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=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import 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 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 [2]:

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

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

Out[2]:

	ID	Gene	Variation	Class

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

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

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

3.1.2. Reading Text Data

In [3]:

```
# note the 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[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

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

```
                string += word + " "

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 1037.3273577580303 seconds
```

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

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

In [8]:

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

In [9]:

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

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

In [11]:

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

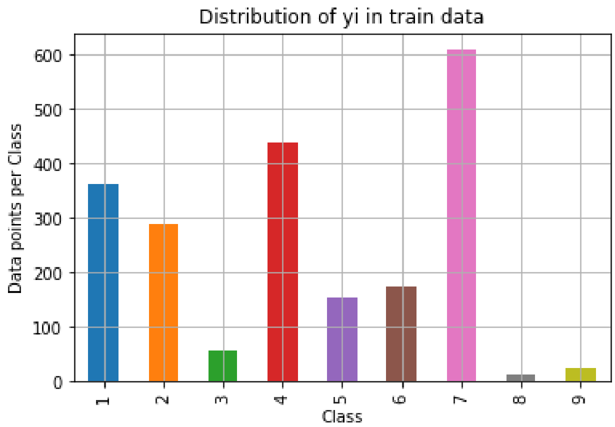
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

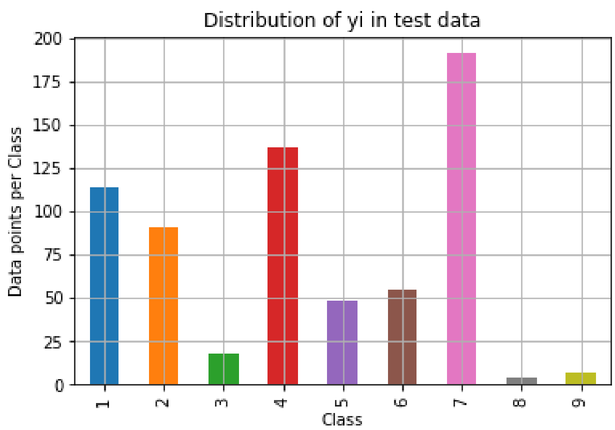
In [12]:

```
# it returns a dict, keys as class labels and values as the number of data 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 yi in train data')
plt.grid()
plt.show()

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


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

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

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

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

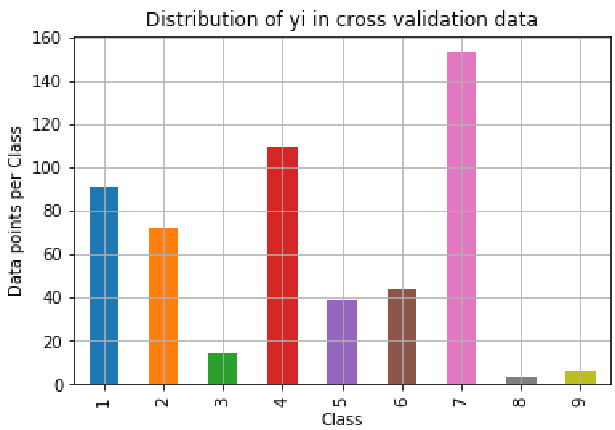



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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




```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
```

```
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
--------------------------------------------------------------------------------
```




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

3.2 Prediction using a 'Random' Model

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

In [13]:

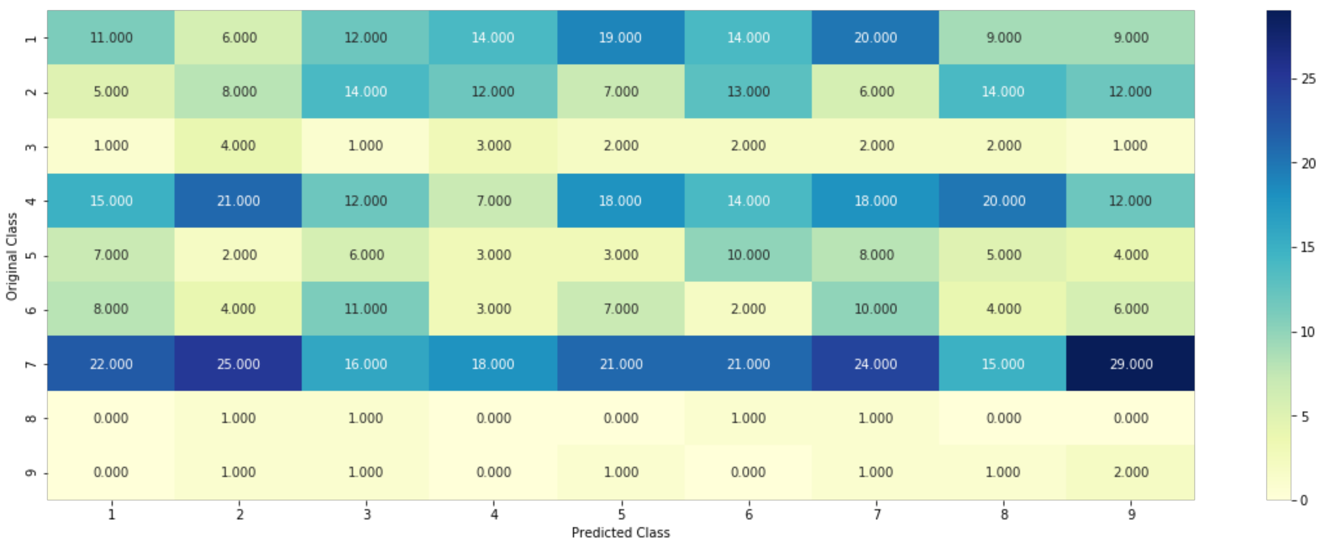
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
```

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

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

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

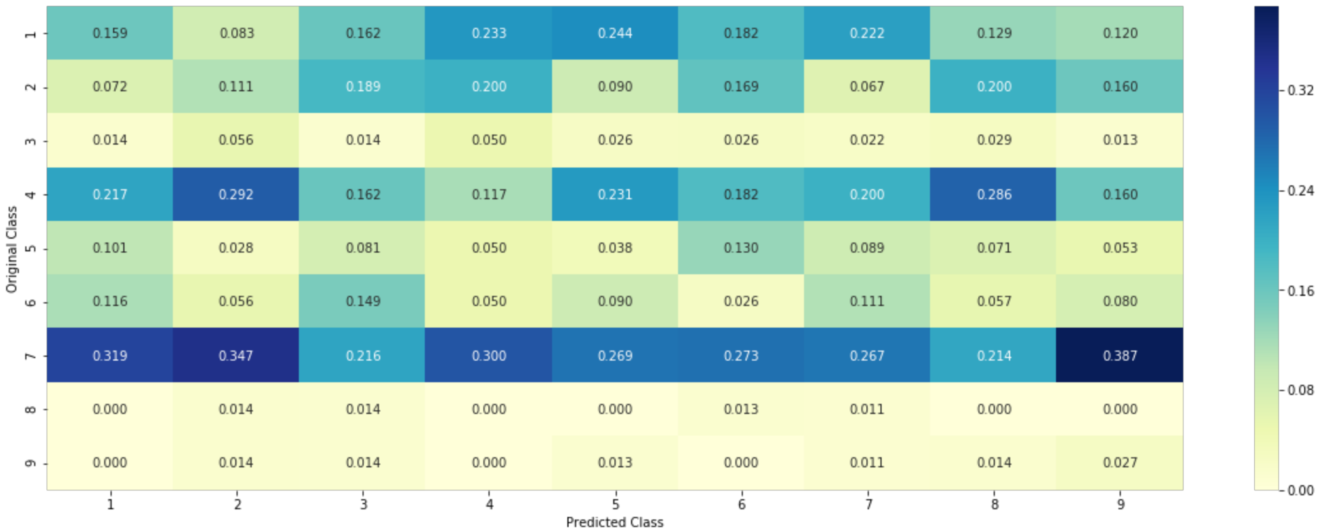
In [14]:

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

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


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

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

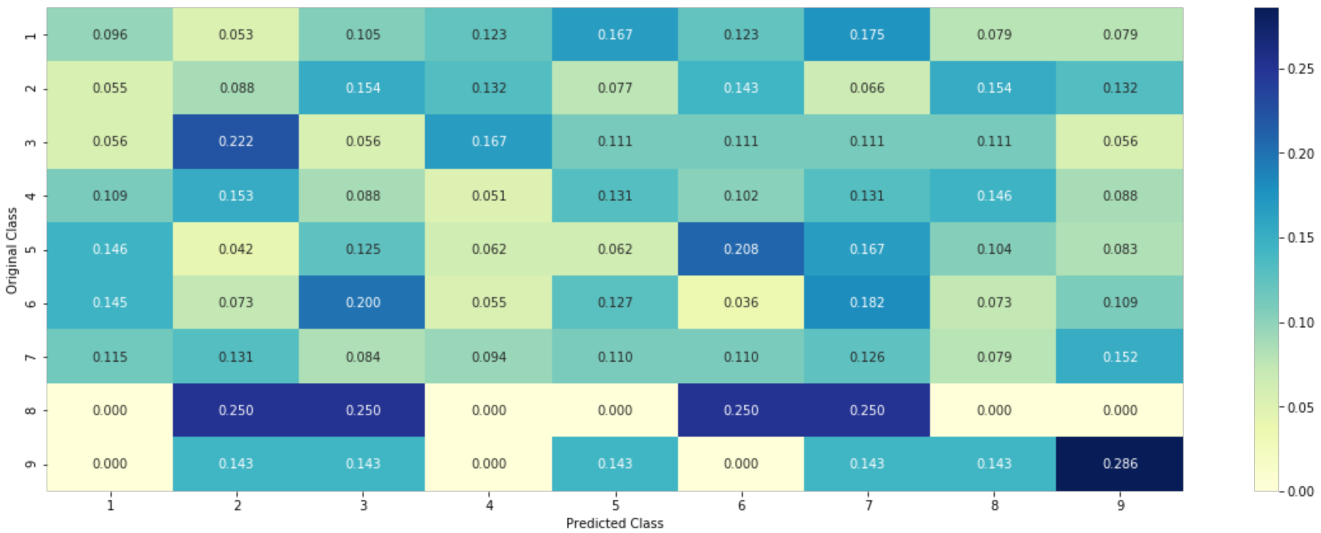
```
Log loss on Cross Validation Data using Random Model 2.411364671909254
Log loss on Test Data using Random Model 2.487011870936331
-------------------- Confusion matrix --------------------
```



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




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



3.3 Univariate Analysis

In [15]:

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

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
```

```
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
```

```
#       'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #       ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:

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

```
Number of Unique Genes : 236
BRCA1     171
TP53      105
EGFR       94
BRCA2      82
PTEN       74
BRAF       59
KIT        57
ALK        48
ERBB2      43
PIK3CA     41
Name: Gene, dtype: int64
```

In [17]:

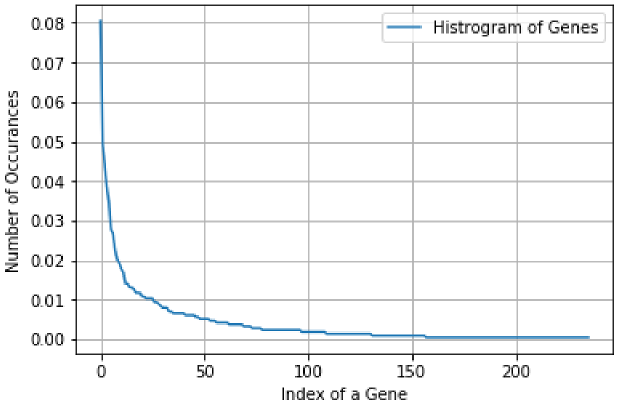
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
d they are distibuted as follows",)
```

```
Ans: There are 236 different categories of genes in the train data, and they are distibuted as fol
lows
```

In [18]:

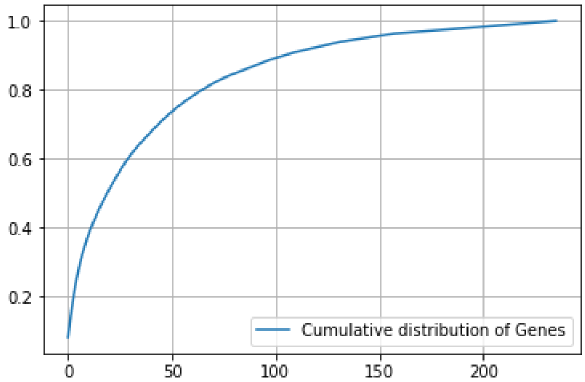
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
```

```
plt.grid()
plt.show()
```



In [19]:

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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:

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

In [21]:

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

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

In [22]:

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

In [23]:

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

Out[23]:

```
754      ERBB2
1443      SPOP
2353     AURKA
2937       BTK
1769      IDH2
Name: Gene, dtype: object
```

In [24]:

```python
gene_vectorizer.get_feature_names()
```

Out[24]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
```

```
    'cdk6',
    'cdkn1a',
    'cdkn1b',
    'cdkn2a',
    'cdkn2b',
    'cebpa',
    'chek2',
    'cic',
    'crebbp',
    'ctcf',
    'ctla4',
    'ctnnb1',
    'ddr2',
    'dicer1',
    'dnmt3a',
    'dnmt3b',
    'egfr',
    'eif1ax',
    'elf3',
    'ep300',
    'epas1',
    'epcam',
    'erbb2',
    'erbb3',
    'erbb4',
    'ercc2',
    'ercc3',
    'ercc4',
    'erg',
    'errfi1',
    'esr1',
    'etv1',
    'etv6',
    'ewsr1',
    'ezh2',
    'fanca',
    'fat1',
    'fbxw7',
    'fgf19',
    'fgf3',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'flt1',
    'flt3',
    'foxa1',
    'foxl2',
    'foxo1',
    'foxp1',
    'fubp1',
    'gata3',
    'gnaq',
    'gnas',
    'h3f3a',
    'hla',
    'hnf1a',
    'hras',
    'idh1',
    'idh2',
    'igf1r',
    'ikbke',
    'ikzf1',
    'inpp4b',
    'jak1',
    'jak2',
    'jun',
    'kdm5a',
    'kdm5c',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'klf4',
    'kmt2a',
    'kmt2c',
    'kmt2d',
    'knstrn',
```

```
    'kras',
    'lats1',
    'lats2',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mdm4',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mga',
    'mlh1',
    'mpl',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myd88',
    'myod1',
    'ncor1',
    'nf1',
    'nf2',
    'nfe2l2',
    'nfkbia',
    'nkx2',
    'notch1',
    'notch2',
    'npm1',
    'nras',
    'nsd1',
    'ntrk1',
    'ntrk2',
    'ntrk3',
    'nup93',
    'pak1',
    'pax8',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
    'pik3r3',
    'pim1',
    'pms1',
    'pms2',
    'pole',
    'ppm1d',
    'ppp2r1a',
    'ppp6c',
    'prdm1',
    'ptch1',
    'pten',
    'ptpn11',
    'ptprd',
    'ptprt',
    'rab35',
    'rac1',
    'rad21',
    'rad50',
    'rad51b',
    'rad51c',
    'rad54l',
    'raf1',
    'rara',
    'rasa1',
    'rb1',
    'rbm10',
    'ret',
    'rheb',
    'rhoa',
    'rictor',
    'rit1',
```

```
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xrcc2',
 'yap1']
```

In [25]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 236)
```

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

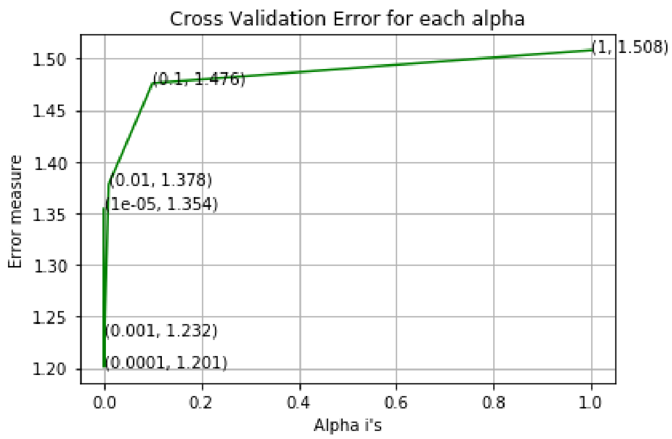
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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:
#------------------------------


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

```
For values of alpha =  1e-05 The log loss is: 1.3543979410776887
For values of alpha =  0.0001 The log loss is: 1.2009360230078854
For values of alpha =  0.001 The log loss is: 1.232060808264493
For values of alpha =  0.01 The log loss is: 1.3779864840543716
For values of alpha =  0.1 The log loss is: 1.4759331708816559
For values of alpha =  1 The log loss is: 1.5078842555095349
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0446178147460472
For values of best alpha =  0.0001 The cross validation log loss is: 1.2009360230078854
For values of best alpha =  0.0001 The test log loss is: 1.235313858284605
```

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

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

In [27]:

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

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(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.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  236  genes in train dataset?
Ans
1. In test data 646 out of 665 : 97.14285714285714
2. In cross validation data 516 out of  532 : 96.99248120300751
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [28]:

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

