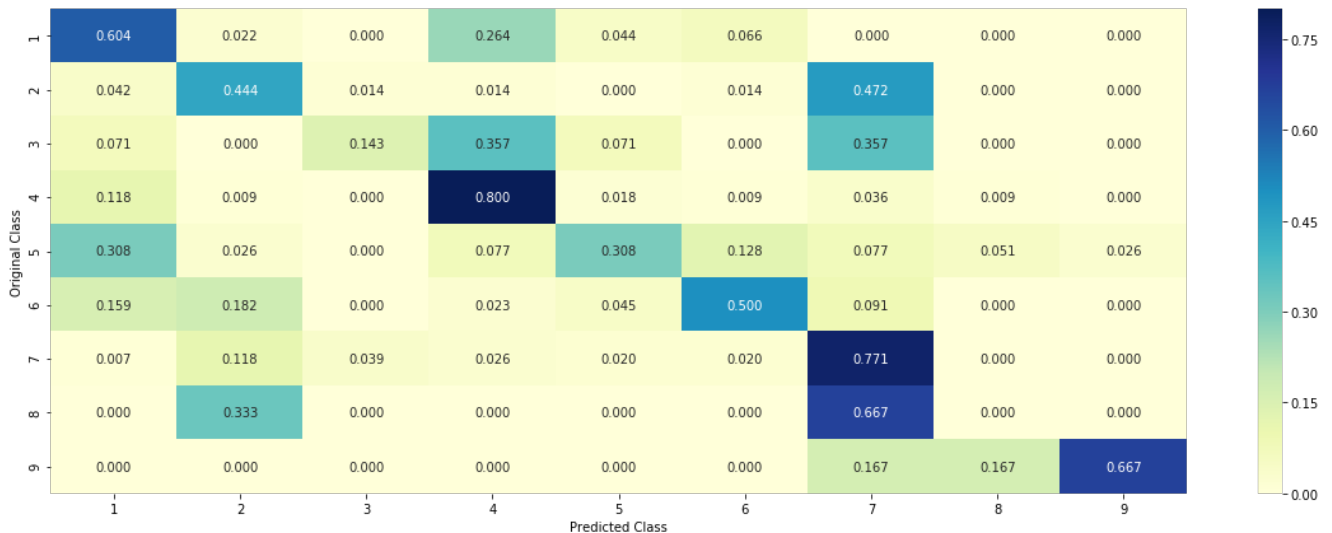


----- Precision matrix (Column Sum=1) -----





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



4.2.3. Sample Query point -1

In [73]:

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

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

Predicted Class : 9

Actual Class : 6

The 11 nearest neighbours of the test points belongs to classes [6 6 6 6 6 6 6 6 6 6 6]

Frequency of nearest points : Counter({6: 11})

4.2.4. Sample Query Point-2

In [74]:

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

```

test_point_index = 100

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

```

```

Predicted Class : 7
Actual Class : 2
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [2 7 7 7 2 2 2 7 7 7]
Frequency of nearest points : Counter({7: 7, 2: 4})

```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [75]:

```

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

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

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

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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)

    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))

```



```

# to avoid rounding error while multiplying probabilities we use log-probability estimates
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

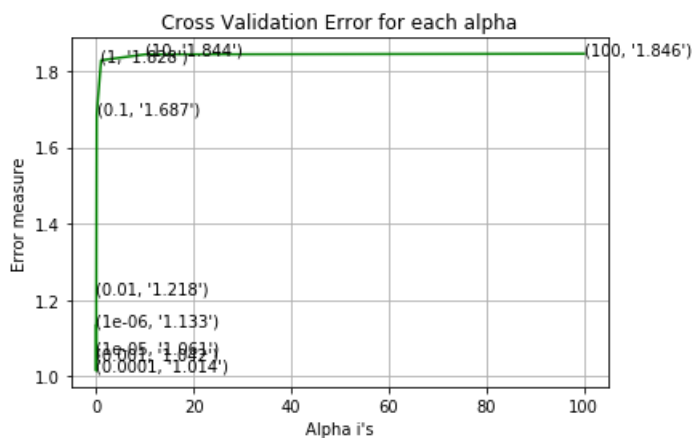
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-06
Log Loss : 1.1332607894229252
for alpha = 1e-05
Log Loss : 1.0610327461109685
for alpha = 0.0001
Log Loss : 1.0138413983629366
for alpha = 0.001
Log Loss : 1.042474659076955
for alpha = 0.01
Log Loss : 1.217843678634886
for alpha = 0.1
Log Loss : 1.6868047353360354
for alpha = 1
Log Loss : 1.8284832711964811
for alpha = 10
Log Loss : 1.843791685699172
for alpha = 100
Log Loss : 1.8455839386011739

```



```

For values of best alpha = 0.0001 The train log loss is: 0.45786829431688403
For values of best alpha = 0.0001 The cross validation log loss is: 1.0138413983629366
For values of best alpha = 0.0001 The test log loss is: 0.9218869949078555

```

4.3.1.2. Testing the model with best hyper paramters

In [76]:

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

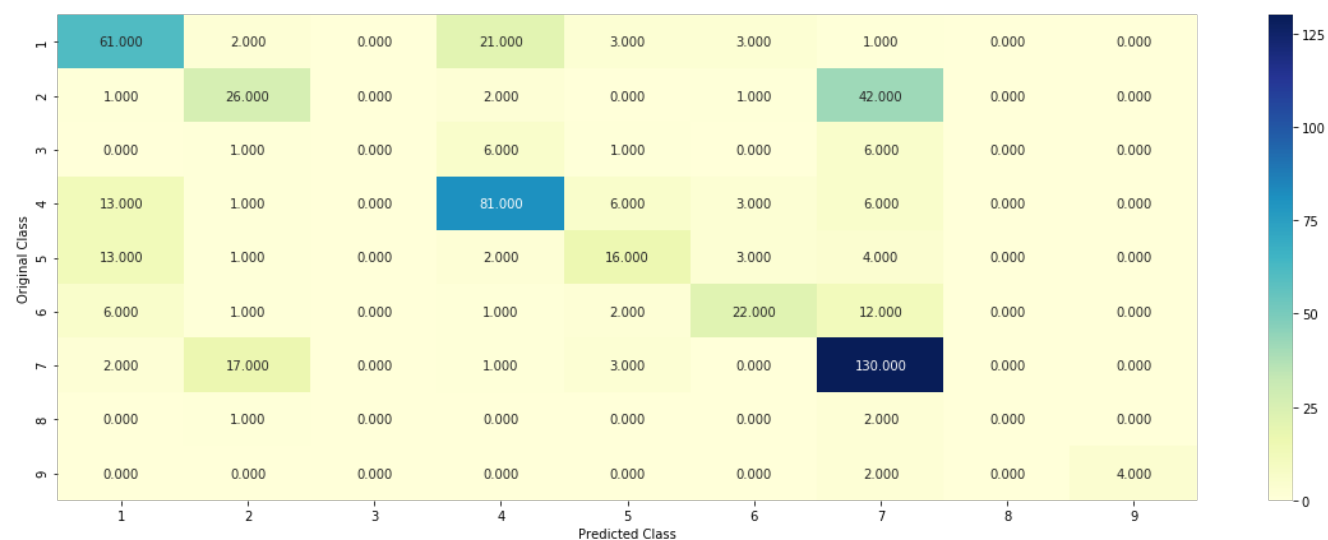
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

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

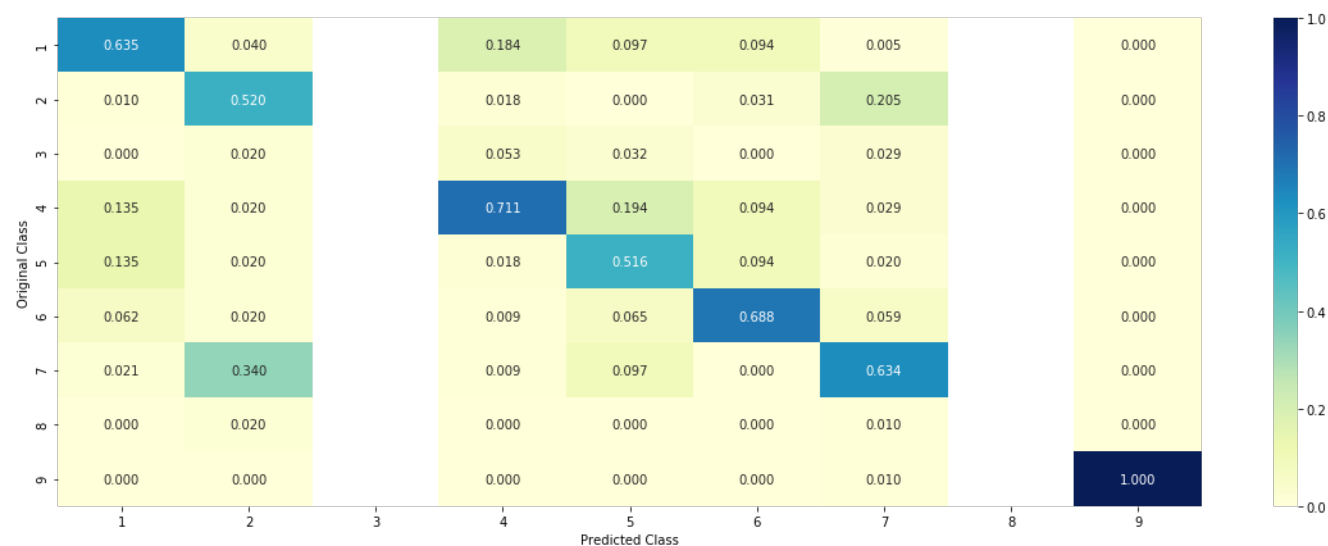
Log loss : 1.0138413983629366

Number of mis-classified points : 0.3609022556390977

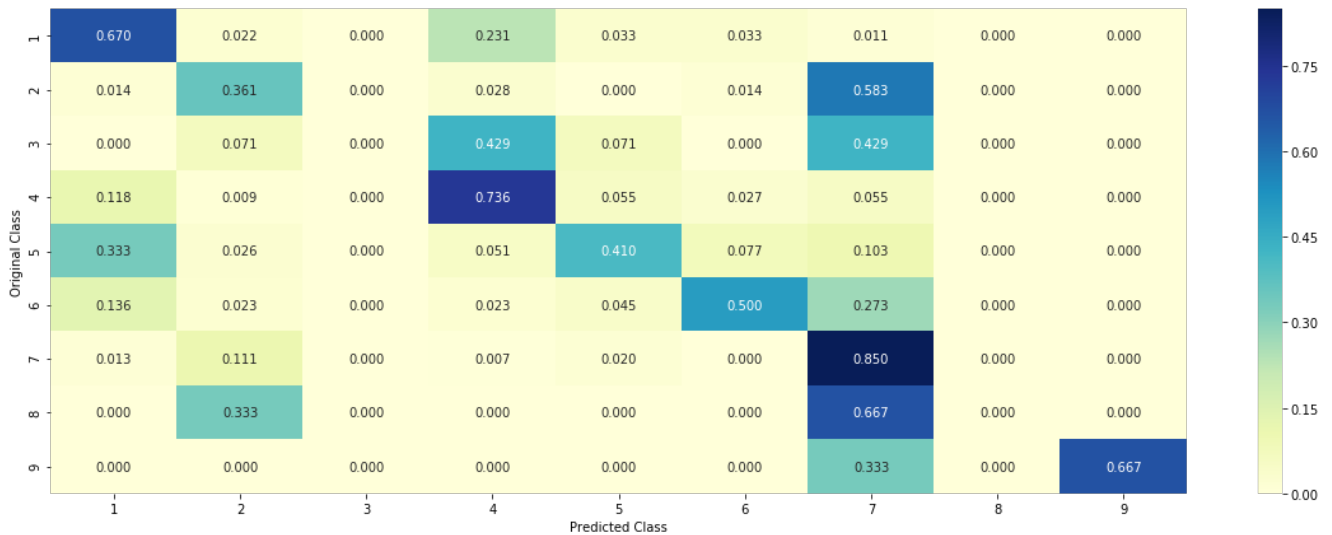
----- Confusion matrix -----



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



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



4.3.1.3. Feature Importance

In [0]:

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

4.3.1.3.1. Correctly Classified point

In [78]:

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

Predicted Class : 6

Predicted Class Probabilities: [[0.0441 0.0049 0.0217 0.1046 0.0389 0.7784 0.0031 0.0031 0.0012]]