```
Number of Unique Variations : 1933
Truncating_Mutations    61
Deletion                46
Amplification           44
Fusions                 23
Q61R                     3
G12V                     3
E17K                     3
E330K                    2
Q22K                     2
G12C                     2
Name: Variation, dtype: int64
```

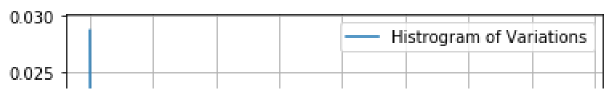
In [29]:

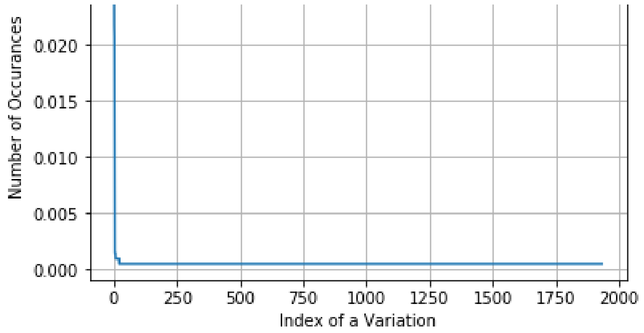
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1933 different categories of variations in the train data, and they are distibuted
as follows
```

In [30]:

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

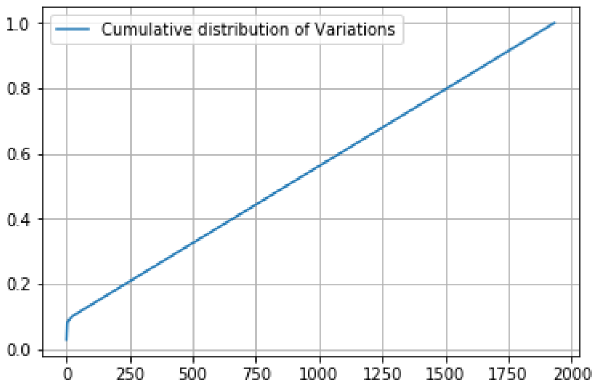




In [31]:

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

```
[0.0287194  0.05037665 0.07109228 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:

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

In [33]:

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

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

In [34]:

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

In [35]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1965)
```

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

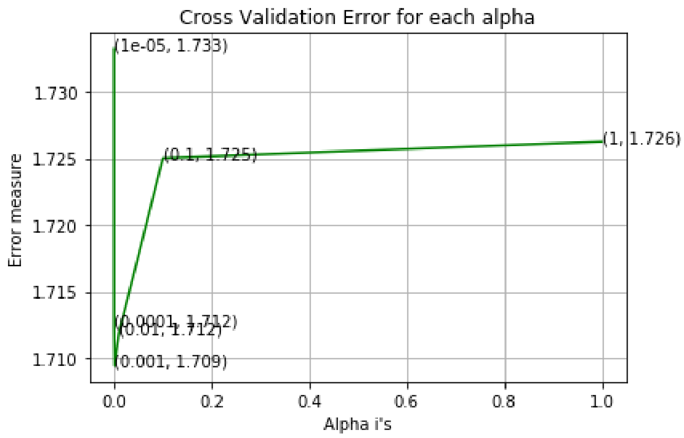
Let's build a model just like the earlier!

In [36]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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:
#------------------------------


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.clas
ses_, 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_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7332486696654548
For values of alpha =  0.0001 The log loss is: 1.7124014702453267
For values of alpha =  0.001 The log loss is: 1.7094048827168797
For values of alpha =  0.01 The log loss is: 1.7118632793639137
For values of alpha =  0.1 The log loss is: 1.7250297733730298
For values of alpha =  1 The log loss is: 1.72625708575747
```




```
For values of best alpha =  0.001 The train log loss is: 1.046204318583874
For values of best alpha =  0.001 The cross validation log loss is: 1.7094048827168797
For values of best alpha =  0.001 The test log loss is: 1.7137069679487078
```

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

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

In [37]:

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

```
Q12. How many data points are covered by total  1933  genes in test and cross validation data
sets?
Ans
1. In test data 77 out of 665 : 11.578947368421053
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 predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

In [38]:

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

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

In [39]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(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 [41]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
#text_vectorizer = CountVectorizer(min_df=3)
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 featu
res) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

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

In [42]:

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

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


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

In [43]:

```
#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 [44]:

```
# 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 [45]:

```
# 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 [46]:

```
#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 [47]:

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

```
Counter({250.28782977814006: 1, 181.89913034638352: 1, 137.7375017011242: 1, 132.76986311245594: 1
, 131.4310072138832: 1, 117.50260224612069: 1, 117.23332775207926: 1, 115.5131481655906 5: 1,
110.81589671020464: 1, 110.1837652119254: 1, 106.77771524549122: 1, 90.48043038114845: 1,
88.72155780500756: 1, 88.4827590566821: 1, 82.54008112558063: 1, 80.83494629642914: 1,
80.03492581527614: 1, 79.46220373212238: 1, 78.96626494307284: 1, 77.20691302108429: 1,
76.59252713537408: 1, 75.02687334083132: 1, 71.19932562611334: 1, 70.96204265328538: 1,
68.15296600480002: 1, 67.92827045651899: 1, 67.48438824441438: 1, 66.80701240459162: 1,
64.31277646295291: 1, 64.04285815293417: 1, 64.01420438954314: 1, 63.87090808388448: 1,
63.62973025695855: 1, 60.145801618730104: 1, 60.055018551940044: 1, 58.555397986298665: 1,
57.28289473278494: 1, 57.00564083384034: 1, 54.43084527380317: 1, 52.23988658653901: 1,
51.966485363975245: 1, 51.193765578514935: 1, 50.844512362645304: 1, 49.55076652305236: 1,
49.45149108553814: 1, 47.74487619424509: 1, 47.13466332770438: 1, 46.7661703107039: 1,
45.57058176601787: 1, 44.38123816818661: 1, 44.14228668890134: 1, 43.6071178134211: 1,
43.52754118831114: 1, 43.25981006833426: 1, 43.20452002346774: 1, 43.19704969429656: 1,
42.62082932715437: 1, 42.5761574508931: 1, 42.495189539793806: 1, 42.39286720447528: 1,
42.36189115303191: 1, 42.17296932849946: 1, 41.53117320542898: 1, 41.35789720017843: 1,
40.49969902564977: 1, 40.33366718240532: 1, 40.28163765355951: 1, 40.111962844784586: 1,
40.07132725112731: 1, 39.8662261502998: 1, 39.23432622523735: 1, 39.027084613519 5: 1,
38.35123368952777: 1, 37.642606993640015: 1, 36.819059128586545: 1, 36.391064191949795: 1,
36.34114861608737: 1, 36.19610042705511: 1, 36.17987000567006: 1, 36.06010644201681: 1,
35.8961697965971: 1, 35.865997605056094: 1, 35.57491893696375: 1, 35.548344926318876: 1,
35.487305939250476: 1, 34.800903933326616: 1, 34.61547976514832: 1, 34.2145129695290 7: 1,
33.82232670608538 4: 1, 33.765135594695145: 1, 33.48421085880734: 1, 33.46312653019187: 1,
33.31639810366531: 1, 33.050108261983745: 1, 33.04478123672333: 1, 32.88974610168398: 1,
32.84771347960146: 1, 32.6387717979967: 1, 32.27699089530106: 1, 32.23194755599303: 1,
32.19714535505097: 1, 32.16300411949033: 1, 32.14455456949364: 1, 31.991942561342583: 1,
31.96978931615793: 1, 31.87595181198467: 1, 31.68204687909445: 1, 31.55273812592529: 1,
```

31.509841884281105: 1, 31.160316555537097: 1, 31.155374148001684: 1, 30.91323157739808: 1,
30.666613180111383: 1, 30.659817236712357: 1, 30.638001089201847: 1, 30.602111688468913: 1,
30.141648736332787: 1, 30.13503195588182: 1, 30.09682474699029: 1, 29.958225536432394: 1,
29.92315042578296: 1, 29.466443103694527: 1, 29.388451923293573: 1, 29.336531266477923: 1,
29.221138335777585: 1, 28.855294693700067: 1, 28.73078054999083: 1, 28.411851154050495: 1,
28.401681341880288: 1, 28.371115011457995: 1, 28.317938857256507: 1, 27.936251153671638: 1,
27.83413760239202: 1, 27.749576021283175: 1, 27.65984080827898: 1, 27.501714799182924: 1,
27.468623979936417: 1, 27.468424041545923: 1, 27.43370040949668: 1, 27.280940848559787: 1,
27.104904767117564: 1, 26.931803686247644: 1, 26.91281768887538: 1, 26.844974517269907: 1,
26.74505813143393: 1, 26.699005680460495: 1, 26.357862519404758: 1, 26.162854518398625: 1,
25.914786117285374: 1, 25.894434072246394: 1, 25.712370130107676: 1, 25.58352375087898: 1,
25.56766376381119: 1, 25.565440314441354: 1, 25.457639702120133: 1, 25.44370269488574: 1,
25.386947257399296: 1, 25.314773403040554: 1, 25.210350096028332: 1, 25.208971532414797: 1,
25.170070016525745: 1, 25.047605795229202: 1, 25.047369618734304: 1, 25.002216257255355: 1,
24.97227062584625: 1, 24.895957744540997: 1, 24.789357006810206: 1, 24.762858227424154: 1,
24.727495976097217: 1, 24.60157218622088: 1, 24.33232268684407: 1, 24.32054016818028: 1, 24.3027276
01951904: 1, 24.192542038751288: 1, 24.135845060359458: 1, 24.103287661706624: 1,
24.0565055673654: 1, 23.94279308061528: 1, 23.913294382001407: 1, 23.880043743835348: 1,
23.776175047708207: 1, 23.538538567382503: 1, 23.42414045664511: 1, 23.383230375223206: 1,
23.290577391288384: 1, 23.240318817058995: 1, 23.234098218581053: 1, 23.18800010741215: 1,
23.172135532980274: 1, 23.1029994281379: 1, 23.05299101019056: 1, 23.05094286323644: 1,
23.037439973626903: 1, 23.034515121220597: 1, 22.942262032661286: 1, 22.856843013142615: 1,
22.8044623754955: 1, 22.746068124712917: 1, 22.658481015715356: 1, 22.39000591188257: 1,
22.26706014891157: 1, 22.20710488982929: 1, 22.20469708902197: 1, 22.202674784208515: 1,
22.19995675530534: 1, 22.05892487517508: 1, 22.05054371544582: 1, 22.02400523148353: 1,
21.827007919310272: 1, 21.80498863326187: 1, 21.757846185599657: 1, 21.73080349339036: 1,
21.72592764798742: 1, 21.6731819848318: 1, 21.670814835198875: 1, 21.634696702515523: 1,
21.556554324197922: 1, 21.54666338393522: 1, 21.53458307488921: 1, 21.438924916766283: 1,
21.422704242945024: 1, 21.392836956878387: 1, 21.3854195464647: 1, 21.351739915405357: 1,
21.33842630109267: 1, 21.27129754532339: 1, 21.254035521921505: 1, 21.2529164715983: 1,
21.15368743082888: 1, 21.115762204503543: 1, 21.08005566246209: 1, 21.061871626916805: 1,
21.05038364157323: 1, 20.60594391459304: 1, 20.603949392274654: 1, 20.57245686192224: 1,
20.501111140265763: 1, 20.499564200646397: 1, 20.461750505814106: 1, 20.41480089648129: 1,
20.348387972860955: 1, 20.346519161850182: 1, 20.305801676708786: 1, 20.232280308672642: 1,
20.215601547523985: 1, 20.145337122070487: 1, 20.092028334870513: 1, 20.069974826350276: 1,
19.997683425720002: 1, 19.97797106621733: 1, 19.976122656945897: 1, 19.888789051279677: 1,
19.8860353271846: 1, 19.870567871426093: 1, 19.700577005864677: 1, 19.693401289609827: 1,
19.60707495683224: 1, 19.59482781497228: 1, 19.58025618651953: 1, 19.55279663831962: 1,
19.506912418476926: 1, 19.500027627433198: 1, 19.4744935226755: 1, 19.470729772584363: 1,
19.461374882240218: 1, 19.399117965816526: 1, 19.39222368342594: 1, 19.3875488148936: 1,
19.34190003300807: 1, 19.31734860317294: 1, 19.31231027618785: 1, 19.249648668790662: 1,
19.215900726440456: 1, 19.212933868961436: 1, 19.093397496362908: 1, 19.024960092967902: 1,
19.018692124384525: 1, 19.001990039193505: 1, 18.96636649836016: 1, 18.91184040260117: 1,
18.892512014596782: 1, 18.867578261251023: 1, 18.805960982764056: 1, 18.78326578342225: 1,
18.780028129052848: 1, 18.70711805273936: 1, 18.67176205271083: 1, 18.66264963395748: 1, 18.6339195
0071768: 1, 18.55901458866407: 1, 18.494182530093475: 1, 18.48686580305785: 1, 18.475691857482914:
1, 18.398889485561273: 1, 18.36967897232435: 1, 18.3604956835302: 1, 18.132200515187154: 1,
18.0944593409387: 1, 18.061249066712527: 1, 18.044234592998105: 1, 18.00571969997361: 1,
17.973737310039677: 1, 17.96632992399162: 1, 17.93923934747344: 1, 17.90714982973129: 1, 17.8935519
29560193: 1, 17.880622006567048: 1, 17.859262368494043: 1, 17.85070066952413: 1,
17.836909038076797: 1, 17.82445589549348: 1, 17.819286742416754: 1, 17.809923401848135: 1,
17.80221326912358: 1, 17.698748808098404: 1, 17.68398328784442: 1, 17.670182024793007: 1,
17.640284194592734: 1, 17.59778437299893: 1, 17.59278643883626: 1, 17.5900075467978: 1,
17.546248992472663: 1, 17.51329091135764: 1, 17.511244105785188: 1, 17.46620053813244: 1,
17.46427640912374: 1, 17.447067899970552: 1, 17.432794368402714: 1, 17.427325199750765: 1,
17.39908322620921: 1, 17.364563745391763: 1, 17.251260602259062: 1, 17.250320042195003: 1,
17.244575382589073: 1, 17.19828012484854: 1, 17.188119306938724: 1, 17.182639666879602: 1,
17.13943509446894: 1, 17.093497728694626: 1, 17.093092080959945: 1, 17.08371426294351: 1,
17.037769625756937: 1, 17.017676709691454: 1, 17.002287636894167: 1, 16.99660990544699: 1,
16.931250198909627: 1, 16.924151085376074: 1, 16.85694314574931: 1, 16.839125877693682: 1,
16.82267285464906: 1, 16.807471202072403: 1, 16.700869037891515: 1, 16.693170207528553: 1,
16.66158651289093: 1, 16.653509772029466: 1, 16.65186942743953: 1, 16.640945637382046: 1,
16.60509917524801: 1, 16.603468946605087: 1, 16.5868983962249: 1, 16.584345230579192: 1,
16.583759983642555: 1, 16.581622715695524: 1, 16.541695422041233: 1, 16.530028225776544: 1,
16.441958545153735: 1, 16.349120976640233: 1, 16.31639041949751: 1, 16.285576048084998: 1,
16.205595558326017: 1, 16.180882173875506: 1, 16.152082424690263: 1, 16.067950374340484: 1,
16.039157663905744: 1, 16.00116448602588: 1, 15.968184820909466: 1, 15.937166333367893: 1,
15.935600961002258: 1, 15.890812232991458: 1, 15.853918836728726: 1, 15.82787879741012: 1,
15.818506941766282: 1, 15.814121073647579: 1, 15.730923793891833: 1, 15.692610028686936: 1,
15.691693603363694: 1, 15.652804830067815: 1, 15.565061121275914: 1, 15.548508340198701: 1,
15.541459743380114: 1, 15.527878944462767: 1, 15.490585524477895: 1, 15.478408612824628: 1,
15.469586891007859: 1, 15.461379319745081: 1, 15.434675376303655: 1, 15.430775940367544: 1,
15.41548733967926: 1, 15.353982978992262: 1, 15.338916273553702: 1, 15.32075540914188: 1,
15.29213707731727: 1, 15.26511414943579: 1, 15.255524093133992: 1, 15.242764921283424: 1,
15.241329083372483: 1, 15.22625923332637: 1, 15.210053932072338: 1, 15.164782355829038: 1,
15.149688750071306: 1, 15.147402465459638: 1, 15.105763687203622: 1, 15.09247609464531: 1,
15.07327985346284: 1, 15.057249383903587: 1, 15.051619265639733: 1, 15.046043080252506: 1,

15.032288209240015: 1, 15.011531654568147: 1, 15.008900422456474: 1, 15.000772361880284: 1,
14.993482393504118: 1, 14.96794249157899: 1, 14.92306507913161: 1, 14.908761493232811: 1,
14.907542474939904: 1, 14.885353989278908: 1, 14.884973657214084: 1, 14.857930588077172: 1,
14.845215097987763: 1, 14.81433583153587: 1, 14.80595419432331: 1, 14.799960525065035: 1,
14.779961665109086: 1, 14.739854092786382: 1, 14.738620611916067: 1, 14.721506990746892: 1,
14.648762075218581: 1, 14.63409268185454: 1, 14.583234907044138: 1, 14.576614392345686: 1,
14.559677669600468: 1, 14.544966621571305: 1, 14.542716209719195: 1, 14.532029732837966: 1,
14.506125257877946: 1, 14.476411703247857: 1, 14.432038592610217: 1, 14.415517642486448: 1,
14.40382680934368: 1, 14.380519685765854: 1, 14.351826707462283: 1, 14.336982329280548: 1,
14.247826052676878: 1, 14.247333315583314: 1, 14.215170187175005: 1, 14.183154594137493: 1,
14.175777889412304: 1, 14.149559631293105: 1, 14.092335900327889: 1, 14.071658383369803: 1,
14.050118496541605: 1, 13.984465686164118: 1, 13.973078148218532: 1, 13.921865105438629: 1,
13.879805582407073: 1, 13.875310839499026: 1, 13.865082947882746: 1, 13.850606352846665: 1,
13.79185410048901: 1, 13.768713488585126: 1, 13.73756547481706: 1, 13.7089526838948: 1,
13.70195744833872: 1, 13.663089301778502: 1, 13.656927561292013: 1, 13.651238350992625: 1,
13.649368781905437: 1, 13.592936404818879: 1, 13.576027347827402: 1, 13.572417475168882: 1,
13.56489700084903: 1, 13.562755669127107: 1, 13.554018382328492: 1, 13.532242380662645: 1,
13.511516326687858: 1, 13.509050338056184: 1, 13.506527587148062: 1, 13.500219063072244: 1,
13.444270966716061: 1, 13.441610263700023: 1, 13.434244546140757: 1, 13.419355428372201: 1,
13.414733978460552: 1, 13.40743532369851: 1, 13.384211157617242: 1, 13.375936669962472: 1,
13.324105433950733: 1, 13.29928347899567: 1, 13.297233594741828: 1, 13.283212967959072: 1,
13.274957352264947: 1, 13.257405628135512: 1, 13.243302085638383: 1, 13.198253903431644: 1,
13.141065519824972: 1, 13.05591553922027: 1, 13.055298322381121: 1, 12.99825243257721: 1,
12.992574107564973: 1, 12.976181253035765: 1, 12.949769918967004: 1, 12.941903002730005: 1,
12.933448712557068: 1, 12.91970166540113: 1, 12.91919088599462: 1, 12.871932539693436: 1,
12.860818081714363: 1, 12.829722101801732: 1, 12.806890421910555: 1, 12.758126084100928: 1,
12.725158269381202: 1, 12.705678407278759: 1, 12.702371836736342: 1, 12.695073175826504: 1,
12.681238589719802: 1, 12.677739622877281: 1, 12.66901721306531: 1, 12.667164442885822: 1,
12.58626397782261: 1, 12.583665950360146: 1, 12.573731059794133: 1, 12.527693267241695: 1,
12.523743666085583: 1, 12.51639560191847: 1, 12.506805595963545: 1, 12.505333984017: 1,
12.501629488689243: 1, 12.492655192814096: 1, 12.489536854129978: 1, 12.468799617209982: 1,
12.462377789371791: 1, 12.45653343336149: 1, 12.432640105694718: 1, 12.412885485159245: 1,
12.402739932954631: 1, 12.389404489363702: 1, 12.385391161175829: 1, 12.330879435138248: 1,
12.330222606292917: 1, 12.300083592371994: 1, 12.292195821581464: 1, 12.276730967171666: 1,
12.27366280884993: 1, 12.26794163487348: 1, 12.248681557340799: 1, 12.242336252780001: 1,
12.217800313383757: 1, 12.179126715695245: 1, 12.148275106970214: 1, 12.141061576524729: 1,
12.139474122409196: 1, 12.138920129203191: 1, 12.117954727568828: 1, 12.09550412366281: 1,
12.076432020658107: 1, 12.043232121022774: 1, 12.04195211336145: 1, 12.001928350659371: 1,
11.99565556985164: 1, 11.984317802085158: 1, 11.968237995565023: 1, 11.960301672992726: 1,
11.937972978709674: 1, 11.932594643199462: 1, 11.927324442133221: 1, 11.916920327527826: 1,
11.9006858771255: 1, 11.872720403201368: 1, 11.818808940958345: 1, 11.774918004585029: 1,
11.74539930910867: 1, 11.74030539576771: 1, 11.740090011965329: 1, 11.708341050820787: 1,
11.682226161449531: 1, 11.6495909299552: 1, 11.642175651160978: 1, 11.641405960072992: 1,
11.548170363095997: 1, 11.54543065455737: 1, 11.544867478001123: 1, 11.541846963731794: 1,
11.53123731888949: 1, 11.531066075252127: 1, 11.524681450821703: 1, 11.49560245477816: 1,
11.493921393759663: 1, 11.452208540785612: 1, 11.451812224417129: 1, 11.438721146203124: 1,
11.364900201310936: 1, 11.35520019668379: 1, 11.345072595555049: 1, 11.34405724121051: 1,
11.33776782035247: 1, 11.32126210289193: 1, 11.319397019805782: 1, 11.315351590964001: 1,
11.278842121912897: 1, 11.270189097499506: 1, 11.26222692133985: 1, 11.261497935953207: 1,
11.244531680487226: 1, 11.244163682680192: 1, 11.241912986221362: 1, 11.236790005353381: 1,
11.232128615056714: 1, 11.230314789894157: 1, 11.228126753585872: 1, 11.218596131136485: 1,
11.21545212517325: 1, 11.204730726791832: 1, 11.200495193199455: 1, 11.175574701904987: 1,
11.17343650930242: 1, 11.16738632669684: 1, 11.15933560404717: 1, 11.134056291705301: 1,
11.130019773868078: 1, 11.12210916326284: 1, 11.116755912081725: 1, 11.112776279221068: 1,
11.111276882583713: 1, 11.102265651635578: 1, 11.064822775341202: 1, 11.013184686420335: 1,
10.979679055174666: 1, 10.939091796519866: 1, 10.938041719188824: 1, 10.929891612183212: 1,
10.916710804824508: 1, 10.900139933004438: 1, 10.885102526623537: 1, 10.87990817745973: 1,
10.872377168681215: 1, 10.870374644483526: 1, 10.836937561570643: 1, 10.82284829740454: 1,
10.819995244321472: 1, 10.819172943705173: 1, 10.773982699615427: 1, 10.770199181930895: 1,
10.752882026019826: 1, 10.752820442675622: 1, 10.73038158462062: 1, 10.725248764618925: 1,
10.719095058815597: 1, 10.70630100245696: 1, 10.702996912670942: 1, 10.700887081816067: 1,
10.69439648412663: 1, 10.67040926470251: 1, 10.654552108881793: 1, 10.64294064022267: 1,
10.631258510223807: 1, 10.627408559571498: 1, 10.60835032817049: 1, 10.587321102303111: 1,
10.58050940826137: 1, 10.537211631518945: 1, 10.523817424162818: 1, 10.509154687001006: 1,
10.502774145841416: 1, 10.49740172353036: 1, 10.490456166491976: 1, 10.459453785221106: 1,
10.457179244522031: 1, 10.446659941365938: 1, 10.435679595103002: 1, 10.408351762344095: 1,
10.407233907014694: 1, 10.382624647787662: 1, 10.350817378952582: 1, 10.345149827974188: 1,
10.342145250347865: 1, 10.320158777899112: 1, 10.310702536005453: 1, 10.310025050265605: 1,
10.309934668826152: 1, 10.258569738164518: 1, 10.256904096423572: 1, 10.24442664370223: 1,
10.231252506204013: 1, 10.231119363811032: 1, 10.228257545834262: 1, 10.215555308525973: 1,
10.207890436951011: 1, 10.19881920430095: 1, 10.192160491920507: 1, 10.170768016115193: 1,
10.169286114409598: 1, 10.158894129066384: 1, 10.153960240500355: 1, 10.153324018882524: 1,
10.151614320958686: 1, 10.136775136059365: 1, 10.134376501175337: 1, 10.124805522223108: 1,
10.111507056912856: 1, 10.102351165190216: 1, 10.102295509284808: 1, 10.091651296328605: 1,
10.085720691531797: 1, 10.085506905380583: 1, 10.074116775856659: 1, 10.05657689478837: 1,
10.053469541650168: 1, 10.042407554320427: 1, 10.042165760802414: 1, 10.033685084596216: 1,
10.032762197388125: 1, 10.023027515385785: 1, 10.019332270073416: 1, 9.956205012473331: 1,

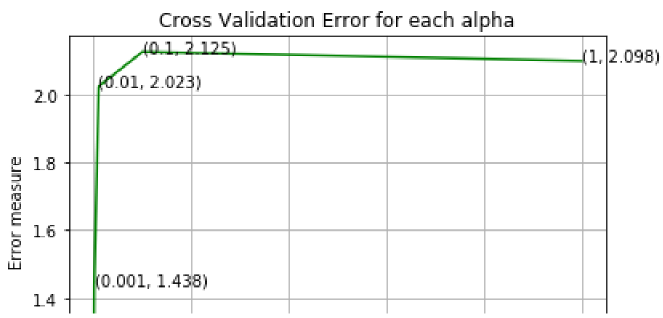
```
9.944063170412525: 1, 9.94134232236788: 1, 9.923395624784986: 1, 9.923349880973882: 1,
9.914391212181979: 1, 9.91356658162806: 1, 9.907091335657247: 1, 9.892767332461762: 1,
9.878985365815817: 1, 9.872365745008496: 1, 9.839363724986347: 1, 9.827307737371342: 1,
9.819316216241969: 1, 9.812742224935366: 1, 9.80282107092211: 1, 9.788511985386638: 1,
9.778182724116206: 1, 9.753467790509884: 1, 9.730639319612404: 1, 9.726915890407064: 1,
9.71069750798245: 1, 9.695838754946763: 1, 9.69535272887245: 1, 9.69368044176532: 1,
9.684670565164724: 1, 9.677559871554749: 1, 9.67646390150053: 1, 9.663118827696104: 1,
9.656922822646509: 1, 9.649935405066122: 1, 9.64557733545324: 1, 9.638764302350191: 1,
9.63212066622879: 1, 9.620793361666873: 1, 9.6099864959595: 1, 9.605810701365638: 1,
9.596679827284385: 1, 9.54751593877357: 1, 9.535984905113281: 1, 9.496953131253195: 1,
9.495305130511248: 1, 9.486908988111281: 1, 9.483245792601672: 1, 9.470402940025606: 1,
9.46613161187345: 1, 9.455167796917399: 1, 9.447092594674594: 1, 9.443511934552562: 1,
9.43382356191621: 1, 9.424324309625483: 1, 9.419017331851139: 1, 9.414439387167379: 1,
9.408154044937179: 1, 9.399947951582106: 1, 9.39422420652567: 1, 9.381754743163748: 1,
9.375653262386761: 1, 9.365861344915404: 1, 9.365674571948855: 1, 9.325073080309041: 1,
9.32069318175793: 1, 9.319539434770265: 1, 9.319321861903918: 1, 9.316135682549646: 1,
9.285573133537225: 1, 9.282308133982385: 1, 9.257710933453774: 1, 9.253010158153105: 1,
9.244353842854142: 1, 9.235560859198422: 1, 9.2346788672071: 1, 9.206712899378523: 1,
9.205733694370837: 1, 9.202643119474216: 1, 9.194426301662386: 1, 9.191763080131155: 1,
9.18927078075952: 1, 9.186557596949102: 1, 9.181969919387958: 1, 9.164515419259905: 1,
9.143582658667027: 1, 9.140210470700952: 1, 9.134665979773843: 1, 9.131516243905814: 1,
9.127156593338999: 1, 9.116861689267845: 1, 9.114014378239473: 1, 9.113399205734252: 1,
9.111788531621988: 1, 9.104935908154127: 1, 9.085848114051766: 1, 9.066906098382406: 1,
9.062092066304286: 1, 9.05293422764485: 1, 9.042465267222942: 1, 9.024711357959433: 1,
8.992466032364009: 1, 8.984126132309271: 1, 8.980627155454943: 1, 8.977781106825708: 1,
8.966832638979815: 1, 8.957539717164542: 1, 8.952184946709462: 1, 8.947648312402555: 1,
8.94318512857735: 1, 8.937247256601202: 1, 8.936059032507995: 1, 8.916337615394978: 1,
8.911429943777982: 1, 8.907032182002636: 1, 8.897286308107356: 1, 8.895497142295252: 1,
8.864553963204976: 1, 8.863394305289575: 1, 8.855589826734114: 1, 8.853189961214536: 1,
8.847412609649744: 1, 8.834495687985417: 1, 8.829254833916135: 1, 8.826111043526618: 1,
8.817097269548658: 1, 8.813296371576138: 1, 8.803439895543509: 1, 8.795744030356504: 1,
8.78650496139027: 1, 8.782916980977783: 1, 8.754154859218866: 1, 8.746013437030923: 1,
8.737749552512785: 1, 8.736263520004812: 1, 8.732624352720412: 1, 8.731873814735563: 1,
8.728965318016401: 1, 8.711529141785958: 1, 8.697152455393594: 1, 8.667354368587384: 1,
8.660199718093413: 1, 8.62741004442276: 1, 8.589804897700194: 1, 8.580807857982023: 1,
8.574004062268365: 1, 8.558371404476143: 1, 8.552195427998123: 1, 8.53842964403895: 1,
8.524439340547419: 1, 8.50720079332992: 1, 8.482706520473535: 1, 8.476620852197843: 1,
8.473164341475997: 1, 8.455040036667963: 1, 8.448640394022748: 1, 8.43418247727304: 1,
8.432792565014548: 1, 8.421138608482448: 1, 8.420896000581577: 1, 8.413512994265343: 1,
8.404929738641068: 1, 8.387173806079426: 1, 8.385522634404717: 1, 8.358071217975114: 1,
8.356029955061524: 1, 8.349172954511564: 1, 8.342249510384445: 1, 8.317518959011572: 1,
8.315041244880732: 1, 8.304598586822939: 1, 8.30391253249989: 1, 8.293520913328512: 1,
8.287293836329585: 1, 8.284951219974959: 1, 8.281133770537888: 1, 8.276090969363059: 1,
8.244108811220343: 1, 8.21661648239883: 1, 8.203374843004827: 1, 8.19614284133955: 1,
8.168394839983327: 1, 8.150349113174107: 1, 8.128948376554002: 1, 8.06932880129278: 1,
8.061100038160436: 1, 8.048070513001129: 1, 8.038944499422433: 1, 8.023600725245698: 1,
8.022694352954908: 1, 8.022391326221316: 1, 8.0137298241541: 1, 8.012632582444029: 1,
8.010940206037674: 1, 7.9999234418111955: 1, 7.974844629402459: 1, 7.971534819754301: 1,
7.968968507518866: 1, 7.951731302419122: 1, 7.94976222583441: 1, 7.9464225339126635: 1,
7.935925203937697: 1, 7.9345278213756645: 1, 7.928513990834298: 1, 7.92067232525677: 1,
7.912453007537953: 1, 7.874873948398512: 1, 7.8698588177030135: 1, 7.8432853321646965: 1,
7.831600219231824: 1, 7.813486408077346: 1, 7.806421671261651: 1, 7.805903856022445: 1,
7.785054392672589: 1, 7.756510922517301: 1, 7.739709802169936: 1, 7.725917128879447: 1,
7.6955717014968705: 1, 7.684937825713798: 1, 7.681041609769158: 1, 7.662906446101357: 1, 7.64783890
64111045: 1, 7.640173727722437: 1, 7.624405975381535: 1, 7.612360825494665: 1, 7.609766447324731:
1, 7.596587101038303: 1, 7.59014113591465: 1, 7.5800577007829135: 1, 7.578483131158092: 1,
7.562412033100059: 1, 7.557608796299638: 1, 7.538736784351862: 1, 7.533871397891775: 1,
7.529906190963129: 1, 7.515823131728324: 1, 7.510016310475832: 1, 7.507131068132315: 1,
7.498516100491958: 1, 7.462731757881189: 1, 7.444606896638274: 1, 7.429348635416504: 1,
7.4153631767503905: 1, 7.3525630505998585: 1, 7.348961915269858: 1, 7.342276907959339: 1,
7.3280042762236: 1, 7.326178417968048: 1, 7.325150169919994: 1, 7.306813440253414: 1, 7.30088188974
3009: 1, 7.1697400643315: 1, 7.1666226349432245: 1, 7.160462927164436: 1, 7.158106663968774: 1,
7.154130767965352: 1, 7.111673933284698: 1, 7.070122957211835: 1, 7.051771218827369: 1,
7.041627098091684: 1, 7.030317285653396: 1, 7.018924620969349: 1, 7.005225662884764: 1,
7.002569844880855: 1, 6.9689698105143005: 1, 6.966273113388892: 1, 6.894143202356875: 1,
6.842739313323377: 1, 6.819825522536672: 1, 6.812948464299724: 1, 6.783984342086131: 1,
6.757753481432383: 1, 6.729166300234247: 1, 6.704635243382411: 1, 6.6851899515110285: 1,
6.641215228973966: 1, 6.620799939245916: 1, 6.556307344780746: 1, 6.548773988033842: 1,
6.356145427022083: 1})
```