Actual Class : 6

349 Text feature [11] present in test data point [True]

Out of the top 500 features 1 are present in query point

4.3.1.3.2. Incorrectly Classified point

In [79]:

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

Predicted Class : 7

Predicted Class Probabilities: [[0.0183 0.1899 0.0021 0.0212 0.013 0.0014 0.7516 0.0014 0.0012]]

Actual Class : 2

24 Text feature [100] present in test data point [True]
115 Text feature [005228] present in test data point [True]
473 Text feature [05] present in test data point [True]
Out of the top 500 features 3 are present in query point

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [80]:

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

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

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

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

alpha = [10 ** x for x in range(-6, 1)]
```

```

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

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

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

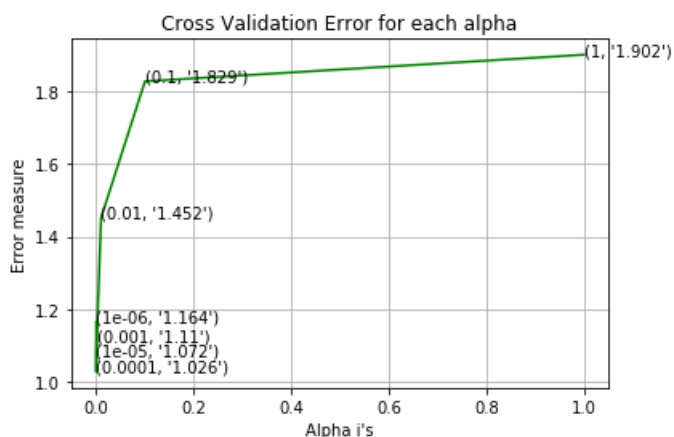
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-06
Log Loss : 1.1644814258435812
for alpha = 1e-05
Log Loss : 1.0715194644434234
for alpha = 0.0001
Log Loss : 1.0263873522848197
for alpha = 0.001
Log Loss : 1.1099980263917337
for alpha = 0.01
Log Loss : 1.4521864672252107
for alpha = 0.1
Log Loss : 1.8289766949862687
for alpha = 1
Log Loss : 1.9024470396998319

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4522444261875607
For values of best alpha = 0.0001 The cross validation log loss is: 1.0263873522848197
For values of best alpha = 0.0001 The test log loss is: 0.9455680308302007

```

4.3.2.2. Testing model with best hyper parameters

In [81]:

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

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

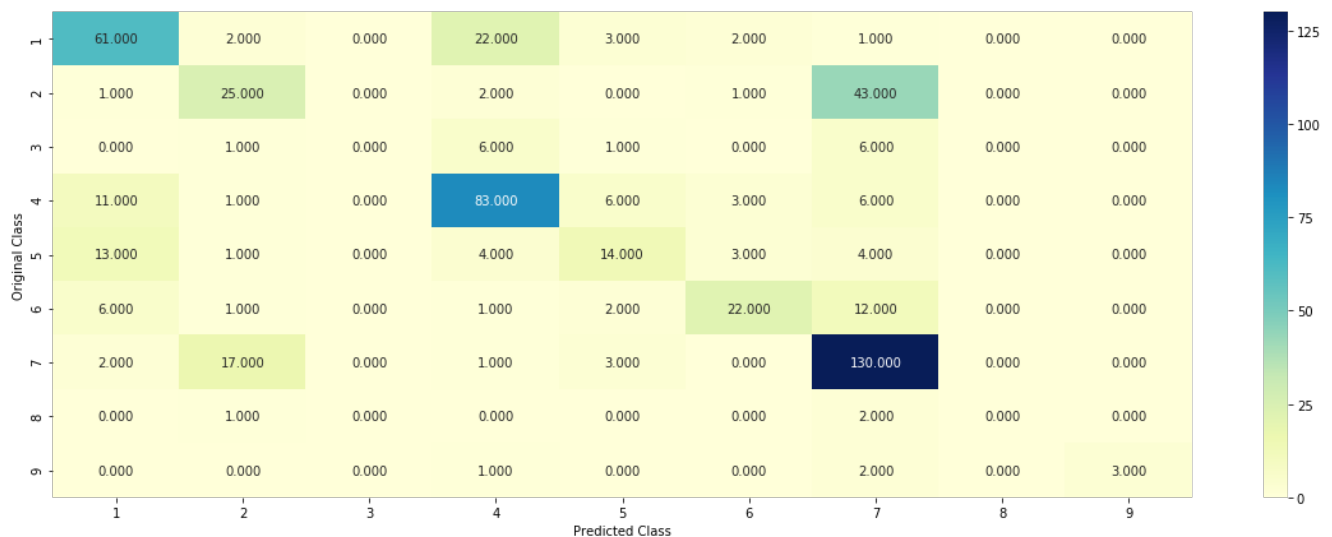
#-----
# video link:
#-----

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

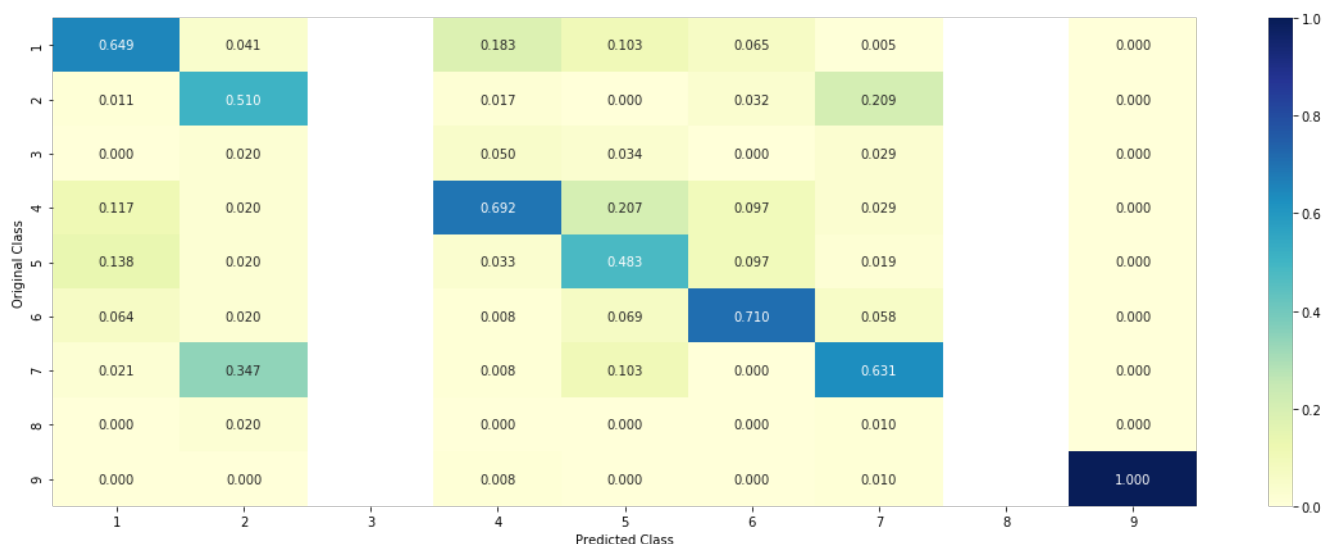
Log loss : 1.0263873522848197

Number of mis-classified points : 0.36466165413533835

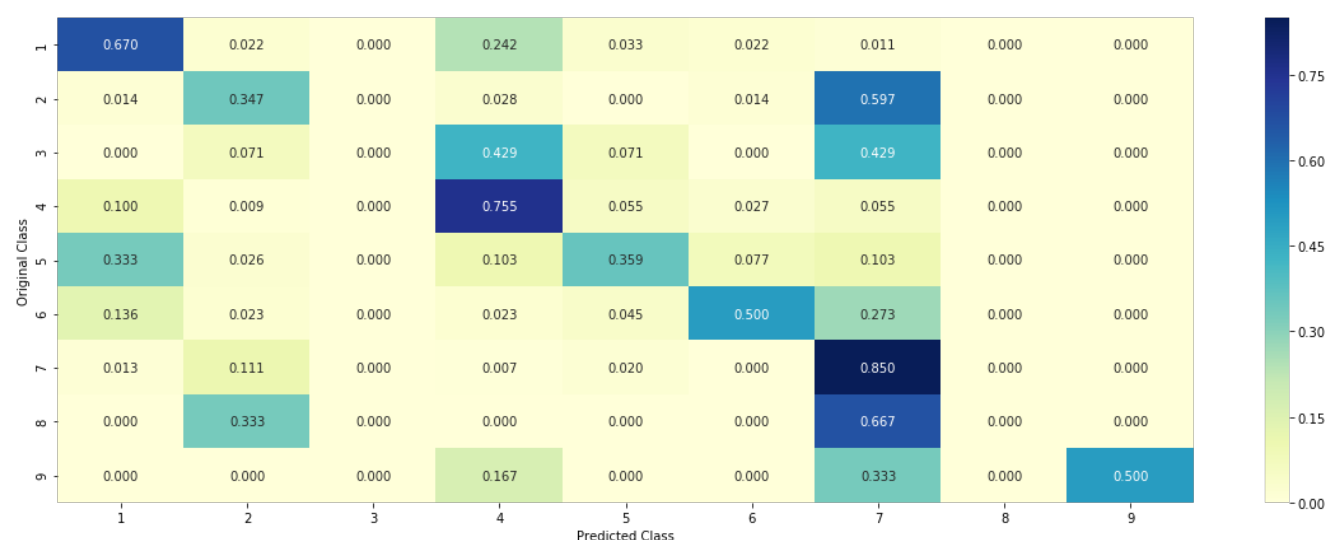
----- Confusion matrix -----



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [82]:

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

Predicted Class : 6

Predicted Class Probabilities: [[3.650e-02 5.600e-03 1.470e-02 9.920e-02 3.840e-02 7.989e-01 4.000e-03 2.300e-03 5.000e-04]]

Actual Class : 6

84 Text feature [11] present in test data point [True]
Out of the top 500 features 1 are present in query point

4.3.2.4. Feature Importance, Incorrectly Classified point

In [83]:

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

Predicted Class : 7

Predicted Class Probabilities: [[2.060e-02 1.842e-01 1.800e-03 2.420e-02 1.180e-02 1.300e-03 7.532e-01 1.180e-02 1.300e-03]]

```

2.300e-03 6.000e-04]]
Actual Class : 2
-----
153 Text feature [005228] present in test data point [True]
375 Text feature [100] present in test data point [True]
Out of the top 500 features 2 are present in query point

```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [84]:

```

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

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

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

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

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

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

```



```

best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=1, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

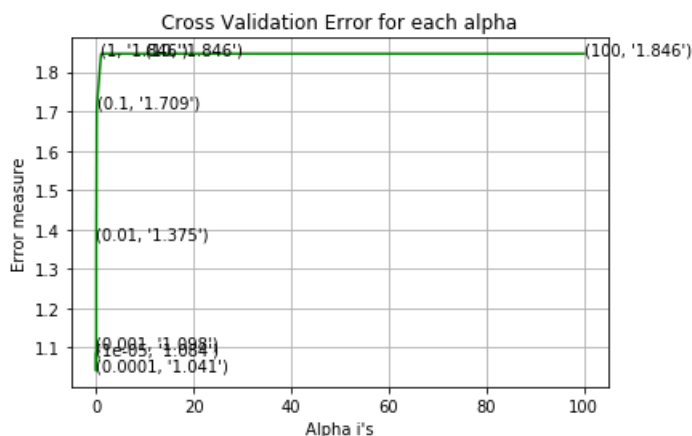
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for C = 1e-05
Log Loss : 1.0843744302025784
for C = 0.0001
Log Loss : 1.040900817608454
for C = 0.001
Log Loss : 1.0984602985534544
for C = 0.01
Log Loss : 1.374999316306491
for C = 0.1
Log Loss : 1.7089660636280517
for C = 1
Log Loss : 1.8461149368397085
for C = 10
Log Loss : 1.8461149449892185
for C = 100
Log Loss : 1.8461149291926422

```



```

For values of best alpha = 0.0001 The train log loss is: 0.40125181651380915
For values of best alpha = 0.0001 The cross validation log loss is: 1.040900817608454
For values of best alpha = 0.0001 The test log loss is: 0.9501105256480938

```

4.4.2. Testing model with best hyper parameters

In [85]:

```

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

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

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

```

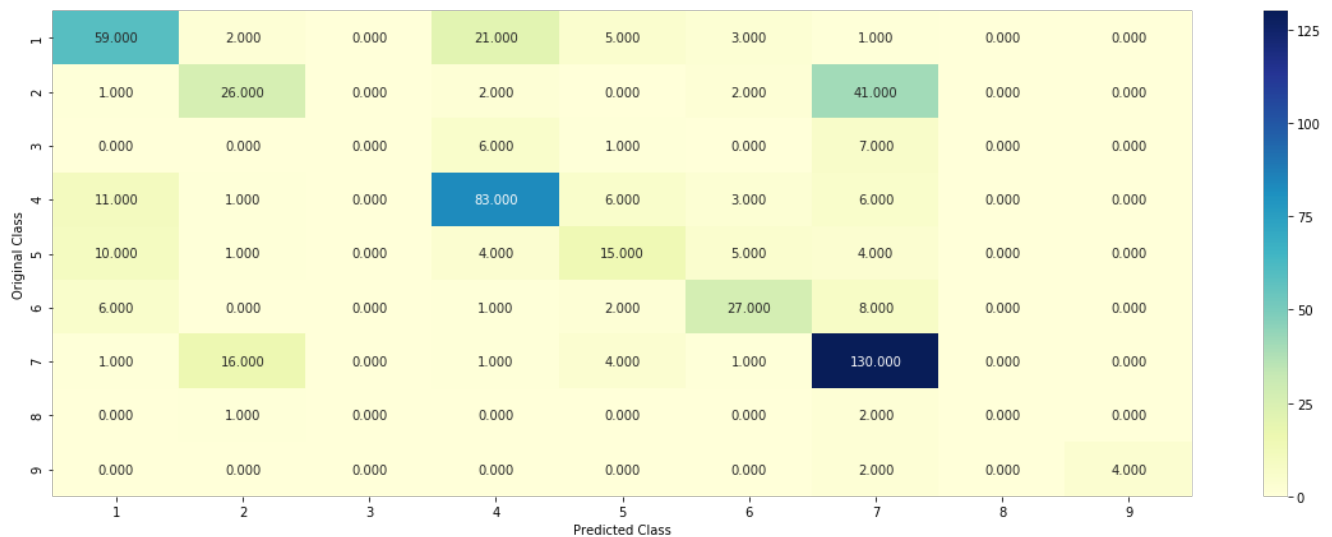
```
# predict(A) PERFORM CLASSIFICATION ON SAMPLES IN A.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

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

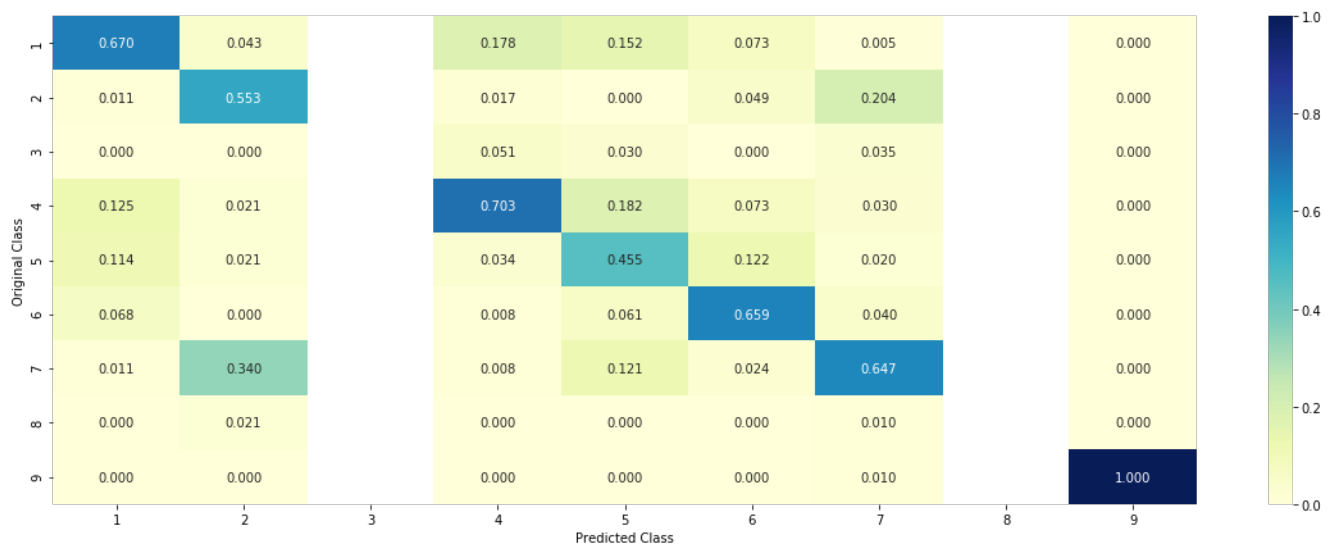
Log loss : 1.040900817608454

Number of mis-classified points : 0.3533834586466165

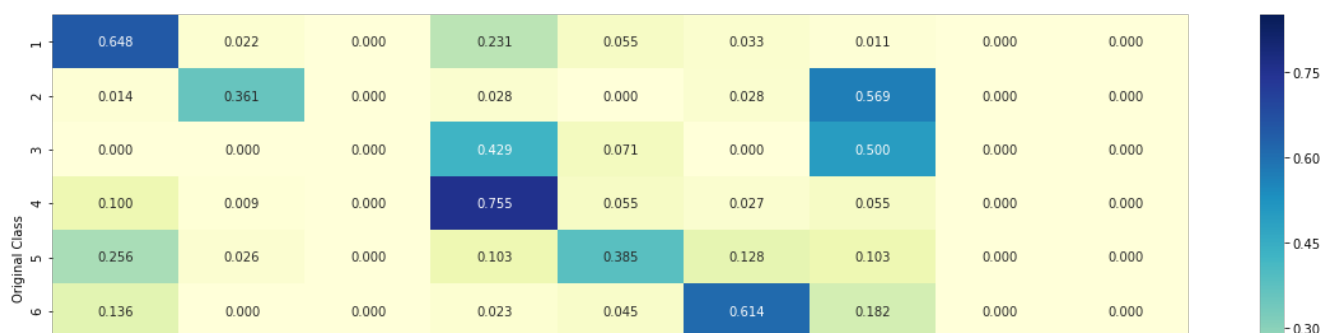
----- Confusion matrix -----

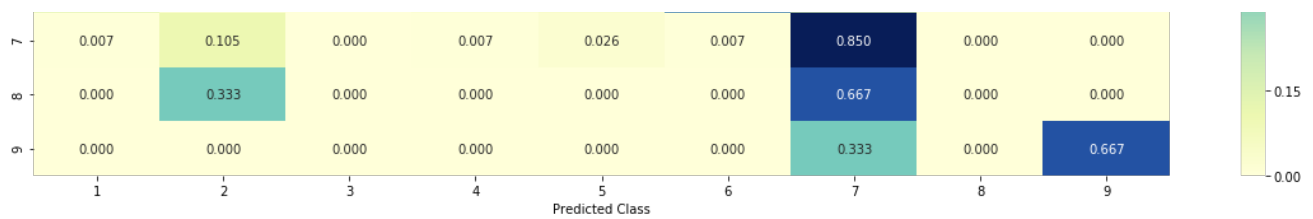


----- Precision matrix (Column Sum=1) -----



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





4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [86]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0432 0.0406 0.0175 0.1203 0.013 0.7397 0.021 0.0025 0.0021]]
Actual Class : 6
-----
Out of the top 500 features 0 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [87]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0184 0.235 0.0054 0.0244 0.0439 0.0019 0.6657 0.0015 0.0039]]
Actual Class : 2
-----
Out of the top 500 features 0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [88]:

```
# -----
```

```

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

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

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

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

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

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

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

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

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

```

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

```
for n_estimators = 100 and max depth = 5
Log Loss : 1.249991703830645
for n_estimators = 100 and max depth = 10
Log Loss : 1.2634434666158416
for n_estimators = 200 and max depth = 5
Log Loss : 1.2286940955122976
for n_estimators = 200 and max depth = 10
Log Loss : 1.257575767970522
for n_estimators = 500 and max depth = 5
Log Loss : 1.2233633127390762
for n_estimators = 500 and max depth = 10
Log Loss : 1.2482453961751725
for n_estimators = 1000 and max depth = 5
Log Loss : 1.221825402235456
for n_estimators = 1000 and max depth = 10
Log Loss : 1.245599658787943
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2179648371934533
for n_estimators = 2000 and max depth = 10
Log Loss : 1.2444717603770685
For values of best estimator = 2000 The train log loss is: 0.8684952301597172
For values of best estimator = 2000 The cross validation log loss is: 1.2179648371934533
For values of best estimator = 2000 The test log loss is: 1.1528420612739096
```

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

In [89]:

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

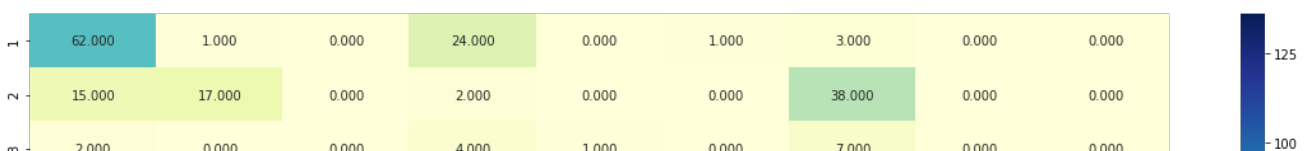
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

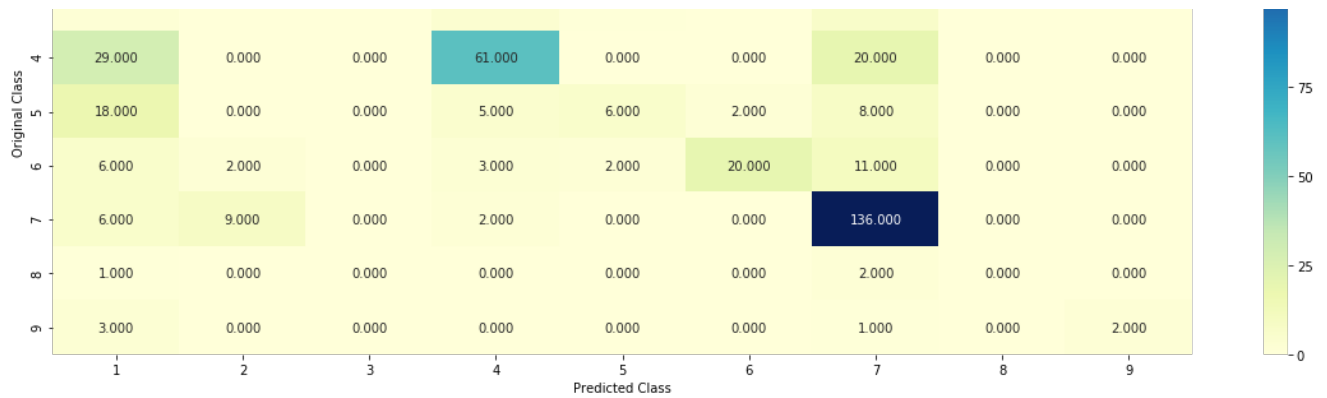
# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