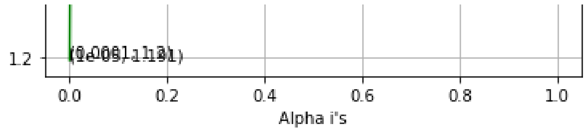
In [48]:

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

# read more about SGDClassifier() at http://scikit-
```

```
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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:
#------------------------------


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.clas
ses_, 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_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.190970191995295
For values of alpha =  0.0001 The log loss is: 1.2002620996145816
For values of alpha =  0.001 The log loss is: 1.4381855375244441
For values of alpha =  0.01 The log loss is: 2.022829066630401
For values of alpha =  0.1 The log loss is: 2.1251205209344732
For values of alpha =  1 The log loss is: 2.0984960087389704
```






```
For values of best alpha =  1e-05 The train log loss is: 0.7927146596035191
For values of best alpha =  1e-05 The cross validation log loss is: 1.190970191995295
For values of best alpha =  1e-05 The test log loss is: 1.1181996155527585
```

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

Ans. Yes, it seems like!

In [49]:

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

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

In [50]:

```
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")
```

```
3.467 % of word of test data appeared in train data
3.928 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [51]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    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 [52]:

```
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 [53]:

```
# 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_r
o))
        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 [54]:

```
# 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))
```

```
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 [55]:

```
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, 3201)
(number of data points * number of features) in test data =  (665, 3201)
(number of data points * number of features) in cross validation data = (532, 3201)
```

In [56]:

```
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

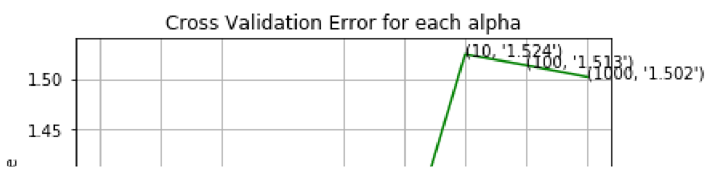
4.1.1.1. Hyper parameter tuning

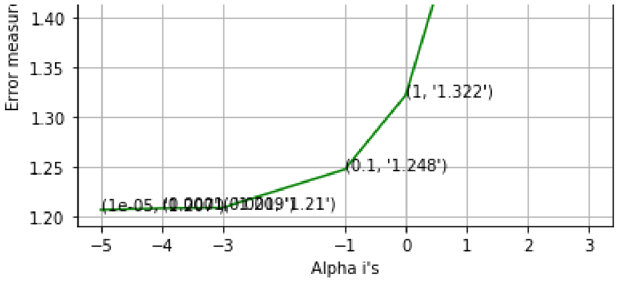
In [57]:

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

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


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

```
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


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

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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-05
Log Loss : 1.2073079192444574
for alpha = 0.0001
Log Loss : 1.2088950142235964
for alpha = 0.001
Log Loss : 1.209546768616028
for alpha = 0.1
Log Loss : 1.2476567915907748
for alpha = 1
Log Loss : 1.321875960959209
for alpha = 10
Log Loss : 1.524486008488661
for alpha = 100
Log Loss : 1.5133785869103091
for alpha = 1000
Log Loss : 1.5017908474508157
```





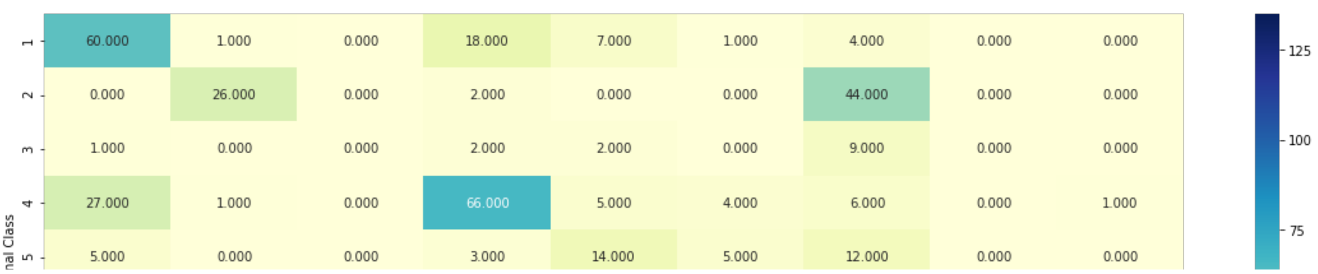

```
For values of best alpha =  1e-05 The train log loss is: 0.5261174102795565
For values of best alpha =  1e-05 The cross validation log loss is: 1.2073079192444574
For values of best alpha =  1e-05 The test log loss is: 1.220805562207063
```

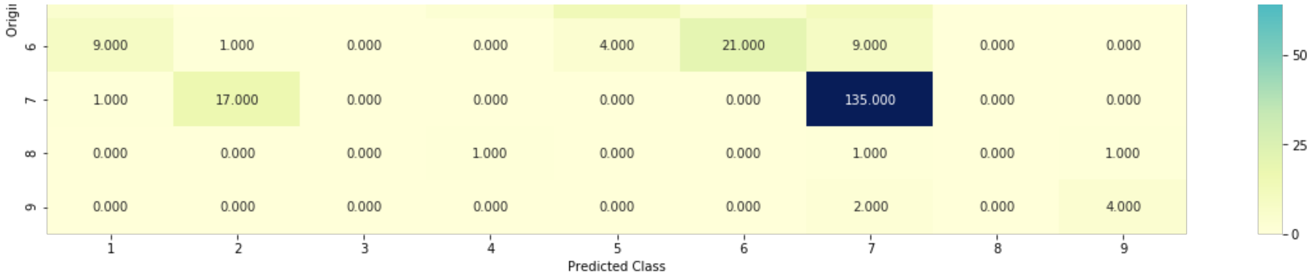
4.1.1.2. Testing the model with best hyper paramters

In [58]:

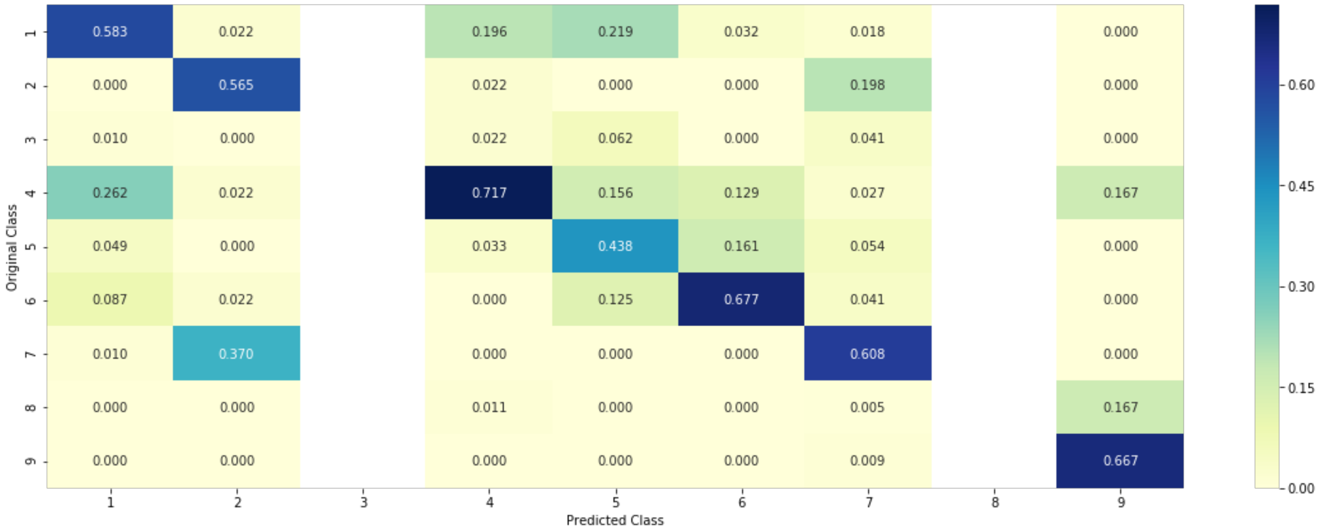
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
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 : 1.2073079192444574
Number of missclassified point : 0.38721804511278196
-------------------- Confusion matrix --------------------
```

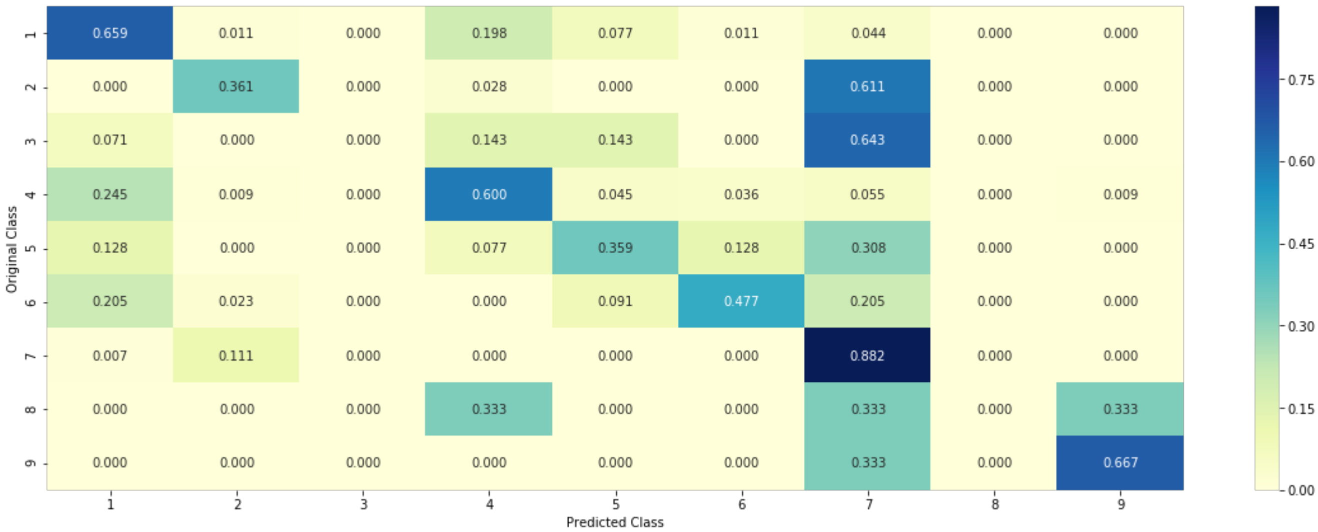





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.1.1.3. Feature Importance, Correctly classified point

In [59]:

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

```
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0606 0.0737 0.0105 0.0664 0.0329 0.031  0.718  0.0036 0.0032]]
Actual Class : 7
--------------------------------------------------
19 Text feature [003] present in test data point [True]
54 Text feature [113] present in test data point [True]
55 Text feature [05] 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 [61]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0958 0.0593 0.0144 0.0911 0.5706 0.0429 0.1166 0.0049 0.0044]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

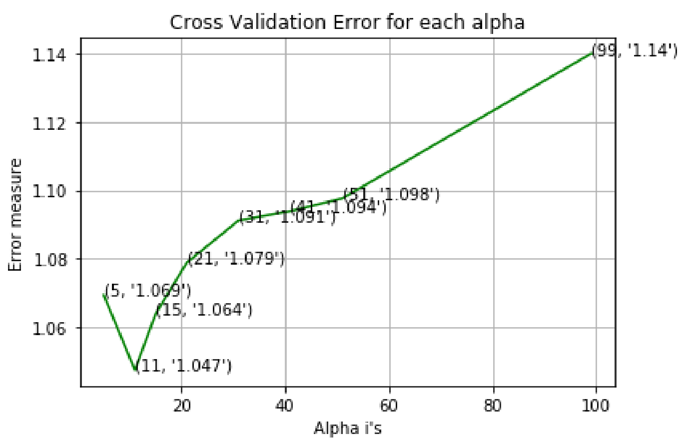
4.2.1. Hyper parameter tuning

In [62]:

```
# 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 probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 1.0693223971331096
for alpha = 11
Log Loss : 1.0474413142199157
for alpha = 15
Log Loss : 1.0638440488053782
for alpha = 21
Log Loss : 1.0787695957131083
for alpha = 31
Log Loss : 1.091124157635689
for alpha = 41
Log Loss : 1.093848094546082
for alpha = 51
Log Loss : 1.0975630115050266
for alpha = 99
Log Loss : 1.1398713102230686
```



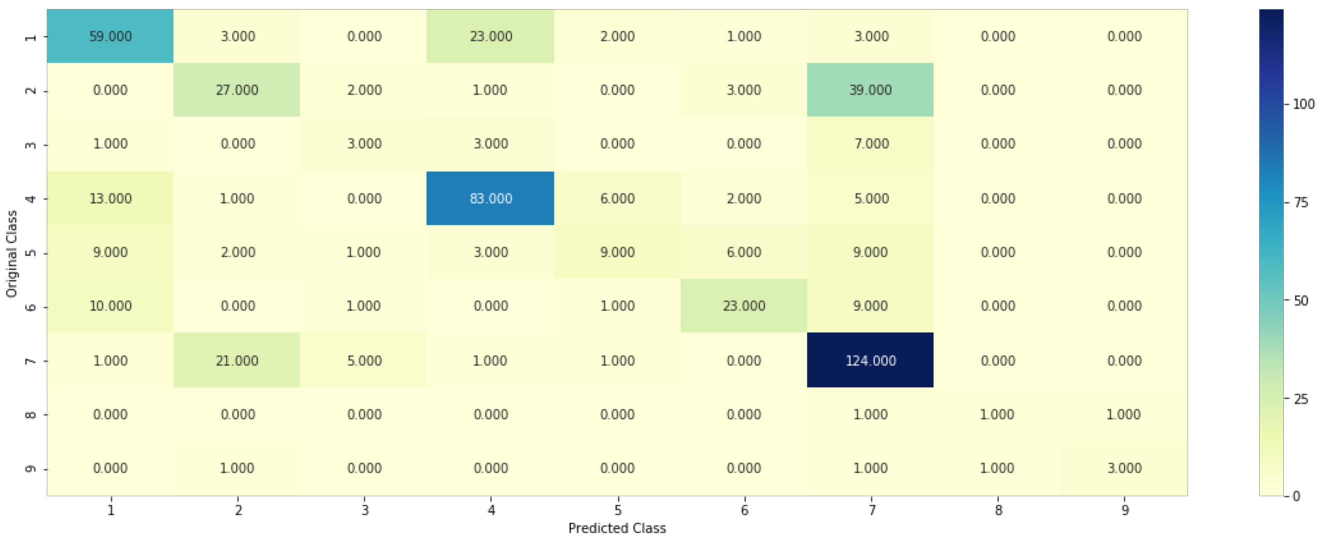
```
For values of best alpha =  11 The train log loss is: 0.6319513259457193
For values of best alpha =  11 The cross validation log loss is: 1.0474413142199157
For values of best alpha =  11 The test log loss is: 1.082338297252076
```

4.2.2. Testing the model with best hyper paramters

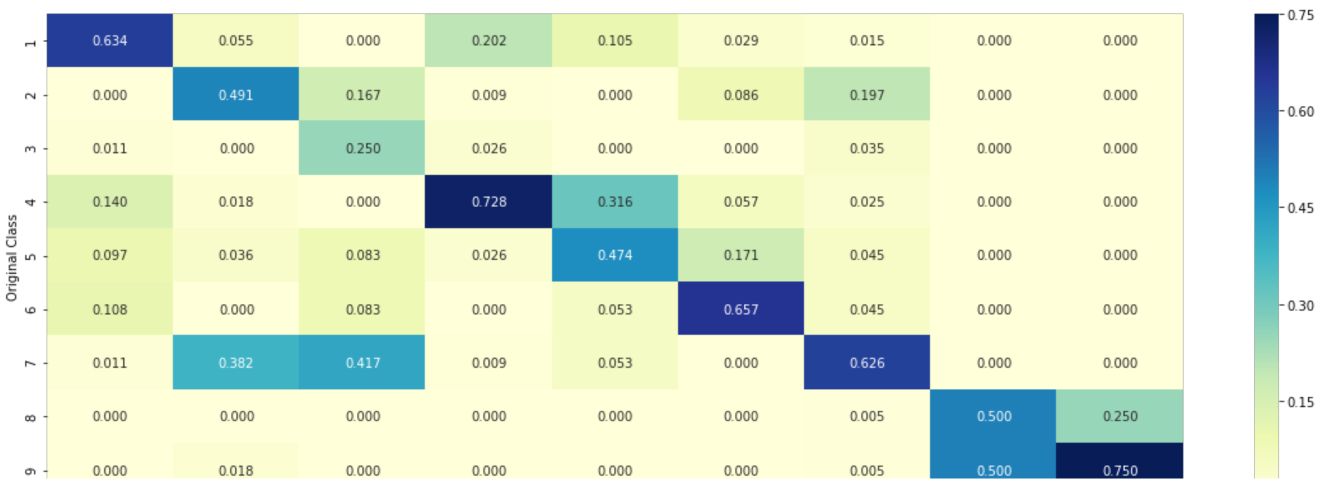
In [63]:

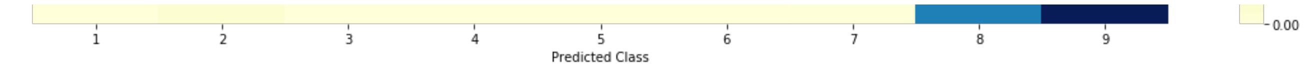
```python
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', 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 : 1.0474413142199157
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

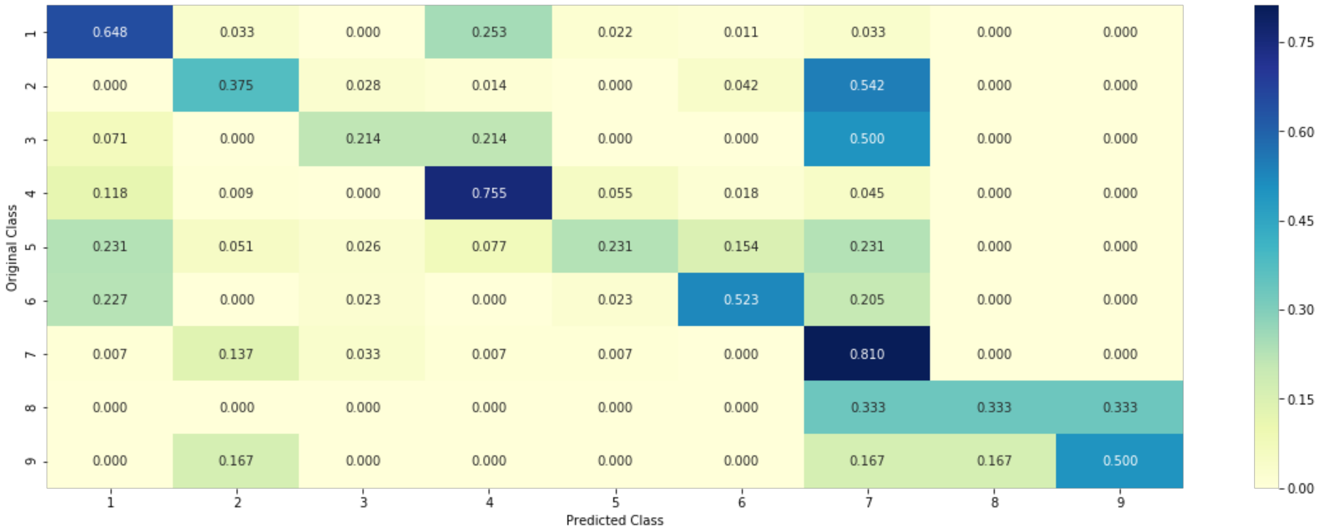



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



4.2.3.Sample Query point -1

In [64]:

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

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

```
Predicted Class : 7
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 7 4 4 7 7 7 7 2]
Fequency of nearest points : Counter({7: 8, 4: 2, 2: 1})
```

4.2.4. Sample Query Point-2

In [65]:

```
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 be
longs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [1 1 1
1 5 5 1 1 4 5 4]
Fequency of nearest points : Counter({1: 6, 5: 3, 4: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

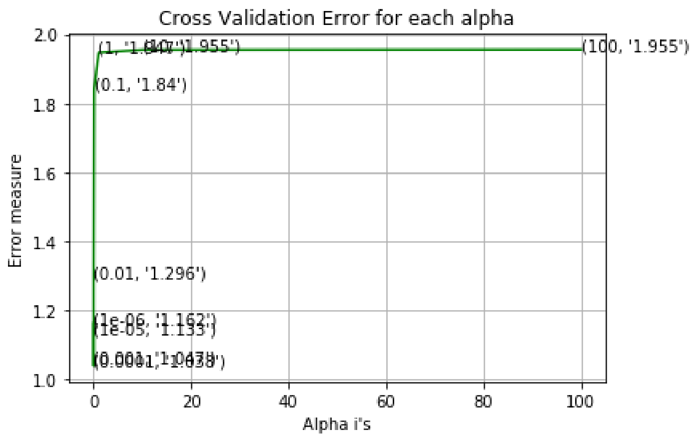
4.3.1.1. Hyper paramter tuning

In [66]:

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

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

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


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

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

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

```
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_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 : 1.1618131465112616
for alpha = 1e-05
Log Loss : 1.132728723179636
for alpha = 0.0001
Log Loss : 1.0376557521411913
for alpha = 0.001
Log Loss : 1.0465008777850782
for alpha = 0.01
Log Loss : 1.295530064919022
for alpha = 0.1
Log Loss : 1.8401512633924022
for alpha = 1
Log Loss : 1.9469178896034376
for alpha = 10
Log Loss : 1.9546525338049245
for alpha = 100
Log Loss : 1.9553488131149375
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44049467815781884
For values of best alpha =  0.0001 The cross validation log loss is: 1.0376557521411913
For values of best alpha =  0.0001 The test log loss is: 1.0274900312851711
```

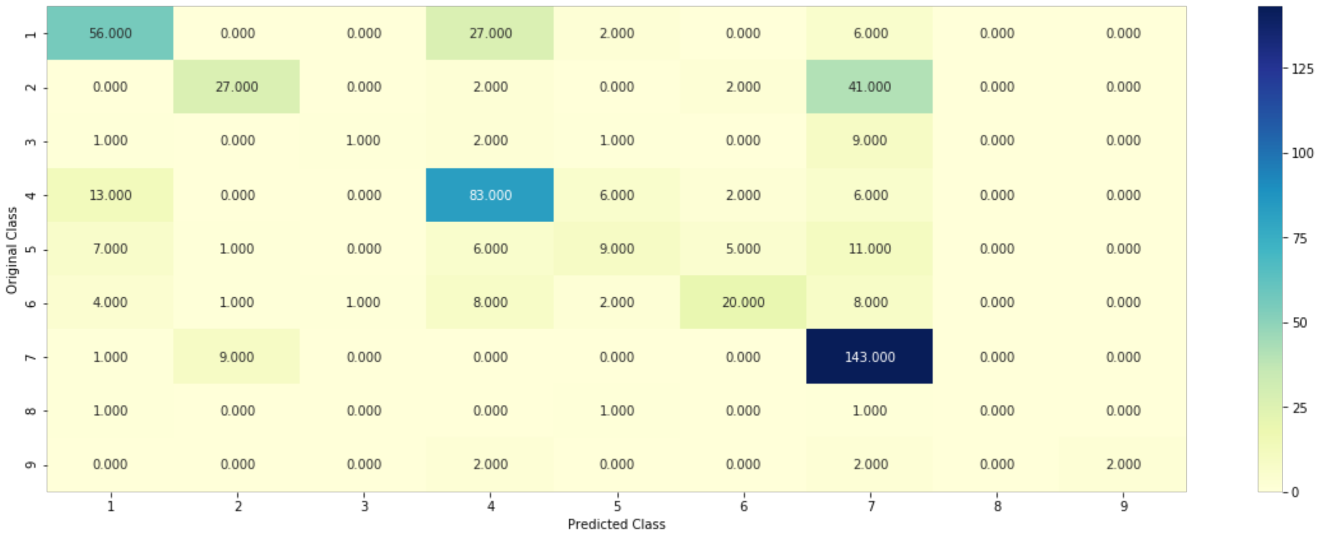
4.3.1.2. Testing the model with best hyper paramters

In [67]:

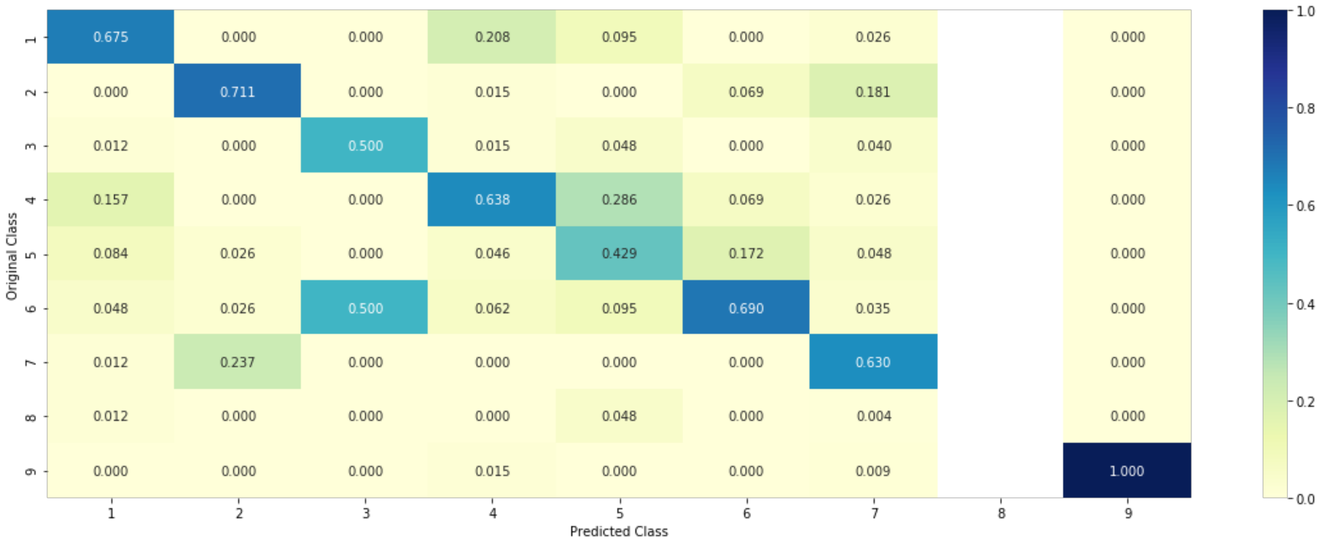
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

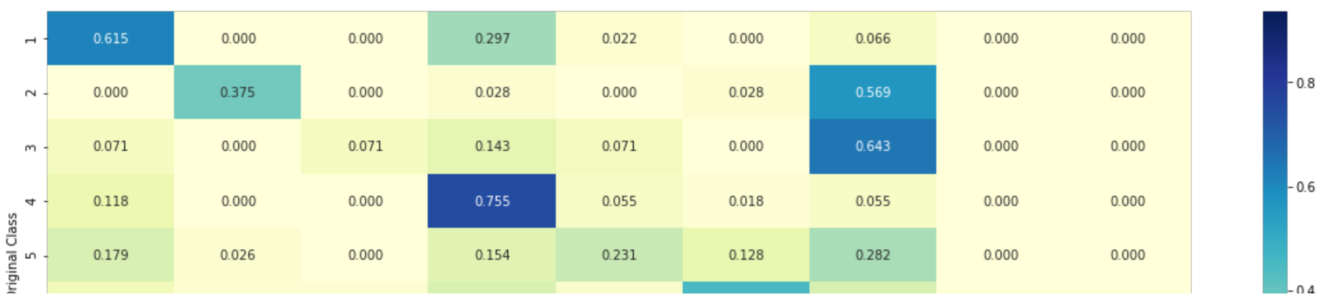
```
Log loss : 1.0376557521411913
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```

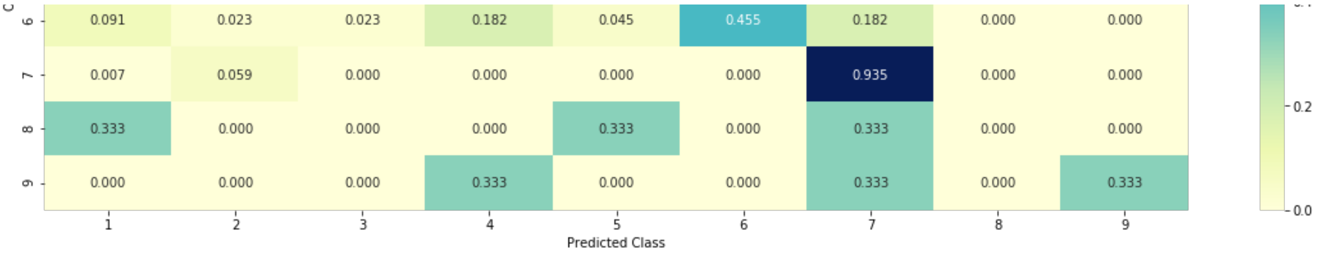



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.1.3. Feature Importance

In [68]:

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

4.3.1.3.1. Correctly Classified point

In [69]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0083 0.0445 0.0033 0.0091 0.0054 0.0133 0.9097 0.0043 0.0021]]
Actual Class : 7
--------------------------------------------------
10 Text feature [05] present in test data point [True]
29 Text feature [11] present in test data point [True]
36 Text feature [003] present in test data point [True]
203 Text feature [13] present in test data point [True]
282 Text feature [113] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [70]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[7.458e-01 1.700e-03 1.000e-03 6.310e-02 1.789e-01 4.900e-03 1.000
e-03
  3.200e-03 5.000e-04]]
Actual Class : 1
--------------------------------------------------
278 Text feature [09] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2. Without Class balancing

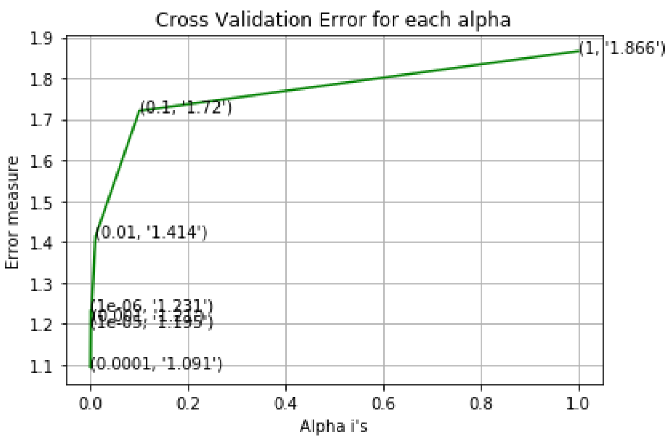
4.3.2.1. Hyper paramter tuning

In [71]:

```
# 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)]
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.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_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 : 1.231469343756046
for alpha = 1e-05
Log Loss : 1.1950823028724153
for alpha = 0.0001
Log Loss : 1.0912221036900396
for alpha = 0.001
Log Loss : 1.2097823670629346
for alpha = 0.01
Log Loss : 1.413626748470682
for alpha = 0.1
Log Loss : 1.7204789997969845
for alpha = 1
Log Loss : 1.86631127759044
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4393736832710683
For values of best alpha =  0.0001 The cross validation log loss is: 1.0912221036900396
For values of best alpha =  0.0001 The test log loss is: 1.04774344307071
```

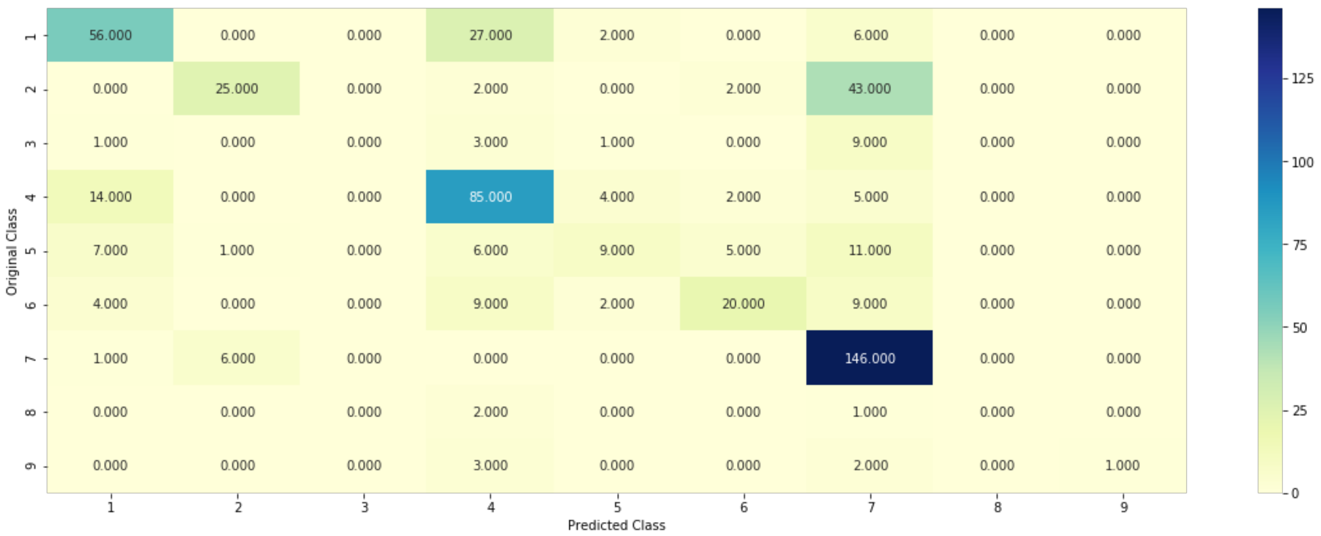
4.3.2.2. Testing model with best hyper parameters

In [72]:

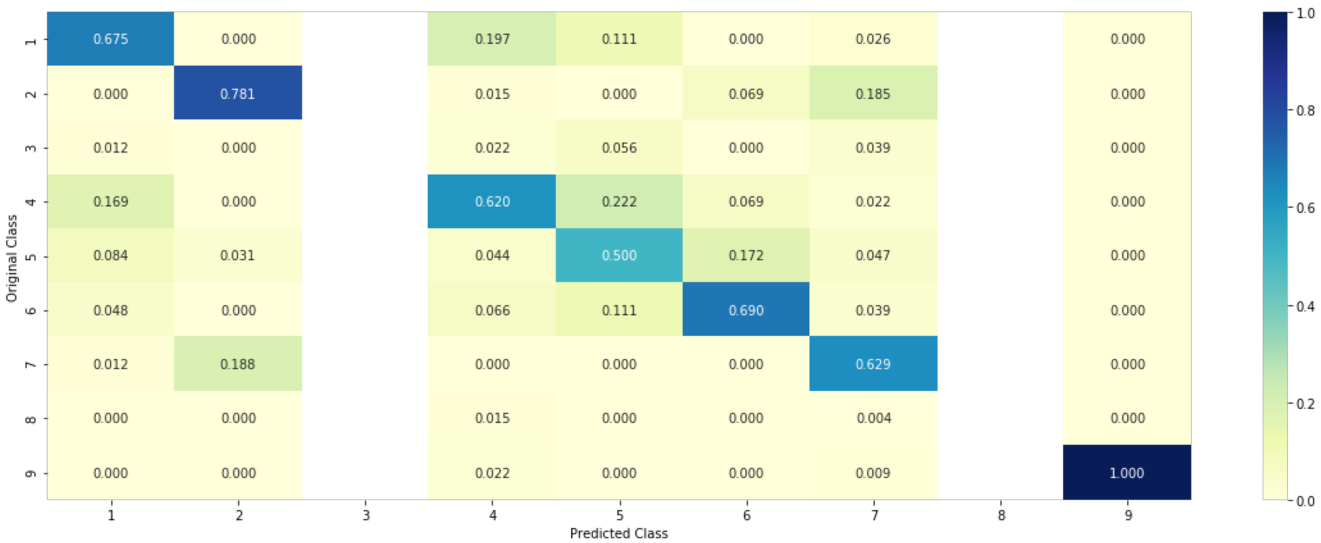
```
# read more about SGDClassifier() at http://scikit-
```