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

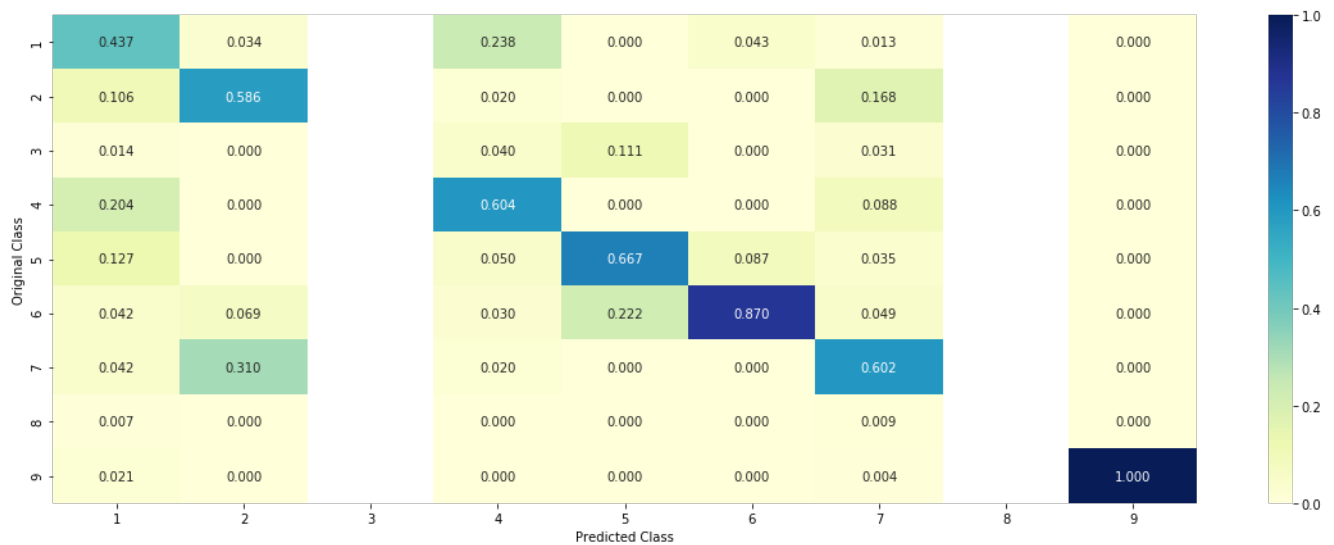
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha*2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.2179648371934533
Number of mis-classified points : 0.42857142857142855
----- Confusion matrix -----
```

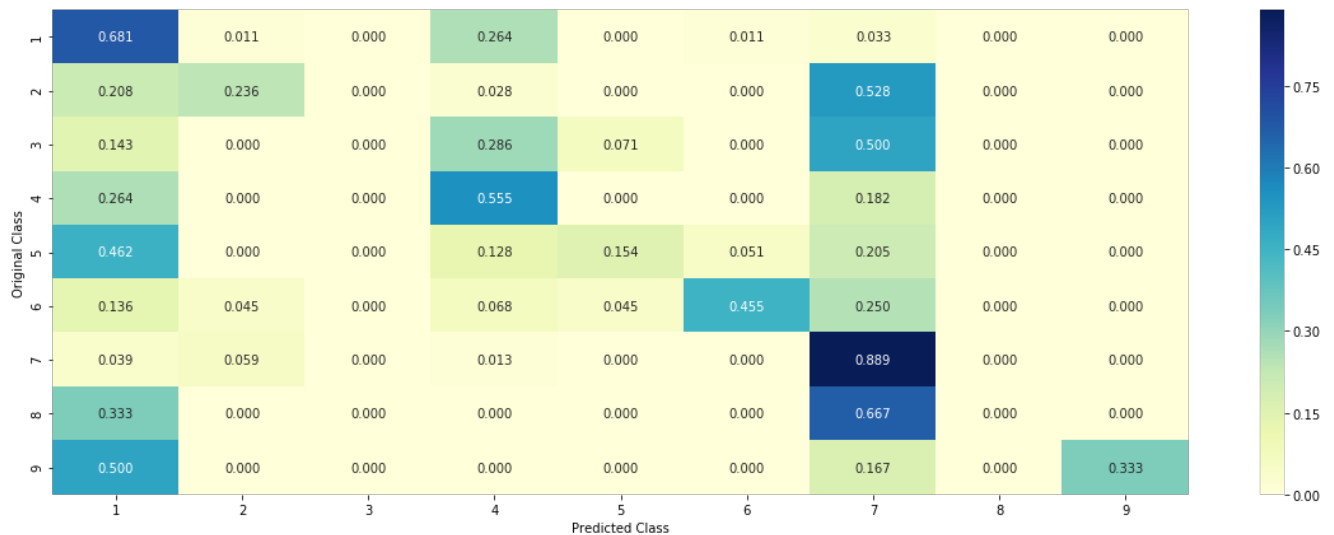




----- Precision matrix (Column Sum=1) -----



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [90]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha*2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
```

```

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```

```

Predicted Class : 6
Predicted Class Probabilities: [[0.1236 0.0088 0.0136 0.1526 0.0491 0.6265 0.0197 0.0024 0.0036]]
Actual Class : 6

```

Out of the top 100 features 0 are present in query point

4.5.3.2. Inorrectly Classified point

In [91]:

```

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

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0491 0.3776 0.0162 0.0341 0.0468 0.046 0.4194 0.0063 0.0043]]
Actual Class : 2

```

Out of the top 100 features 0 are present in query point

4.5.3. Hyper paramter tuning (With Response Coding)

In [92]:

```

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

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

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

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

```

```

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

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

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

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

```

```

for n_estimators = 10 and max depth = 2
Log Loss : 2.1080121490953228
for n_estimators = 10 and max depth = 3
Log Loss : 1.7875330928728776
for n_estimators = 10 and max depth = 5
Log Loss : 1.6454209542287228
for n_estimators = 10 and max depth = 10
Log Loss : 1.7167343424292627
for n_estimators = 50 and max depth = 2
Log Loss : 1.6593102816878993
for n_estimators = 50 and max depth = 3
Log Loss : 1.4358447509849843
for n_estimators = 50 and max depth = 5
Log Loss : 1.3355283052681304
for n_estimators = 50 and max depth = 10
Log Loss : 1.653890423888417

```



```

Log Loss : 1.55691619818187
for n_estimators = 100 and max depth = 2
Log Loss : 1.4899170842820315
for n_estimators = 100 and max depth = 3
Log Loss : 1.2904476630206279
for n_estimators = 100 and max depth = 5
Log Loss : 1.2904476630206279
for n_estimators = 100 and max depth = 10
Log Loss : 1.6144390202839471
for n_estimators = 200 and max depth = 2
Log Loss : 1.5916970066602578
for n_estimators = 200 and max depth = 3
Log Loss : 1.4802244802840412
for n_estimators = 200 and max depth = 5
Log Loss : 1.3442855750037563
for n_estimators = 200 and max depth = 10
Log Loss : 1.5759808242107127
for n_estimators = 500 and max depth = 2
Log Loss : 1.6408089826467431
for n_estimators = 500 and max depth = 3
Log Loss : 1.5226462175717674
for n_estimators = 500 and max depth = 5
Log Loss : 1.3440470631973764
for n_estimators = 500 and max depth = 10
Log Loss : 1.6195373663093586
for n_estimators = 1000 and max depth = 2
Log Loss : 1.6376061369963493
for n_estimators = 1000 and max depth = 3
Log Loss : 1.5380426694927507
for n_estimators = 1000 and max depth = 5
Log Loss : 1.3554868271063036
for n_estimators = 1000 and max depth = 10
Log Loss : 1.6386133055541394
For values of best alpha = 100 The train log loss is: 0.06119518112523
For values of best alpha = 100 The cross validation log loss is: 1.2904476630206279
For values of best alpha = 100 The test log loss is: 1.1845141128617311

```

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

In [93]:

```

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

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

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

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

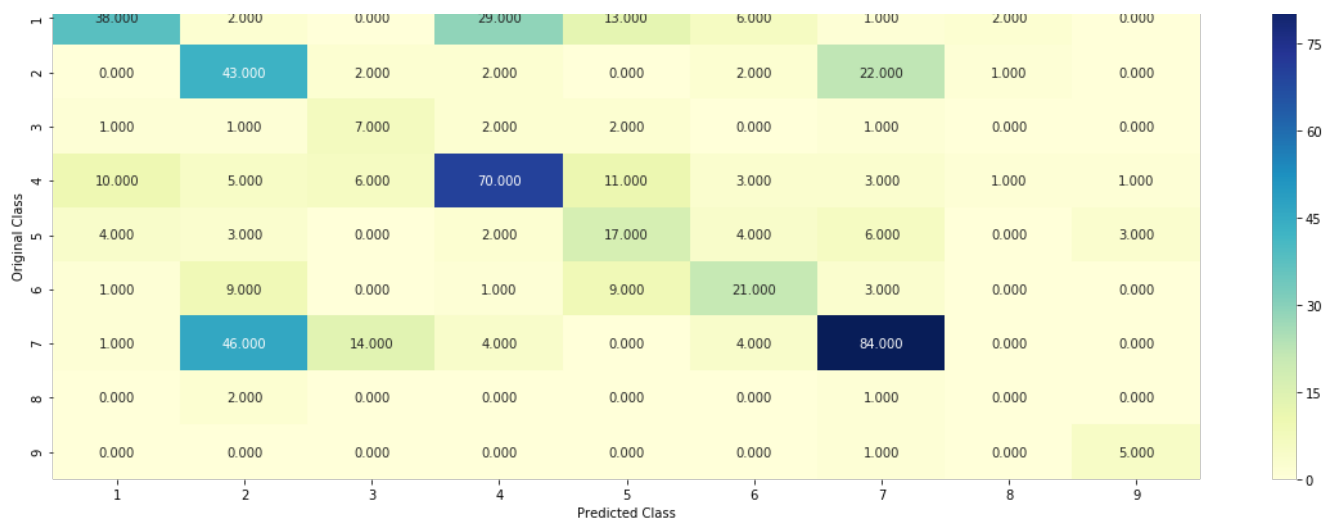
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)],
n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='auto', random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)

```

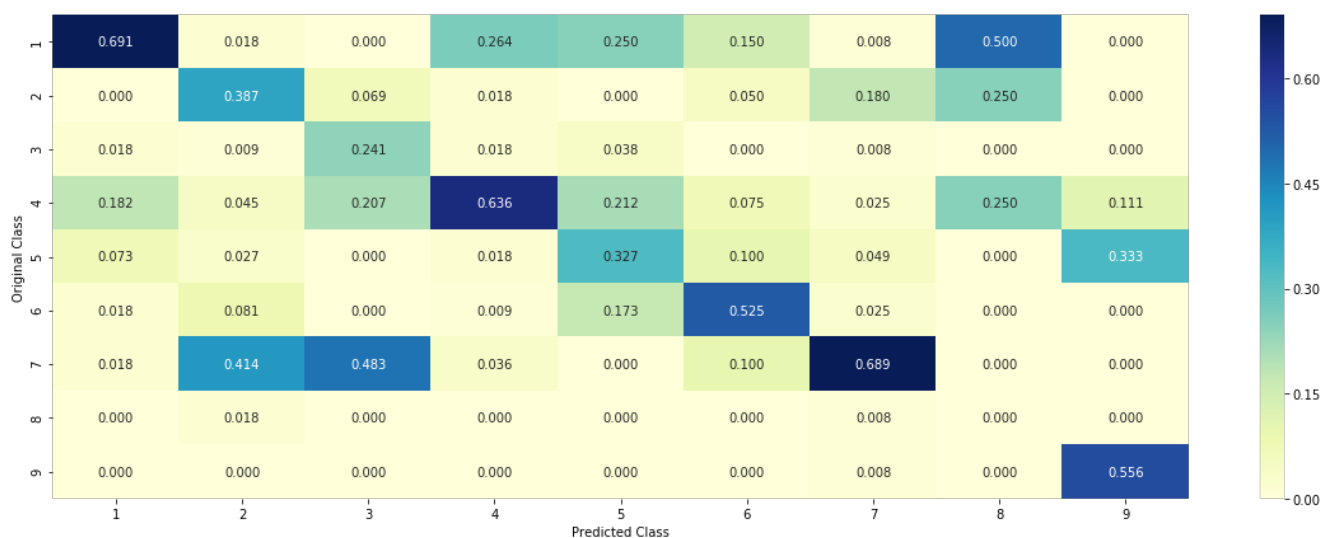
```

Log loss : 1.2904476630206279
Number of mis-classified points : 0.4642857142857143
----- Confusion matrix -----

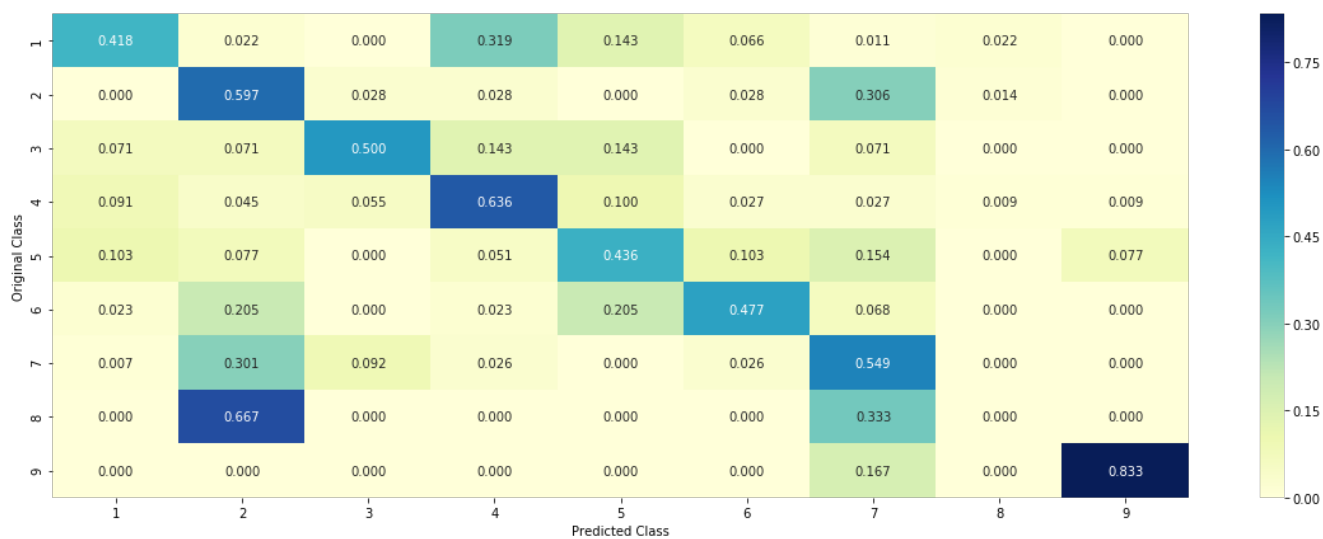
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [94]:

```

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

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

```

```

Predicted Class : 6
Predicted Class Probabilities: [[0.0138 0.0046 0.1074 0.0267 0.1574 0.6791 0.0039 0.0036 0.0033]]
Actual Class : 6

```

```

-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature

```

4.5.5.2. Incorrectly Classified point

In [95]:

```

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

```

```
print("Text is important feature")
```

Predicted Class : 2

Predicted Class Probabilities: [[0.0129 0.7378 0.0286 0.0137 0.0169 0.0288 0.1391 0.0149 0.0073]]

Actual Class : 2

```
-----  
Variation is important feature  
Variation is important feature  
Variation is important feature  
Variation is important feature  
Variation is important feature  
Gene is important feature  
Variation is important feature  
Text is important feature  
Text is important feature  
Gene is important feature  
Text is important feature  
Text is important feature  
Text is important feature  
Text is important feature  
Gene is important feature  
Variation is important feature  
Gene is important feature  
Text is important feature  
Gene is important feature  
Variation is important feature  
Gene is important feature  
Variation is important feature  
Text is important feature  
Text is important feature  
Gene is important feature  
Gene is important feature  
Text is important feature  
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [96]:

```
# read more about SGDClassifier() at http://scikit-  
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html  
# -----  
# default parameters  
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i  
ter=None, tol=None,  
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0  
=0.0, power_t=0.5,  
# class_weight=None, warm_start=False, average=False, n_iter=None)  
  
# some of methods  
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.  
# predict(X) Predict class labels for samples in X.  
  
#-----  
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in  
tuition-1/  
#-----  
  
# read more about support vector machines with linear kernels here http://scikit-  
learn.org/stable/modules/generated/sklearn.svm.SVC.html  
# -----  
# default parameters  
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t  
ol=0.001,  
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra  
ndom_state=None)  
  
# Some of methods of SVM()  
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.  
# predict(X) Perform classification on samples in X.  
# -----
```

```

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

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

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

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

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

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

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

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

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

```

```

Logistic Regression : Log Loss: 1.04
Support vector machines : Log Loss: 1.85
Naive Bayes : Log Loss: 1.21
-----

```

```

Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.035
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.514

```

Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.181
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.398
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.858

4.7.2 testing the model with the best hyper parameters

In [97]:

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

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

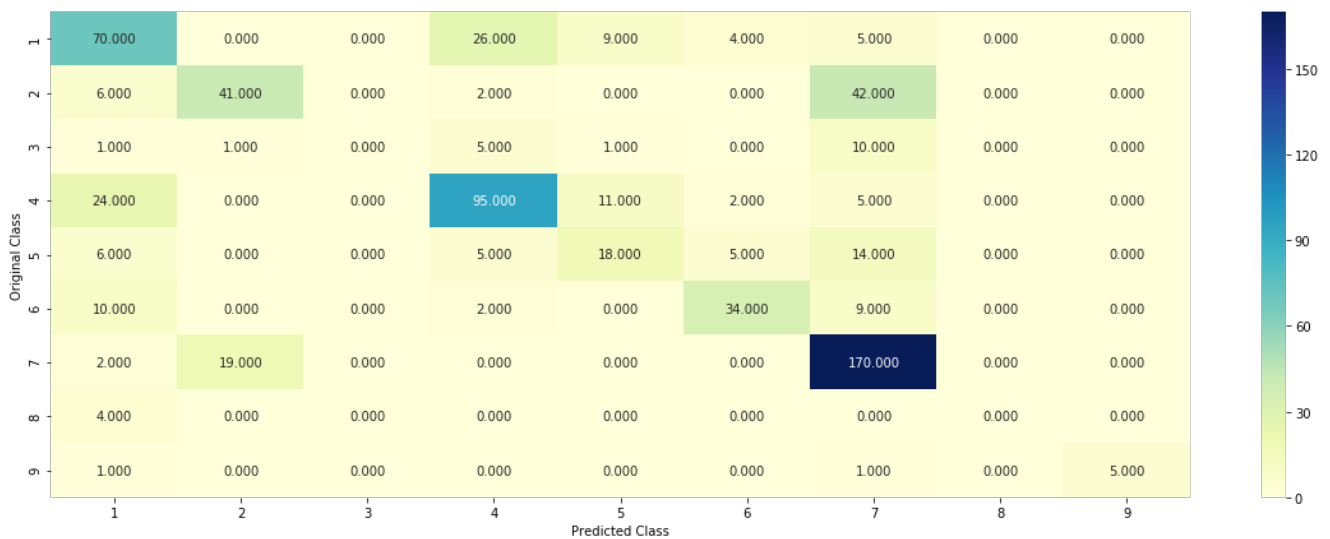
log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

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

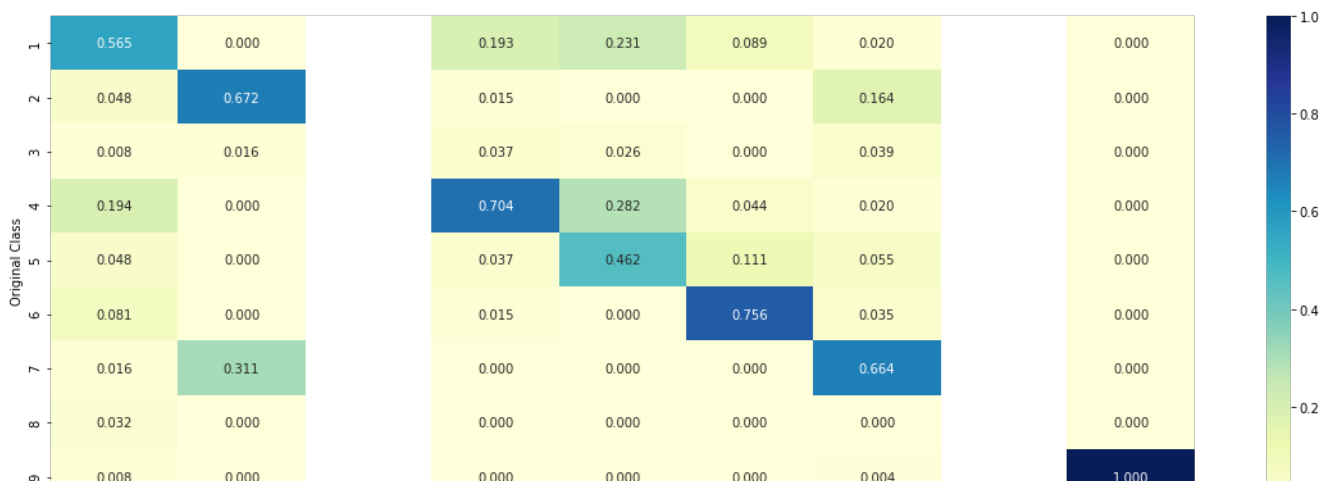
print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_onehotCoding)-test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

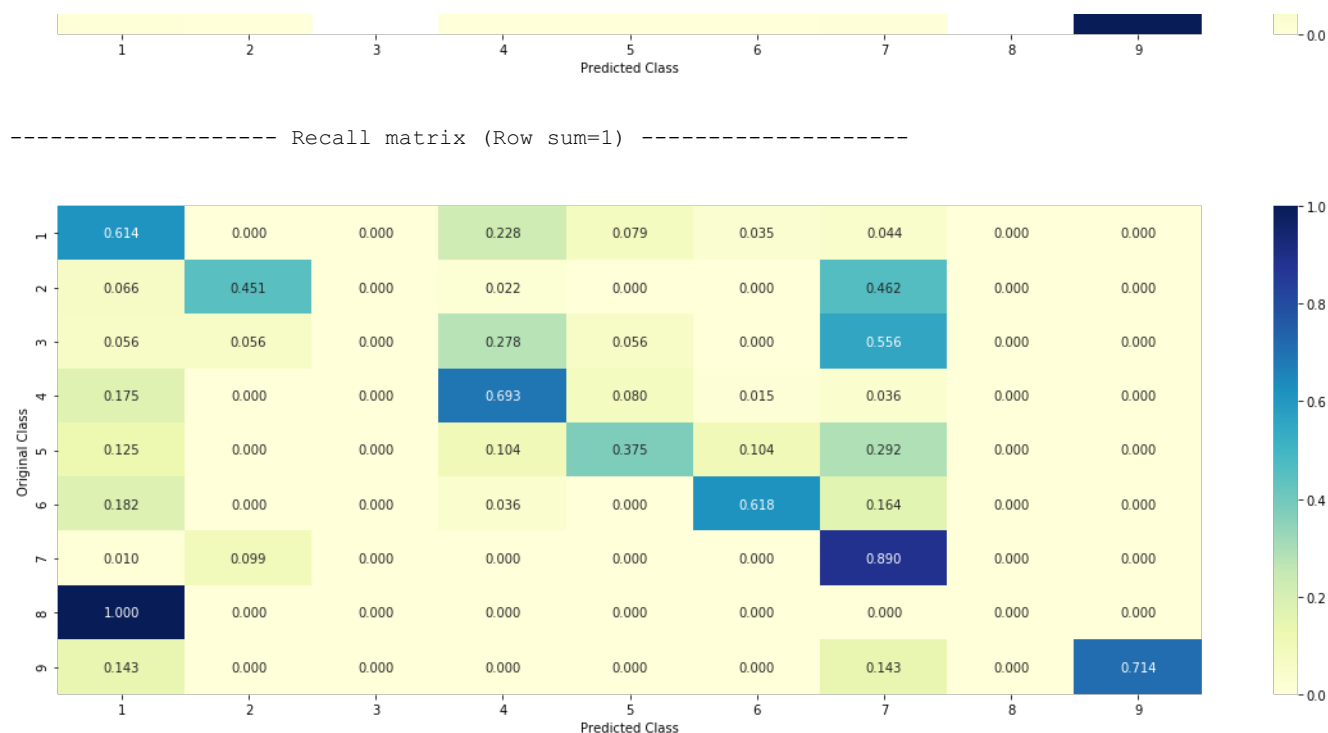
Log loss (train) on the stacking classifier : 0.5538016833228964
Log loss (CV) on the stacking classifier : 1.1809904390953867
Log loss (test) on the stacking classifier : 1.1078029125346778
Number of missclassified point : 0.34887218045112783

----- Confusion matrix -----



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





4.7.3 Maximum Voting classifier

In [98]:

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

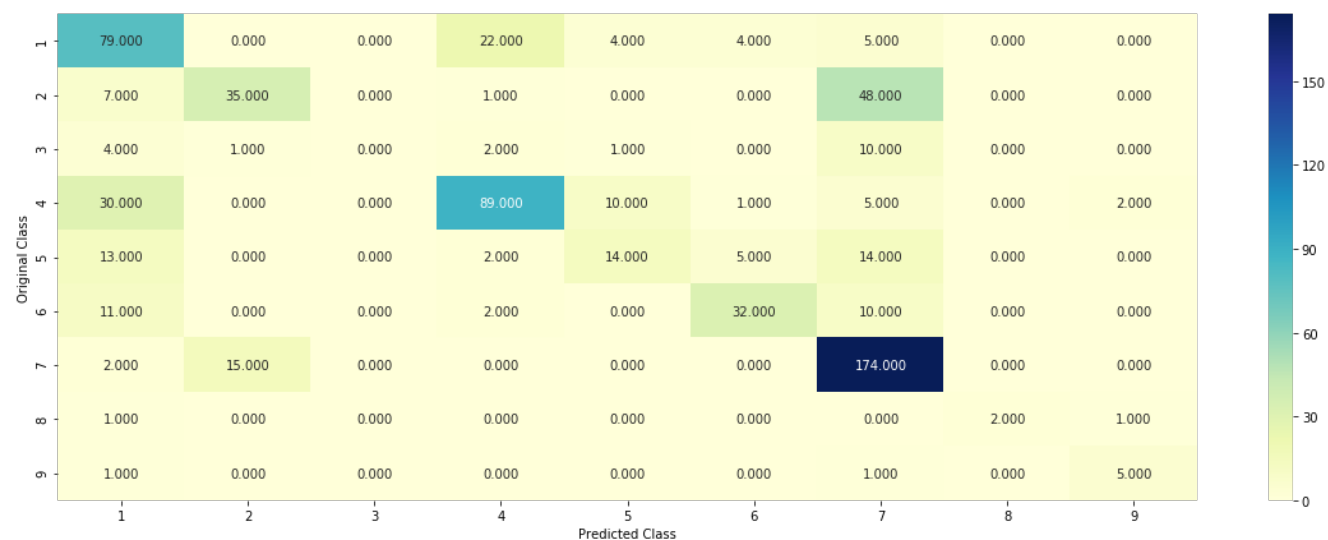
Log loss (train) on the VotingClassifier : 0.8477704570845488

Log loss (CV) on the VotingClassifier : 1.2127729835135057

Log loss (test) on the VotingClassifier : 1.154683840083712

Number of missclassified point : 0.3533834586466165

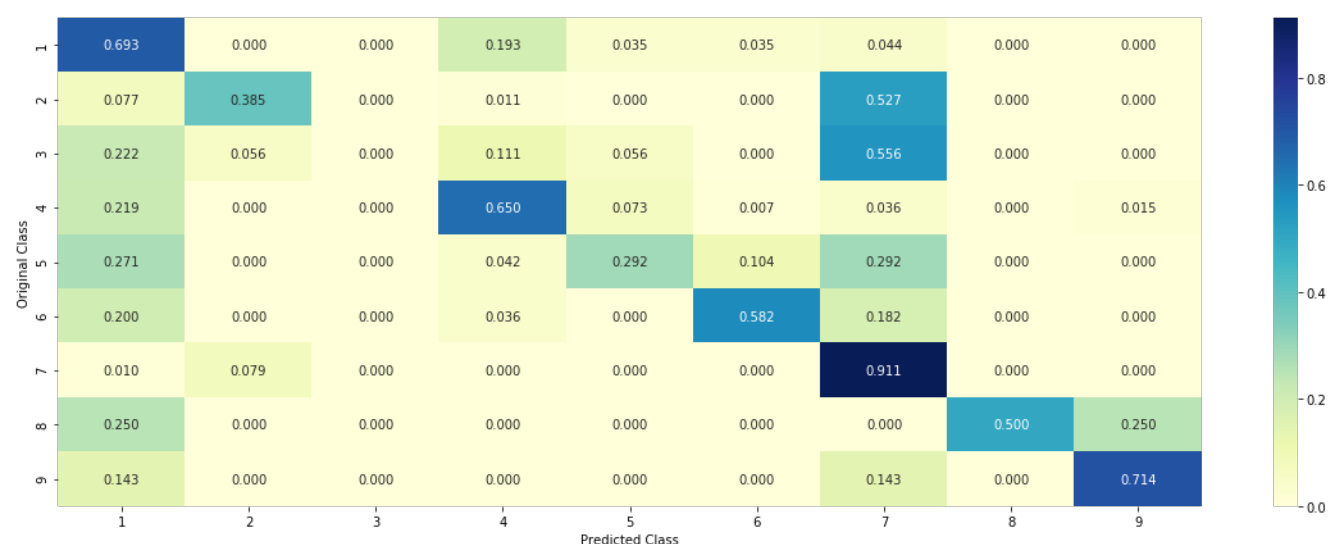
----- Confusion matrix -----



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



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



5. Assignments

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

Apply Logistic regression with CountVectorizer Features, including both1 unigrams and bigrams

In [99]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer_lr = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding_lr = text_vectorizer_lr.fit_transform(train_df['TEXT'])
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding_lr, axis=0)
cv_text_feature_onehotCoding_lr = text_vectorizer_lr.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding_lr, axis=0)
test_text_feature_onehotCoding_lr = text_vectorizer_lr.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding_lr, axis=0)
# getting all the feature names (words)
```



```

train_text_features_lr= text_vectorizer_lr.get_feature_names()
# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
train_text_fea_counts_lr = train_text_feature_onehotCoding_lr.sum(axis=0).A1
# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occurred
text_fea_dict_lr = dict(zip(list(train_text_features_lr),train_text_fea_counts_lr))
print("Total number of unique words in train data :", len(train_text_features_lr))