```
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:
#------------------------------

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)
```

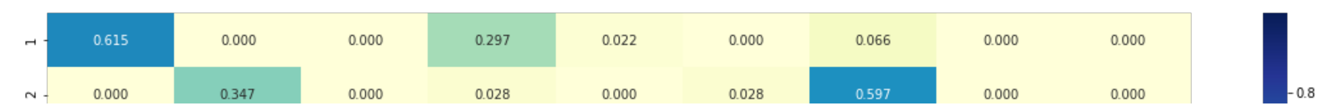
```
Log loss : 1.0912221036900396
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```

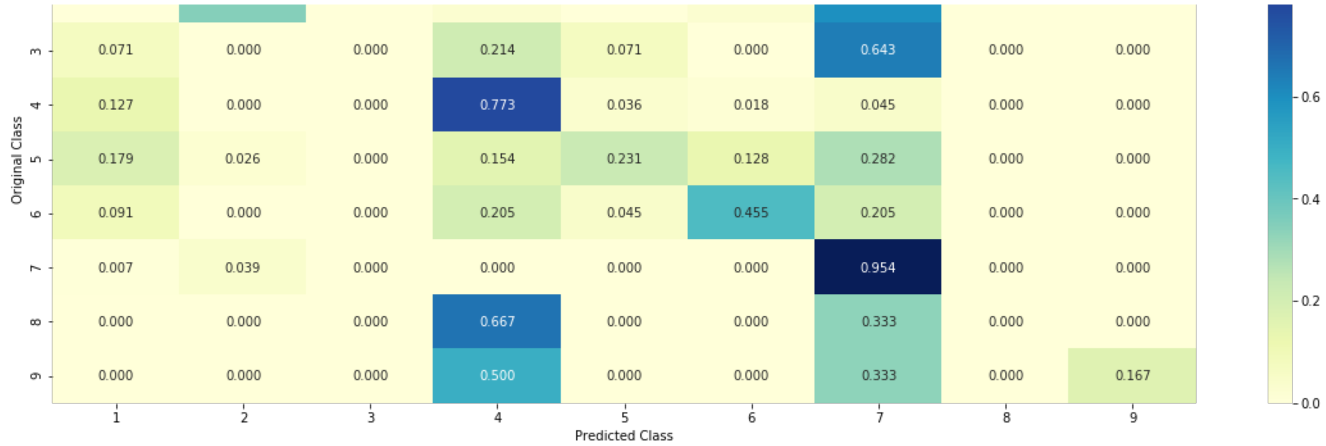



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.2.3. Feature Importance, Correctly Classified point

In [73]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[9.400e-03 4.140e-02 1.900e-03 9.900e-03 5.100e-03 1.170e-02 9.148
e-01
  5.200e-03 6.000e-04]]
Actual Class : 7
--------------------------------------------------
10 Text feature [05] present in test data point [True]
61 Text feature [11] present in test data point [True]
77 Text feature [003] present in test data point [True]
223 Text feature [13] present in test data point [True]
252 Text feature [113] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [74]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7459 0.0018 0.0013 0.0754 0.1594 0.0045 0.0017 0.01   0.    ]]
Actual Class : 1
--------------------------------------------------
280 Text feature [09] present in test data point [True]
```

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

4.4. Linear Support Vector Machines

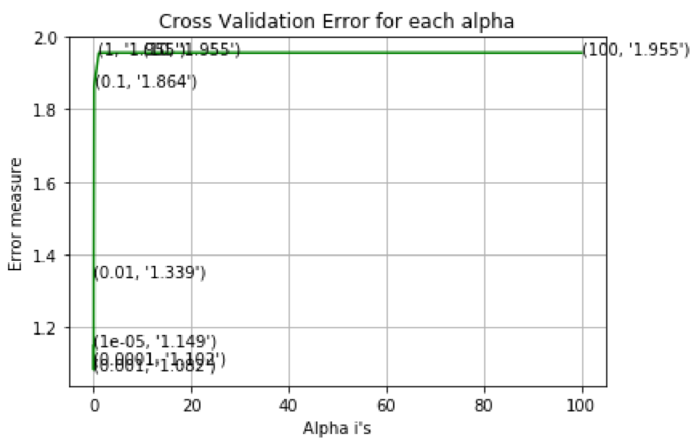
4.4.1. Hyper paramter tuning

In [75]:

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

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, 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/
# --------------------------------



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', r
andom_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 C = 1e-05
Log Loss : 1.1490517041859345
for C = 0.0001
Log Loss : 1.1022186514035937
for C = 0.001
Log Loss : 1.0821531774152828
for C = 0.01
Log Loss : 1.3387207704751602
for C = 0.1
Log Loss : 1.864381179703781
for C = 1
Log Loss : 1.9553249389714373
for C = 10
Log Loss : 1.9553288323866682
for C = 100
Log Loss : 1.9553251082337904
```




```
For values of best alpha =  0.001 The train log loss is: 0.5702308700617547
For values of best alpha =  0.001 The cross validation log loss is: 1.0821531774152828
For values of best alpha =  0.001 The test log loss is: 1.089993449561552
```

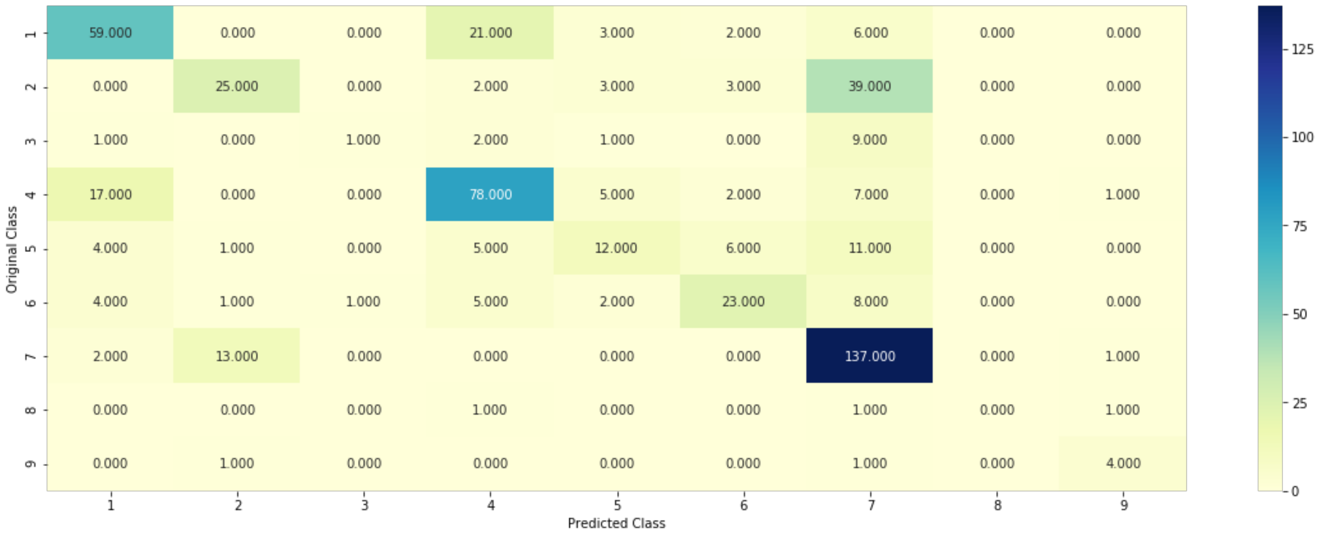
4.4.2. Testing model with best hyper parameters

In [76]:

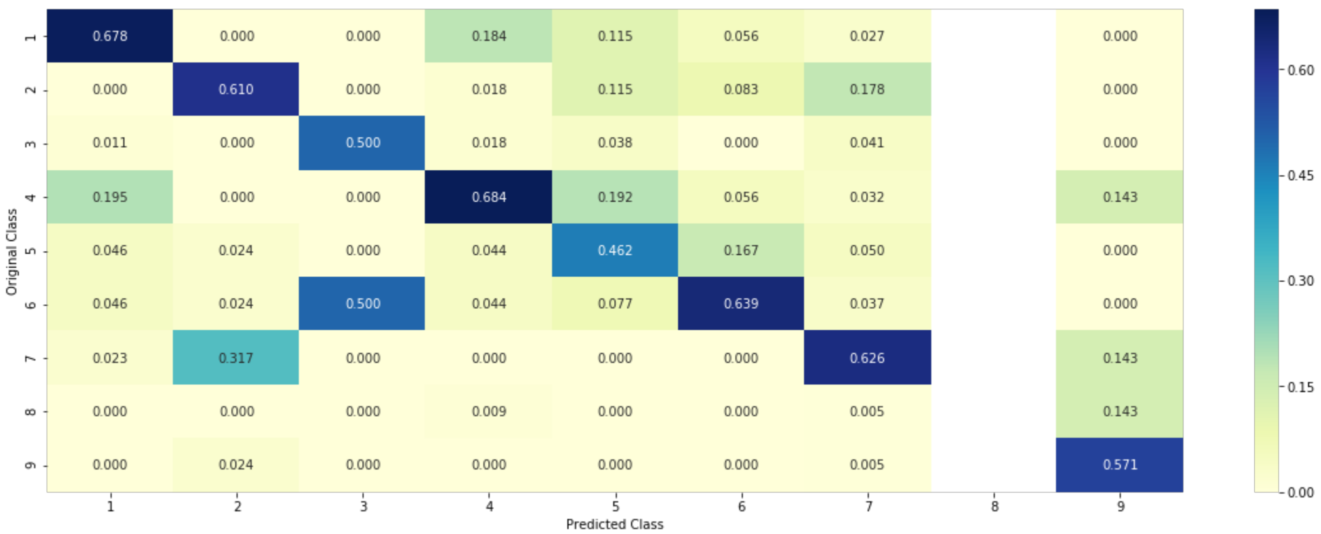
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, 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/
# --------------------------------
```

```
# 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)
```

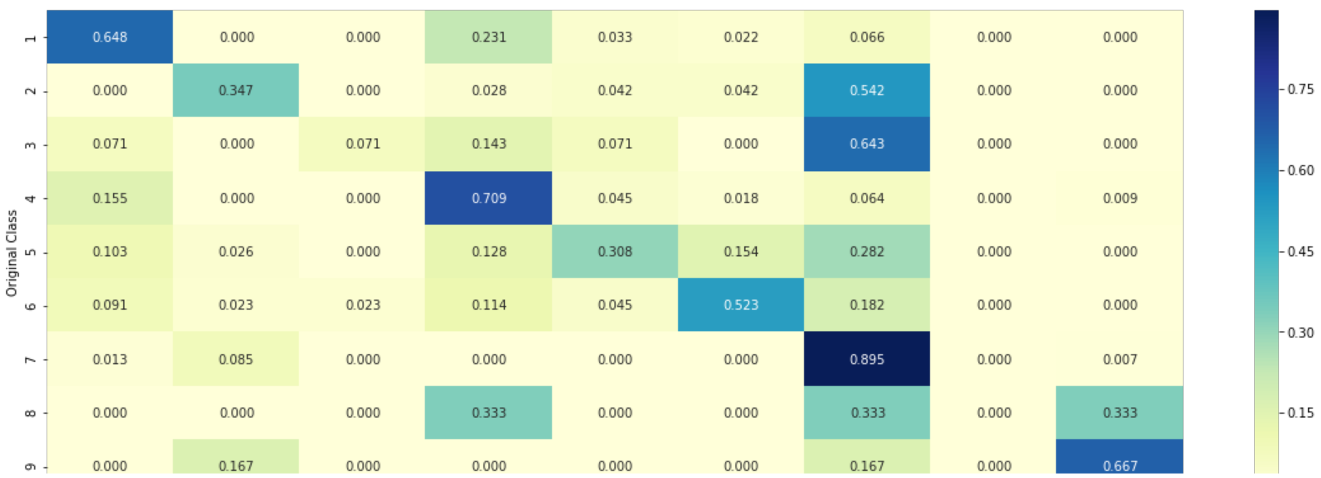
```
Log loss : 1.0821531774152828
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```

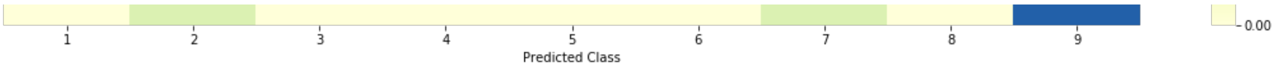



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [77]:

```
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(-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: [[0.0516 0.0229 0.0074 0.0439 0.0166 0.0335 0.8169 0.0048 0.0024]]
Actual Class : 7
--------------------------------------------------
18 Text feature [05] present in test data point [True]
32 Text feature [003] present in test data point [True]
45 Text feature [11] present in test data point [True]
259 Text feature [113] present in test data point [True]
283 Text feature [13] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [78]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6054 0.0372 0.0036 0.0722 0.2381 0.0082 0.0287 0.0054 0.0012]]
Actual Class : 1
--------------------------------------------------
347 Text feature [09] present in test data point [True]
387 Text feature [0008] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [79]:

```
# ----------------------------------
# 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/
# ----------------------------------


# 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))
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))
```

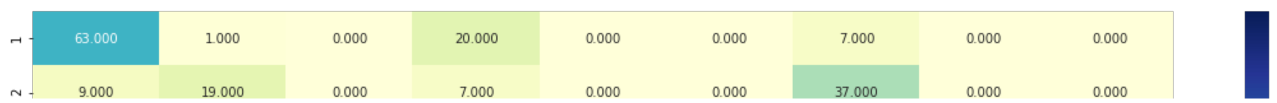
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2371773417444063
for n_estimators = 100 and max depth =  10
Log Loss : 1.281052493103825
for n_estimators = 200 and max depth =  5
Log Loss : 1.2239592110374093
for n_estimators = 200 and max depth =  10
Log Loss : 1.274374726574212
for n_estimators = 500 and max depth =  5
Log Loss : 1.2229360585130584
for n_estimators = 500 and max depth =  10
Log Loss : 1.2680232418400612
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2181605428787705
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2705171577567593
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2215609009777906
for n_estimators = 2000 and max depth =  10
Log Loss : 1.271284506447873
For values of best estimator =  1000 The train log loss is: 0.8529998811806977
For values of best estimator =  1000 The cross validation log loss is: 1.2181605428787712
For values of best estimator =  1000 The test log loss is: 1.1837052191998825
```