```

Total number of unique words in train data : 760779

In [0]:

```

# one-hot encoding of Gene feature.
gene_vectorizer_g = CountVectorizer()
train_gene_feature_onehotCoding_g = gene_vectorizer_g.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_g = gene_vectorizer_g.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_g = gene_vectorizer_g.transform(cv_df['Gene'])

```

In [101]:

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

Out[101]:

```

793      ERBB4
1572      ALK
905      PDGFRA
1762      IDH1
2194      PTEN
Name: Gene, dtype: object

```

In [102]:

```
print("shape of train: ",train_gene_feature_onehotCoding_g .shape)
```

shape of train: (2124, 238)

In [0]:

```

# one-hot encoding of variation feature.
variation_vectorizer_g = CountVectorizer()
train_variation_feature_onehotCoding_g = variation_vectorizer_g.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_g = variation_vectorizer_g.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_g = variation_vectorizer_g.transform(cv_df['Variation'])

```

In [104]:

```
print("shape of train: ",train_variation_feature_onehotCoding_g .shape)
```

shape of train: (2124, 1959)

In [0]:

```

train_gene_var_onehotCoding_g=hstack((train_gene_feature_onehotCoding_g,train_variation_feature_onehotCoding_g))
test_gene_var_onehotCoding_g
=hstack((test_gene_feature_onehotCoding_g,test_variation_feature_onehotCoding_g))
cv_gene_var_onehotCoding_g
=hstack((cv_gene_feature_onehotCoding_g,cv_variation_feature_onehotCoding_g))

```

In [0]:

```

train_x_onehotCoding_grams = hstack((train_gene_var_onehotCoding_g,
train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))
test_x_onehotCoding_grams = hstack((test_gene_var_onehotCoding_g, test_text_feature_onehotCoding)).tocsr()

```

```
test_y = np.array(list(test_df['Class']))
cv_x_onehotCoding_grams = hstack((cv_gene_var_onehotCoding_g, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [107]:

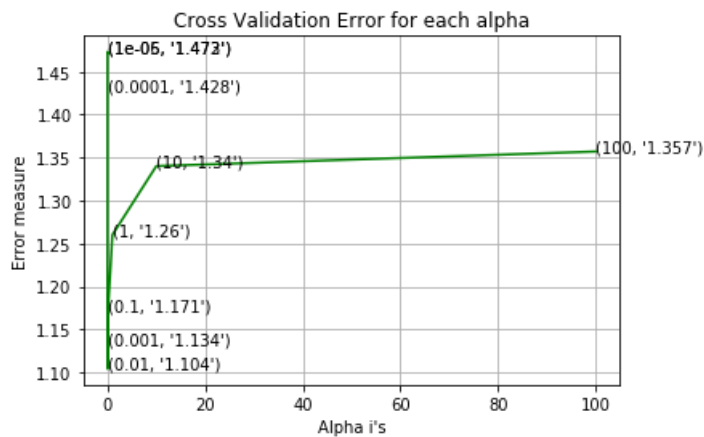
```
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-----
# find more about CalibratedClassifierCV here at
http://scikitlearn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-----
# video link:
#-----
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding_grams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_grams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_grams)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilities we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_grams, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_grams, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding_grams)
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_onehotCoding_grams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_grams)
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 : 1.4717859759937055
for alpha = 1e-05
Log Loss : 1.4731957586237434
for alpha = 0.0001
```

```

for alpha = 0.0001
Log Loss : 1.4276585834091153
for alpha = 0.001
Log Loss : 1.133614306625563
for alpha = 0.01
Log Loss : 1.104219291122409
for alpha = 0.1
Log Loss : 1.1714893767901464
for alpha = 1
Log Loss : 1.2600880279083857
for alpha = 10
Log Loss : 1.340143444429386
for alpha = 100
Log Loss : 1.3570774829101169

```



For values of best alpha = 0.01 The train log loss is: 0.704499911231661
 For values of best alpha = 0.01 The cross validation log loss is: 1.104219291122409
 For values of best alpha = 0.01 The test log loss is: 1.1325079445227053

Testing the model with best hyper paramters

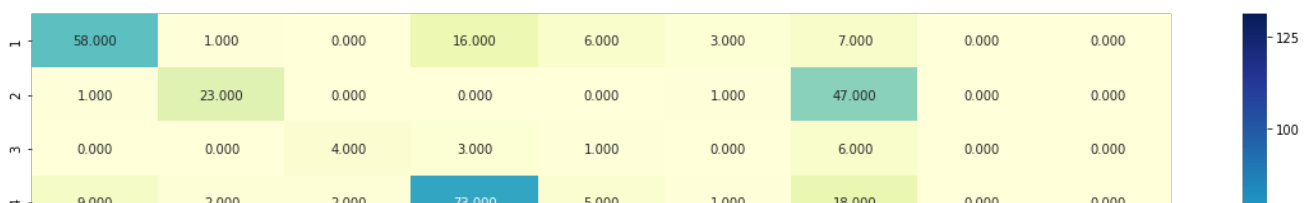
In [108]:

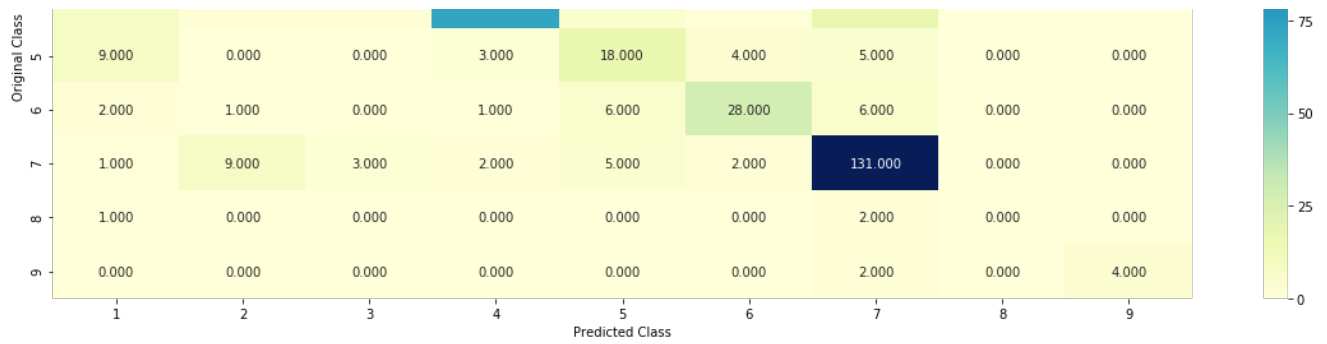
```

# read more about SGDClassifier() at
http://scikitlearn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
# -----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_grams, train_y, cv_x_onehotCoding_grams, cv_y, clf)