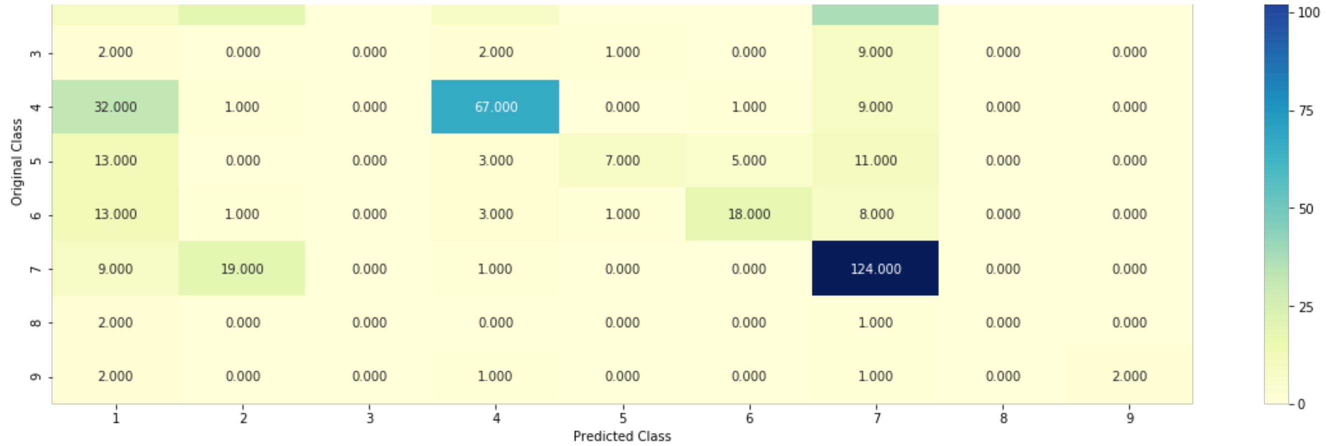
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [80]:

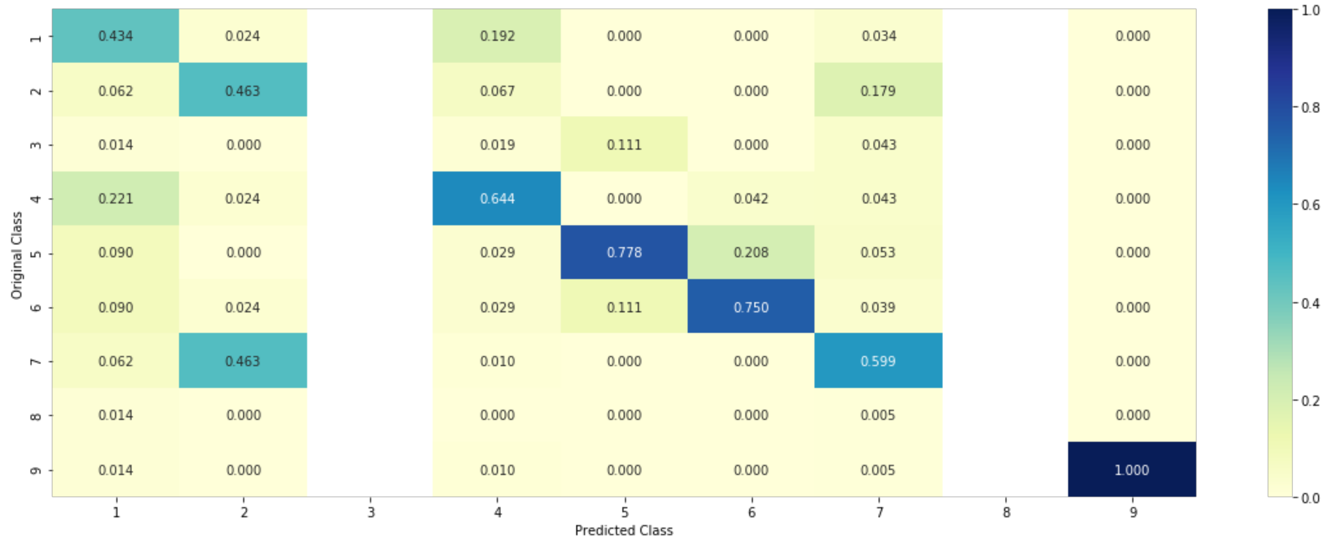
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.21816054287877
Number of mis-classified points : 0.43609022556390975
-------------------- Confusion matrix --------------------
```

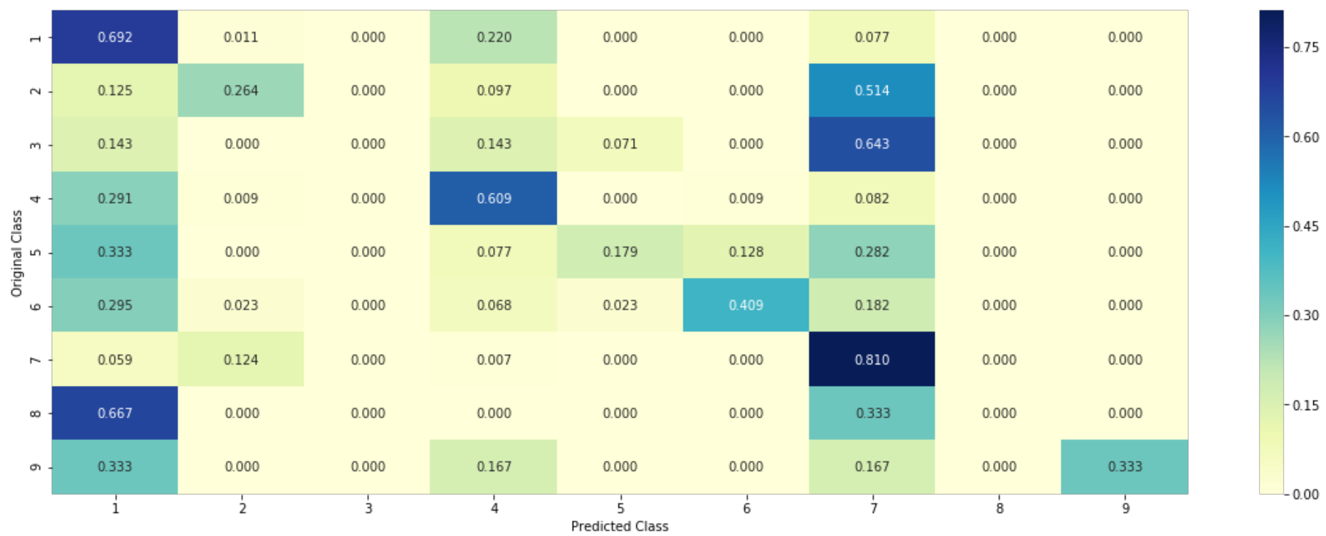





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [81]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
```

```
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0232 0.1854 0.0185 0.0212 0.034  0.0309 0.6777 0.0075 0.0016]]
Actual Class : 7
--------------------------------------------------
4 Text feature [003] present in test data point [True]
5 Text feature [113] present in test data point [True]
66 Text feature [11] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [82]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5982 0.0057 0.0086 0.0975 0.1873 0.0837 0.0129 0.0027 0.0033]]
Actuall Class : 1
--------------------------------------------------
66 Text feature [11] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [83]:

```
# --------------------------------
# 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/
# ----------------------------------


# 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 : 2.1180355296521283
for n_estimators = 10 and max depth =  3
Log Loss : 1.691158339011551
for n_estimators = 10 and max depth =  5
Log Loss : 1.438353525405276
for n_estimators = 10 and max depth =  10
Log Loss : 2.2103132738446956
for n_estimators = 50 and max depth =  2
```

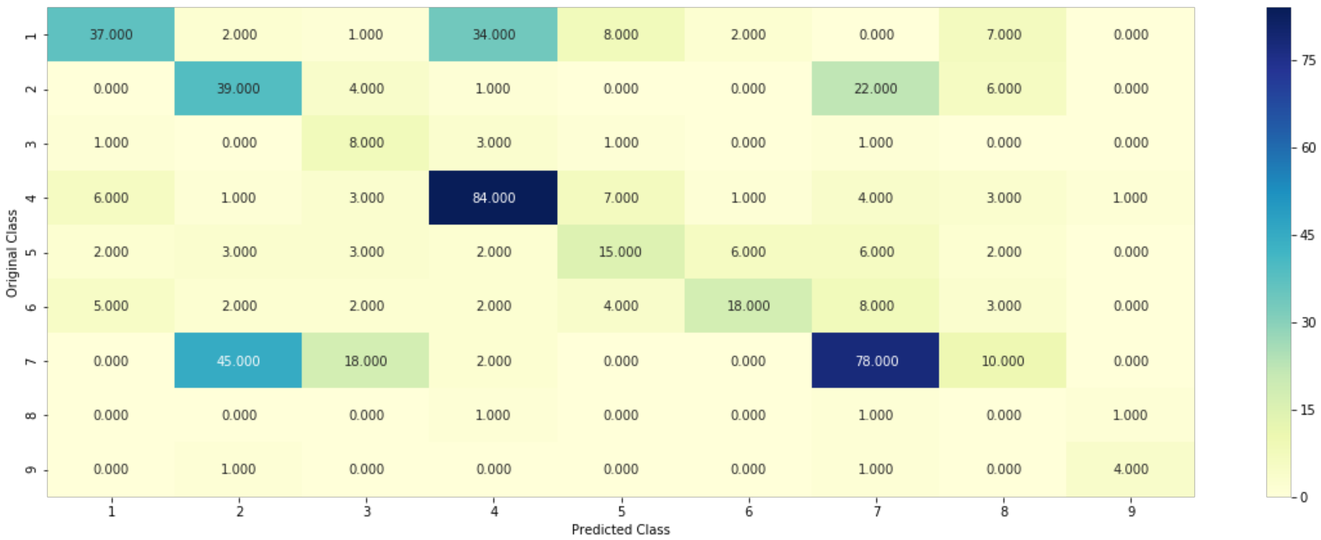
```
Log Loss : 1.742783517268029
for n_estimators = 50 and max depth =  3
Log Loss : 1.4975219861857265
for n_estimators = 50 and max depth =  5
Log Loss : 1.325329713589353
for n_estimators = 50 and max depth =  10
Log Loss : 1.7960751161751043
for n_estimators = 100 and max depth =  2
Log Loss : 1.5561417848198724
for n_estimators = 100 and max depth =  3
Log Loss : 1.469332496236588
for n_estimators = 100 and max depth =  5
Log Loss : 1.347517933031828
for n_estimators = 100 and max depth =  10
Log Loss : 1.716775313420476
for n_estimators = 200 and max depth =  2
Log Loss : 1.6114212032872774
for n_estimators = 200 and max depth =  3
Log Loss : 1.4716214318768122
for n_estimators = 200 and max depth =  5
Log Loss : 1.4074793409571635
for n_estimators = 200 and max depth =  10
Log Loss : 1.7038707211507016
for n_estimators = 500 and max depth =  2
Log Loss : 1.6712114617779303
for n_estimators = 500 and max depth =  3
Log Loss : 1.5469628228540728
for n_estimators = 500 and max depth =  5
Log Loss : 1.3997077604676824
for n_estimators = 500 and max depth =  10
Log Loss : 1.7511087331389497
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6614490859628117
for n_estimators = 1000 and max depth =  3
Log Loss : 1.557146820026146
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3916989200134728
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7471198094545222
For values of best alpha =  50 The train log loss is: 0.05679613757238183
For values of best alpha =  50 The cross validation log loss is: 1.325329713589353
For values of best alpha =  50 The test log loss is: 1.3516778607335125
```

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

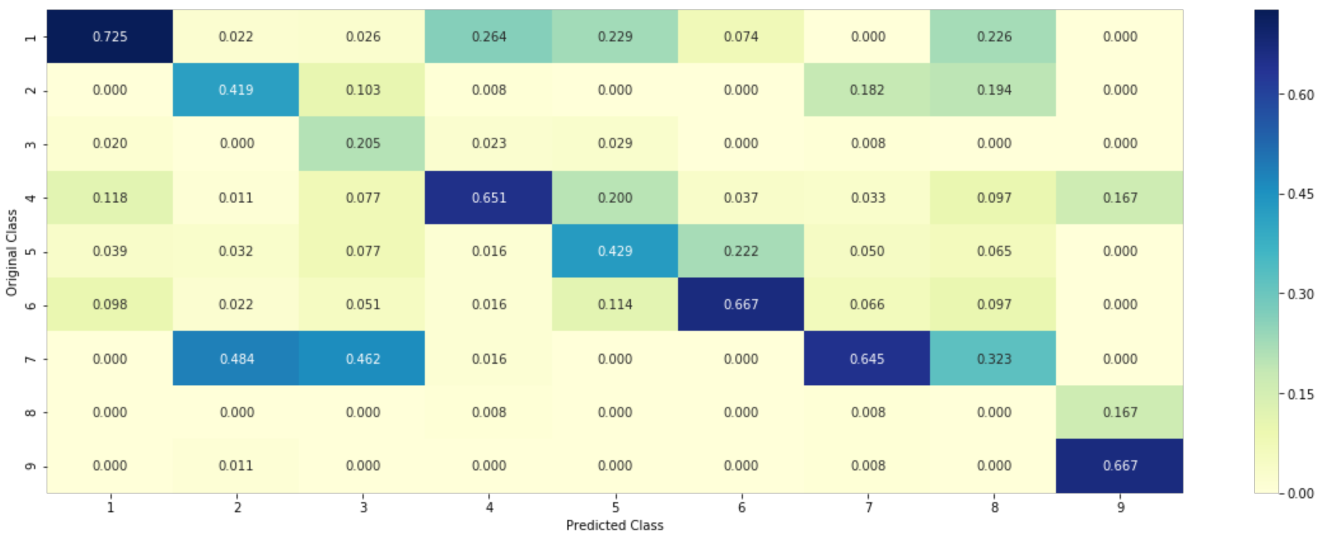
In [84]:

```
# --------------------------------
# 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)
```

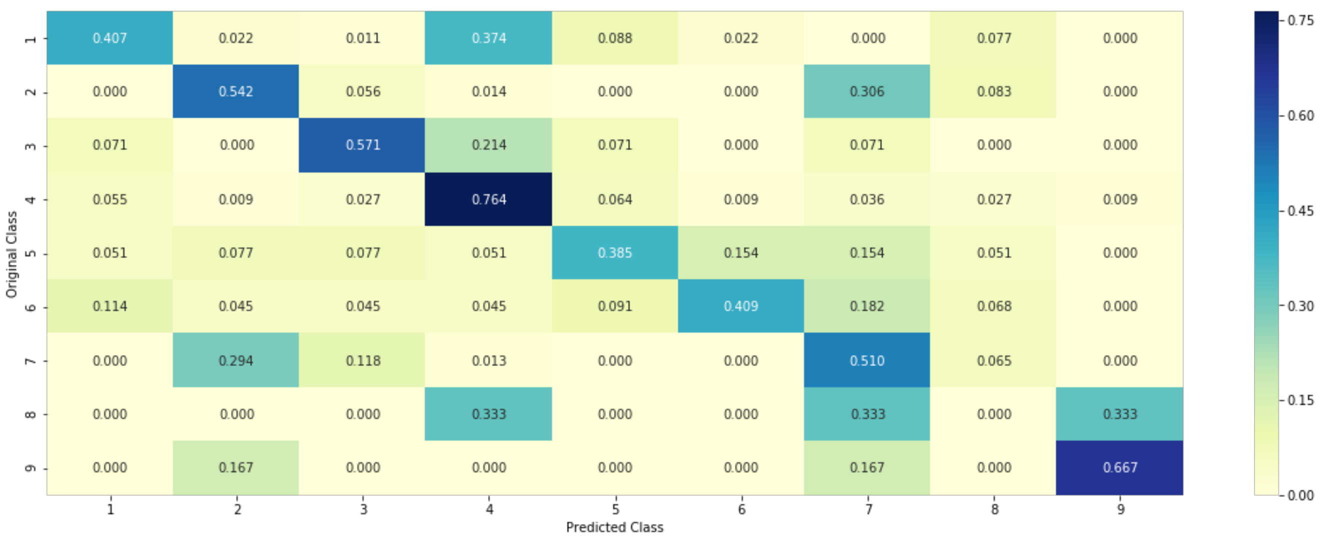
```
Log loss : 1.325329713589353
Number of mis-classified points : 0.4680451127819549
-------------------- Confusion matrix --------------------
```




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




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



4.5.5 Feature Importance

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [85]:

```
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 : 2
Predicted Class Probabilities: [[0.0083 0.4693 0.1073 0.0104 0.0194 0.021  0.2972 0.0605 0.0066]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [86]:

```
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 : 5
Predicted Class Probabilities: [[0.0667 0.0052 0.2192 0.0892 0.5157 0.0897 0.0041 0.0053 0.0049]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [87]:

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

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
```

```
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', 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 Classifer : 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
```

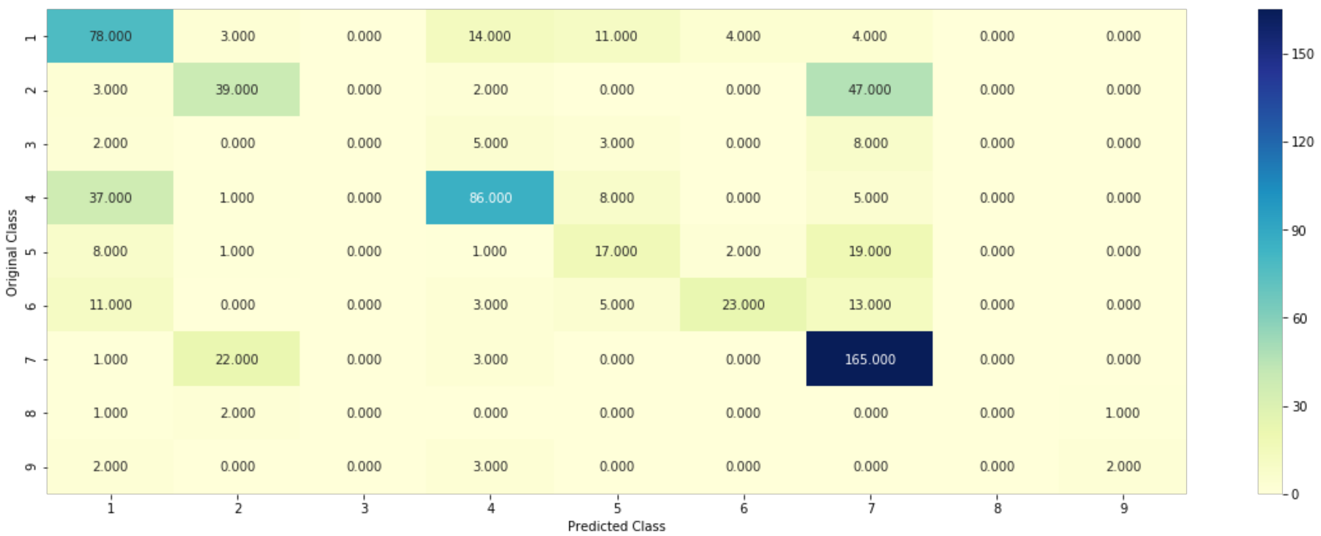
```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.96
Naive Bayes : Log Loss: 1.21
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.504
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.192
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.437
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.941
```

4.7.2 testing the model with the best hyper parameters

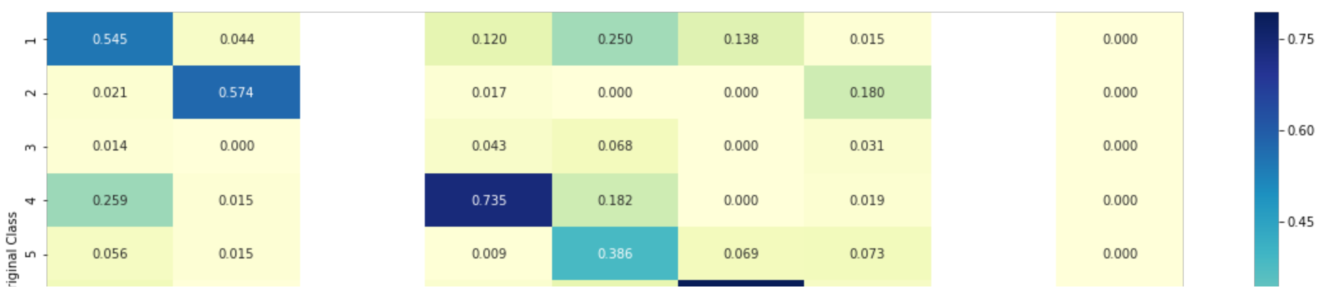
In [88]:

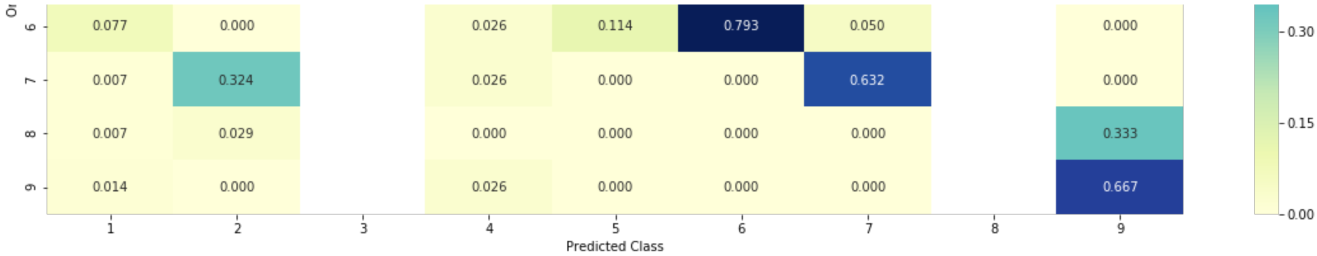
```python
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.5360224374926398
Log loss (CV) on the stacking classifier : 1.1922358608762562
Log loss (test) on the stacking classifier : 1.185589166904231
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```

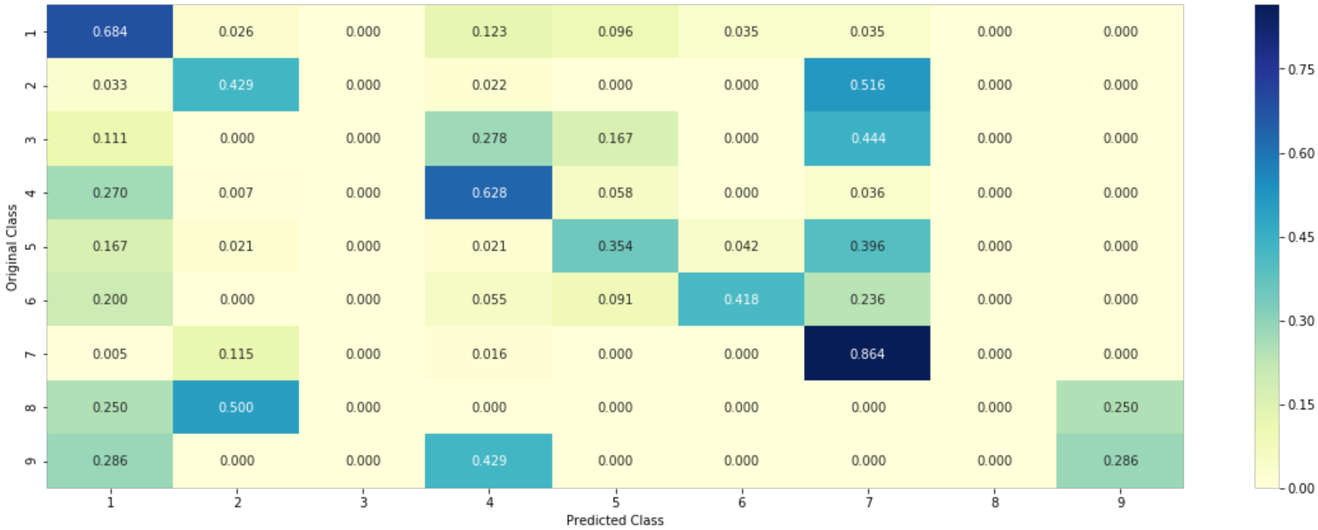



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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

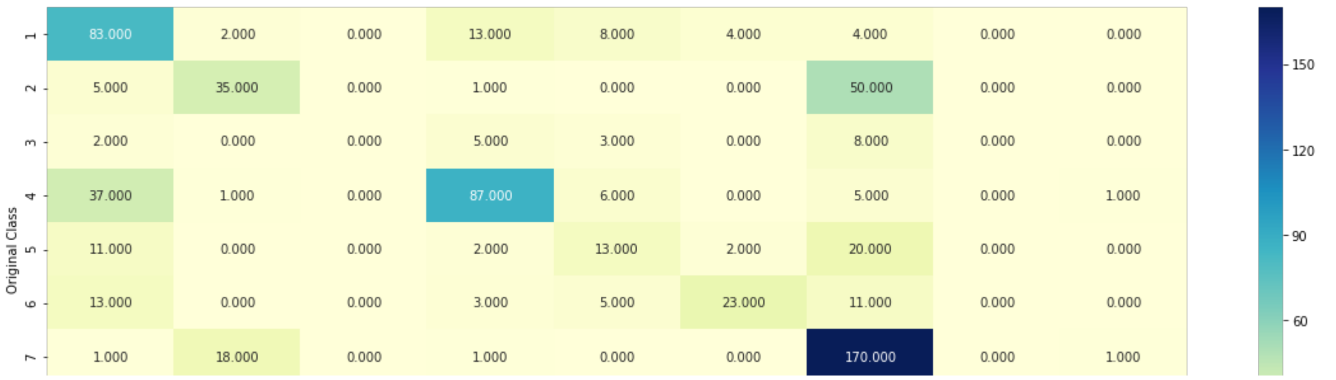


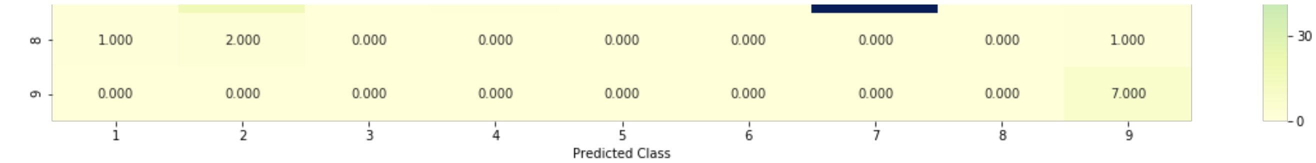
4.7.3 Maximum Voting classifier

In [89]:

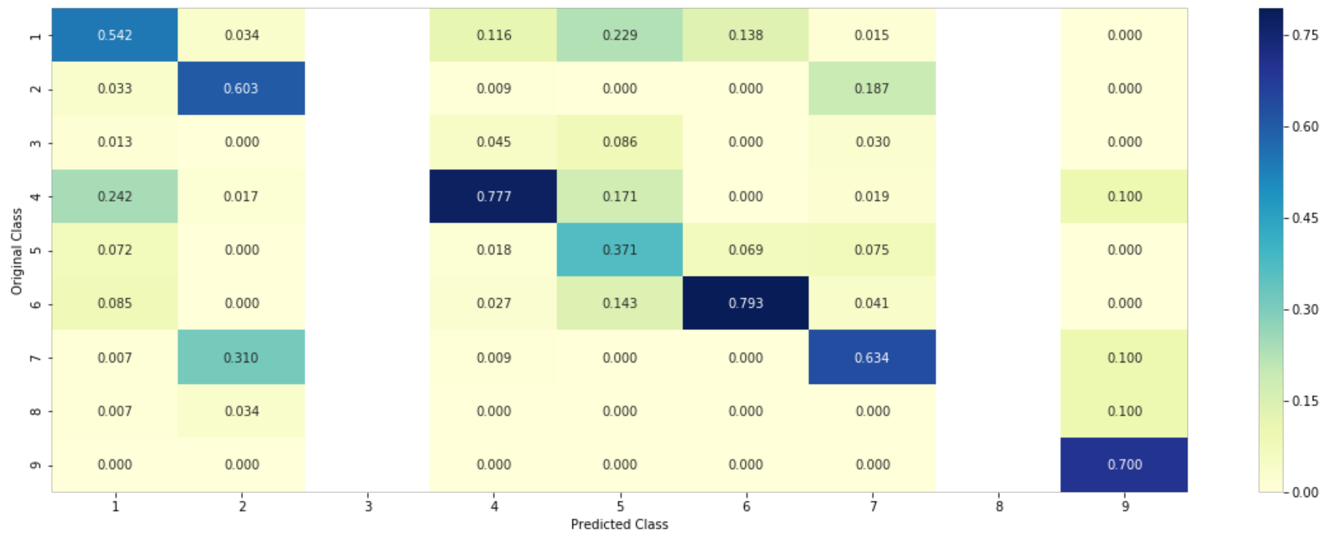
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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 : 0.8316593878662885
Log loss (CV) on the VotingClassifier : 1.2321821714368166
Log loss (test) on the VotingClassifier : 1.2127925842026392
Number of missclassified point : 0.37142857142857144
-------------------- Confusion matrix --------------------
```

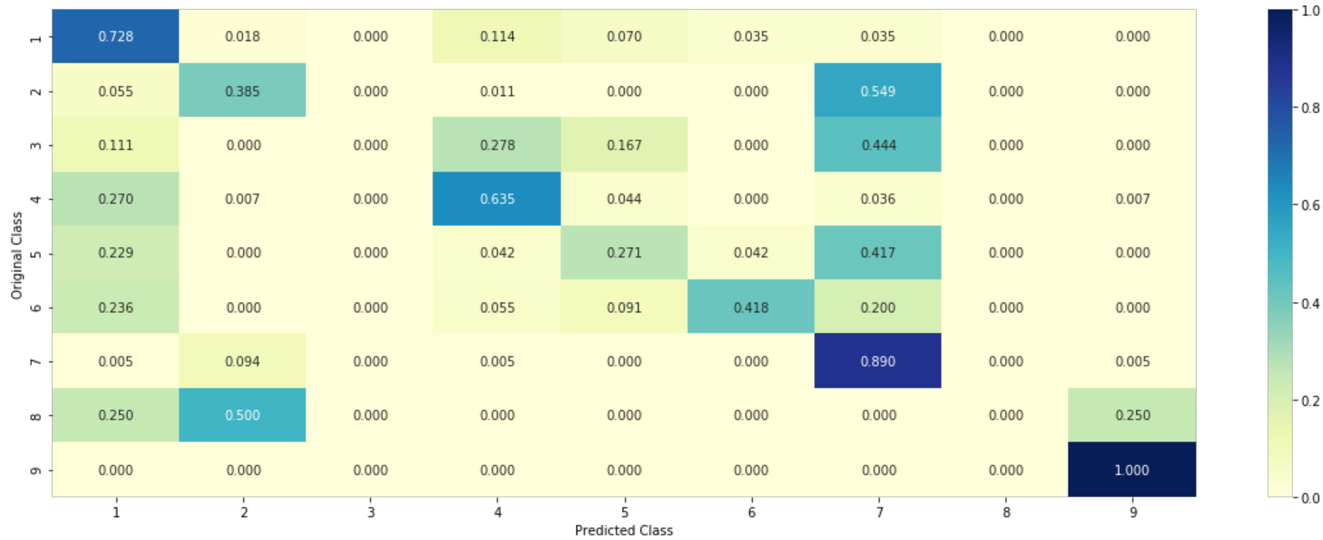





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



5. Conclusion

1. Applied All the models with tf-idf features (Replaced CountVectorizer with tfidfVectorizer and ran the same cells)
2. Used only the top 1000 words based on tf-idf values

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorization", "Model", "Train Loss", "CV Loss", "Test Loss", "Percentage Miscla
ssified"]

x.add_row(["OneHotEnCoding","NaiveBayes", 0.52,1.20,1.22,38.72])
x.add_row(["ResponseCoding","KNN",0.63,1.04,1.08,37.59])
x.add_row(["OneHotEnCoding ClassBalancing","LogisticRegression",0.44,1.03,1.02,35.90])
```

```
x.add_row(["OneHotEnCoding_ClassBalancing","LogisticRegression",0.44,1.03,1.02,35.90])
x.add_row(["OneHotEnCoding_Without_ClassBalancing","LogisticRegression",0.43,1.09,1.04,35.71])
x.add_row(["OneHotEnCoding","LinearSVM",0.57,1.082,1.089,36.27])
x.add_row(["OneHotEnCoding","RandomForest",0.85,1.21,1.18,43.60])
x.add_row(["ResponseCoding","RandomForest",0.05,1.32,1.35,46.80])
x.add_row(["OneHotEnCoding","Stacking",0.53,1.19,1.18,38.34])
x.add_row(["OneHotEnCoding","Voting",0.83,1.23,1.21,37.14])


print(x)
```

```
+---------------------------------------+--------------------+------------+---------+-----------+--
---------------------+
|              Vectorization            |       Model        | Train Loss | CV Loss | Test Loss | P
rcentage Misclassified |
+---------------------------------------+--------------------+------------+---------+-----------+--
---------------------+
|             OneHotEnCoding            |     NaiveBayes     |    0.52    |   1.2   |    1.22   |
38.72              |
|             ResponseCoding            |        KNN         |    0.63    |   1.04  |    1.08   |
37.59              |
|     OneHotEnCoding_ClassBalancing     | LogisticRegression |    0.44    |   1.03  |    1.02   |
35.9              |
| OneHotEnCoding_Without_ClassBalancing | LogisticRegression |    0.43    |   1.09  |    1.04   |
35.71              |
|             OneHotEnCoding            |     LinearSVM      |    0.57    |  1.082  |   1.089   |
36.27              |
|             OneHotEnCoding            |    RandomForest    |    0.85    |   1.21  |    1.18   |
43.6              |
|             ResponseCoding            |    RandomForest    |    0.05    |   1.32  |    1.35   |
46.8              |
|             OneHotEnCoding            |      Stacking      |    0.53    |   1.19  |    1.18   |
38.34              |
|             OneHotEnCoding            |       Voting       |    0.83    |   1.23  |    1.21   |
37.14              |
+---------------------------------------+--------------------+------------+---------+-----------+--
---------------------+
```

1. After Applying TfidfVectorizer with top 1000 words, Test Log Loss for LogisticRegression with Class Balancing = 1.02 which is lower than the log loss with BoW Vectorizer.
2. After Applying TfidfVectorizer with top 1000 words, Test Log Loss for LogisticRegression without Class Balancing = 1.04 which is lower than the log loss with BoW Vectorizer.