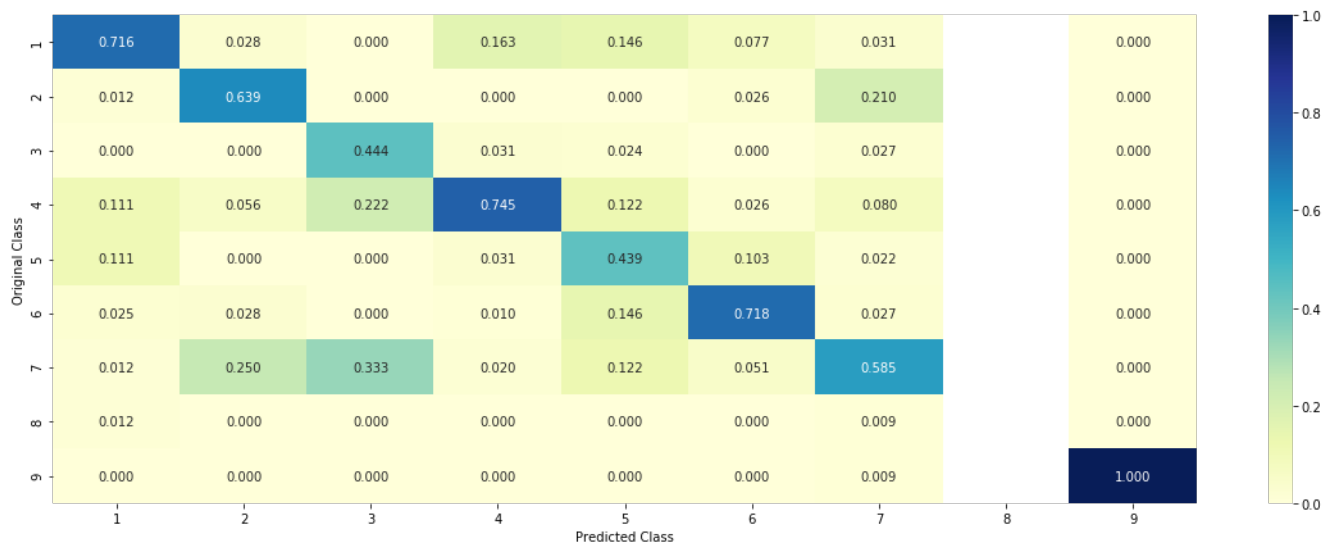
```

Log loss : 1.104219291122409
 Number of mis-classified points : 0.36278195488721804
 ----- Confusion matrix -----

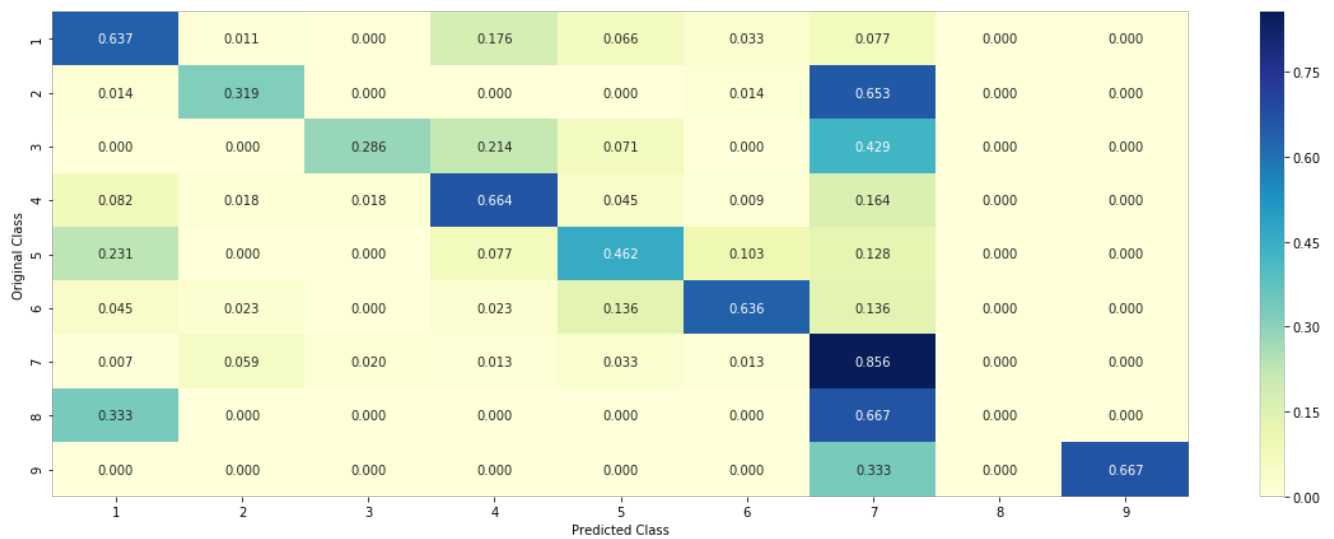




----- Precision matrix (Column Sum=1) -----



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



I used balanced parameter because it performs better than unigram and bigram

4.Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

what if we try to combine both gene and variance

In [0]:

```
gene_variation = []
for gene in result['Gene'].values:
    gene_variation.append(gene)
```

```
for variation in result['Variation'].values:
    gene_variation.append(variation)
```

In [110]:

```
gene_variation[0:10]
```

Out[110]:

```
['FAM58A', 'CBL', 'CBL', 'CBL', 'CBL', 'CBL', 'CBL', 'CBL', 'CBL', 'CBL']
```

Feature Importance

In [0]:

```
tfidfVectorizer = TfidfVectorizer(min_df=5,max_features=3000,ngram_range=(1,4))
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()
train_text = tfidfVectorizer.transform(train_df['TEXT'])
test_text = tfidfVectorizer.transform(test_df['TEXT'])
cv_text = tfidfVectorizer.transform(cv_df['TEXT'])
```

Correctly Classified point

In [124]:

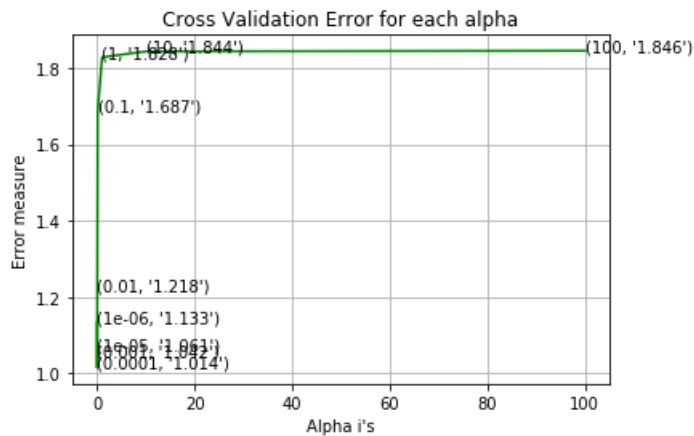
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilities we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1332607894229252
for alpha = 1e-05
Log Loss : 1.0610327461109685
for alpha = 0.0001
Log Loss : 1.0138413983629366
for alpha = 0.001
```

```

Log Loss : 1.042474659076955
for alpha = 0.01
Log Loss : 1.217843678634886
for alpha = 0.1
Log Loss : 1.6868047353360354
for alpha = 1
Log Loss : 1.8284832711964811
for alpha = 10
Log Loss : 1.843791685699172
for alpha = 100
Log Loss : 1.8455839386011739

```



```

For values of best alpha = 0.0001 The train log loss is: 0.45786829431688403
For values of best alpha = 0.0001 The cross validation log loss is: 1.0138413983629366
For values of best alpha = 0.0001 The test log loss is: 0.9218869949078555

```

In [113]:

```

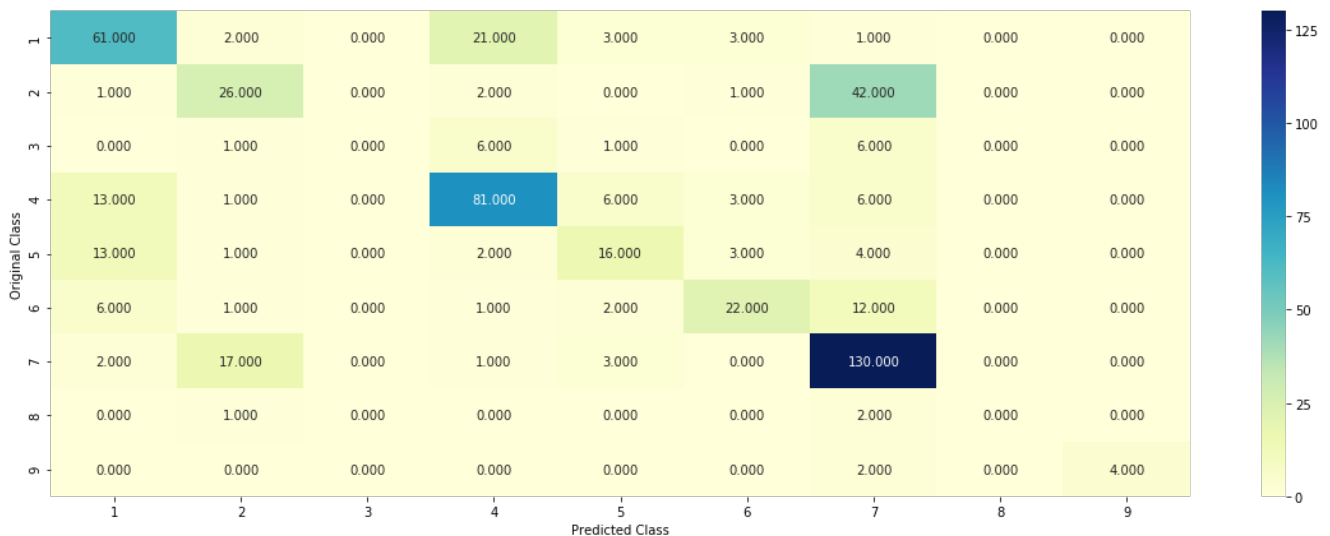
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

```

Log loss : 1.0138413983629366
Number of mis-classified points : 0.3609022556390977
----- Confusion matrix -----

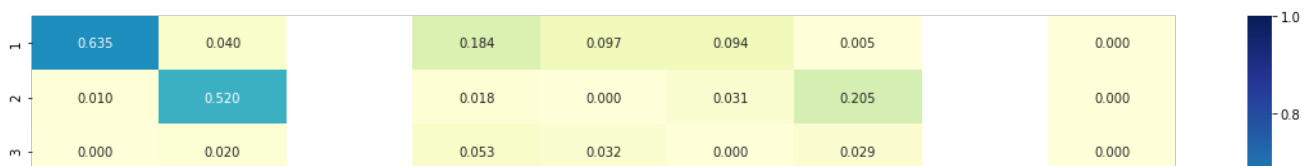
```

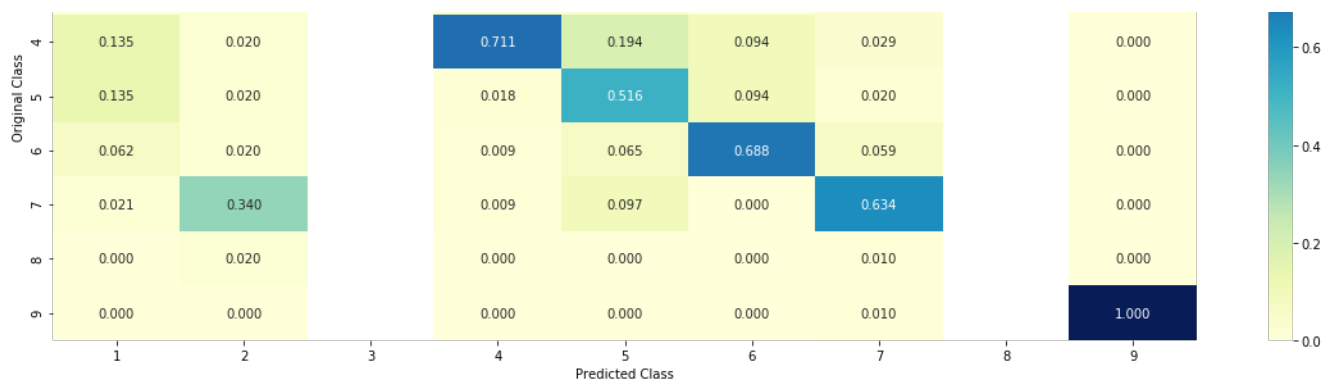


```

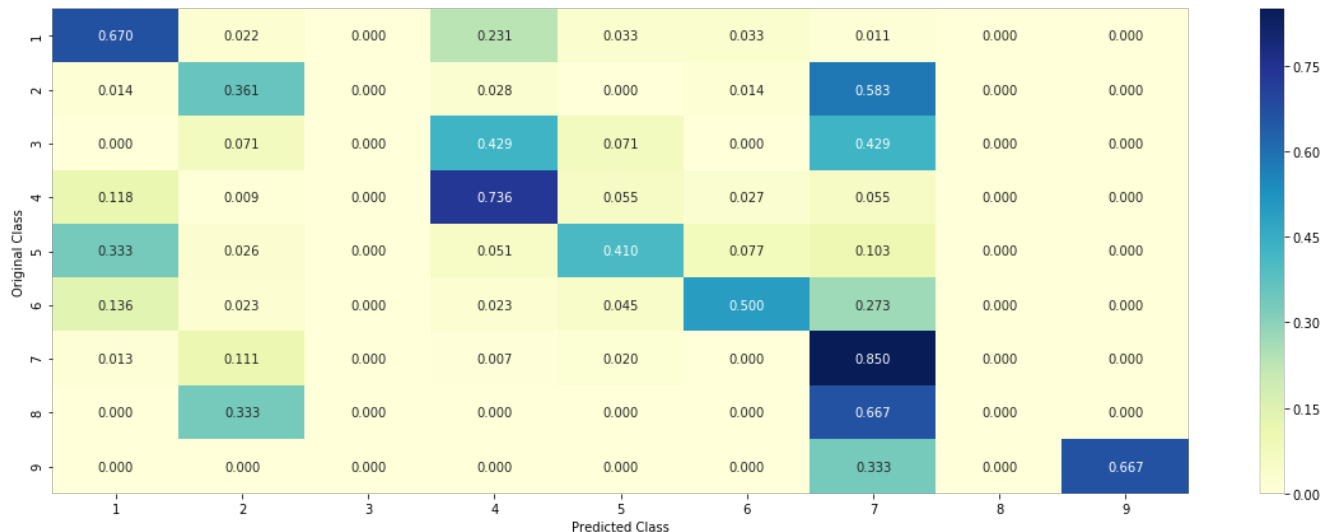
----- Precision matrix (Column Sum=1) -----

```





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



Logistic regrssion without class imbalancing

In [114]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```

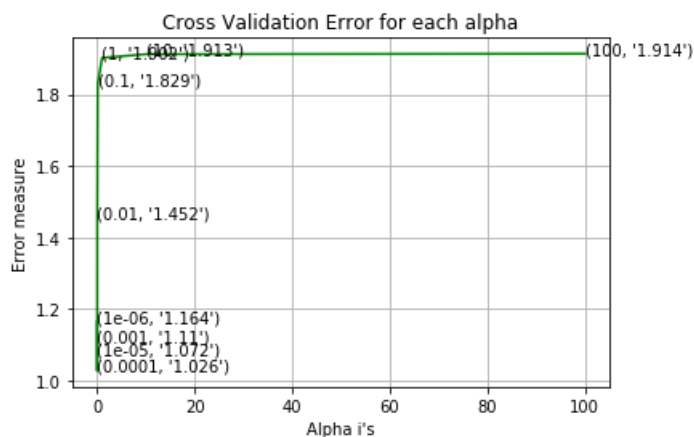
print('For values of best alpha = ', alpha[best_alpha], 'The cross validation log loss is:', log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], 'The test log loss is:', log_loss(y_test,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

```

```

for alpha = 1e-06
Log Loss : 1.1644814258435812
for alpha = 1e-05
Log Loss : 1.0715194644434234
for alpha = 0.0001
Log Loss : 1.0263873522848197
for alpha = 0.001
Log Loss : 1.1099980263917337
for alpha = 0.01
Log Loss : 1.4521864672252107
for alpha = 0.1
Log Loss : 1.8289766949862687
for alpha = 1
Log Loss : 1.9024470396998319
for alpha = 10
Log Loss : 1.9128041103133233
for alpha = 100
Log Loss : 1.9140498222870896

```



```

For values of best alpha = 0.0001 The train log loss is: 0.45786829431688403
For values of best alpha = 0.0001 The cross validation log loss is: 1.0138413983629366
For values of best alpha = 0.0001 The test log loss is: 0.9218869949078555

```

In [115]:

```

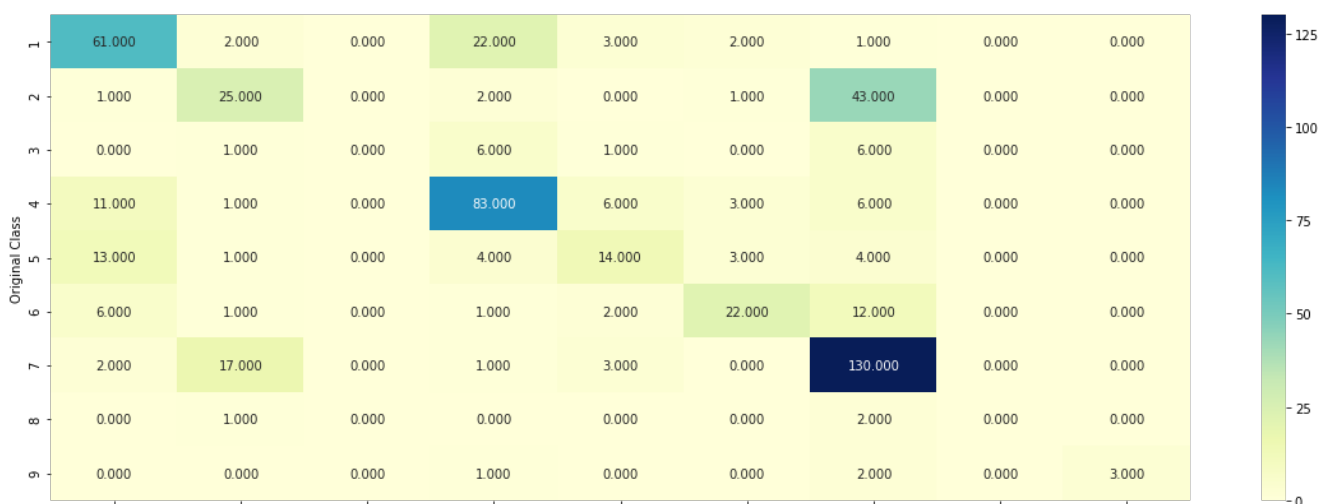
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

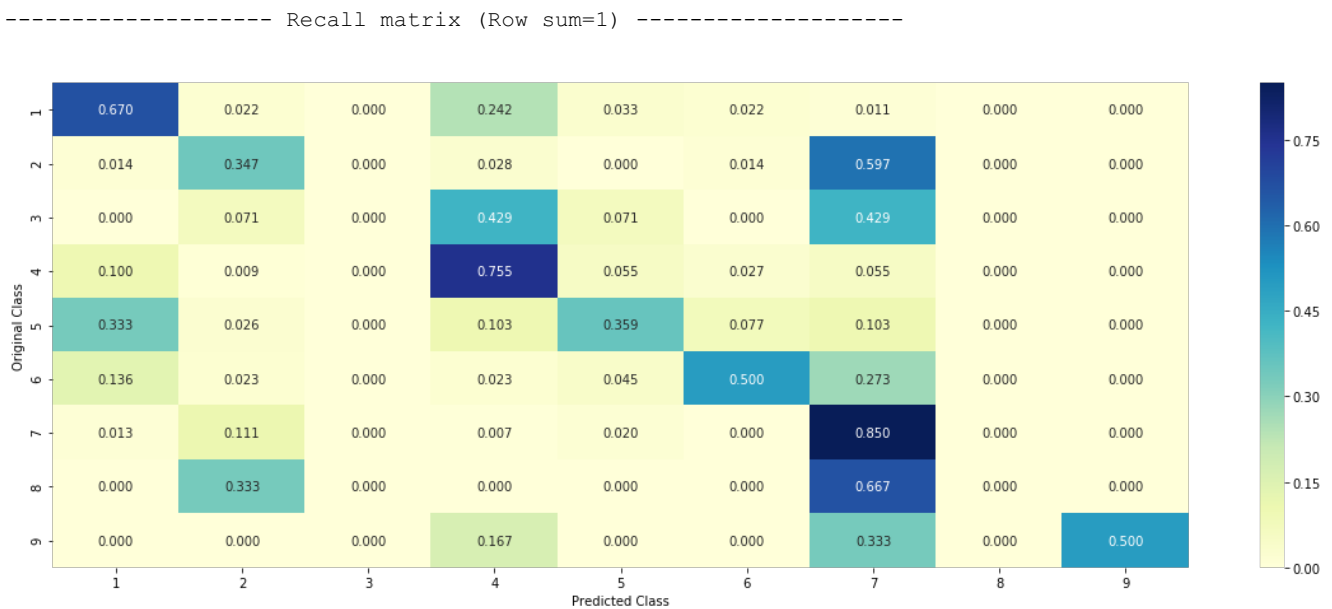
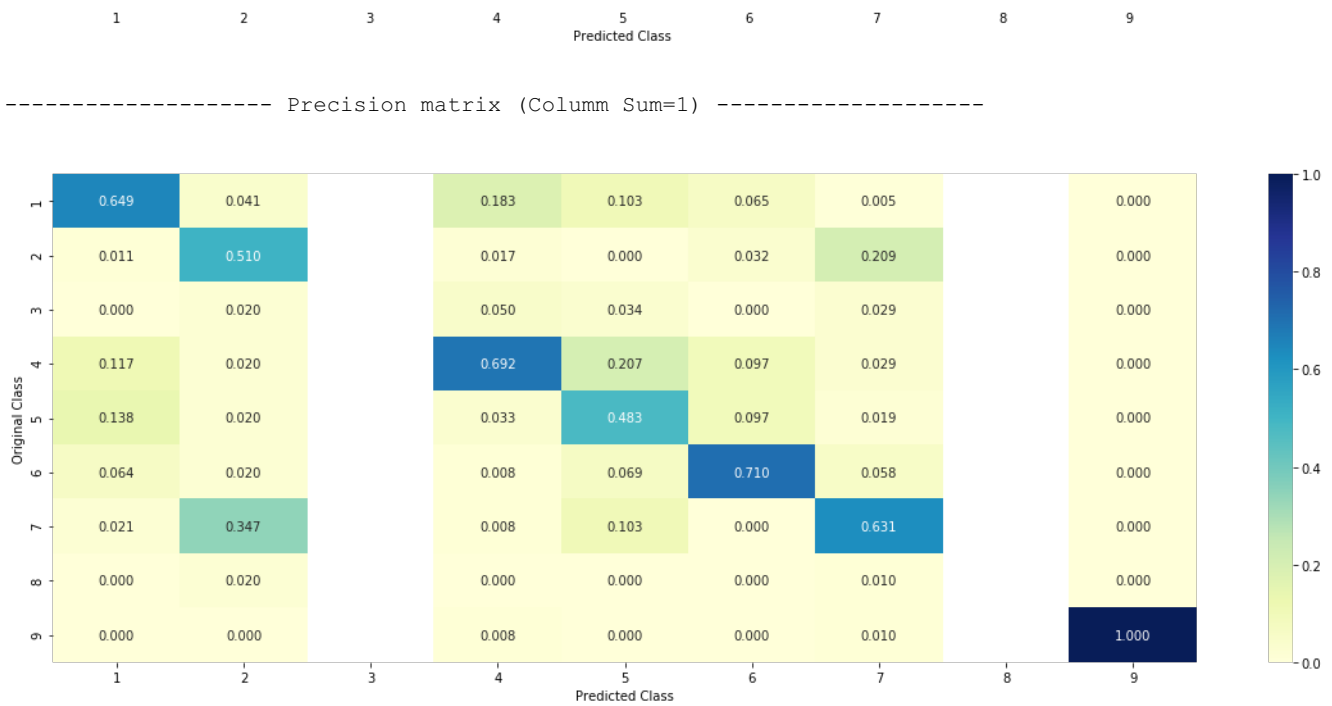
```

```

Log loss : 1.0263873522848197
Number of mis-classified points : 0.36466165413533835
----- Confusion matrix -----

```





In [130]:

```
ptable = PrettyTable()
ptable.field_names=["Model Name","Train","CV","Test","% Misclassified Points"]
ptable.add_row(["Logistic Regression With Class balancing","0.45","1.0","0.92","35"])
ptable.add_row(["Logistic Regression With Class balancing","0.44","1.0","0.92","34"])
print(ptable)
```

Model Name	Train	CV	Test	% Misclassified Points
Logistic Regression With Class balancing	0.45	1.0	0.92	35
Logistic Regression With Class balancing	0.44	1.0	0.92	34