# 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]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
#from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
#from sklearn.cross_validation 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
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import nltk
nltk.download('stopwords')
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/externals/six.py:31: DeprecationWarning: The module is deprecated in version
0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of
six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)


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

```
True
```

# 3.1. Reading Data

## 3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

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

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

- **ID :** the id of the row used to link the mutation to the clinical evidence

- **Gene :** the gene where this genetic mutation is located
- **Variation :** the aminoacid change for this mutations
- **Class :** 1-9 the class this genetic mutation has been classified on

## 3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training/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']
```

| | 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 : 180.05482899999998 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()
```

| | 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)]
```

| | 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]
```

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

| | 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, te
st_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, te
st_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])
```

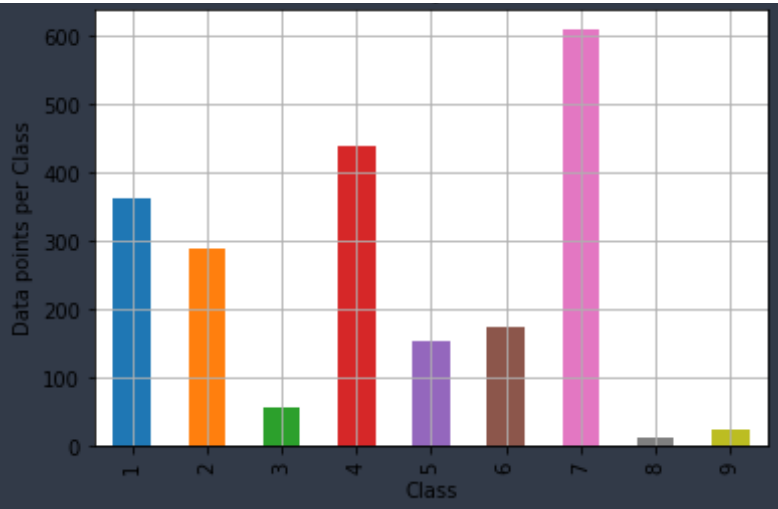
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

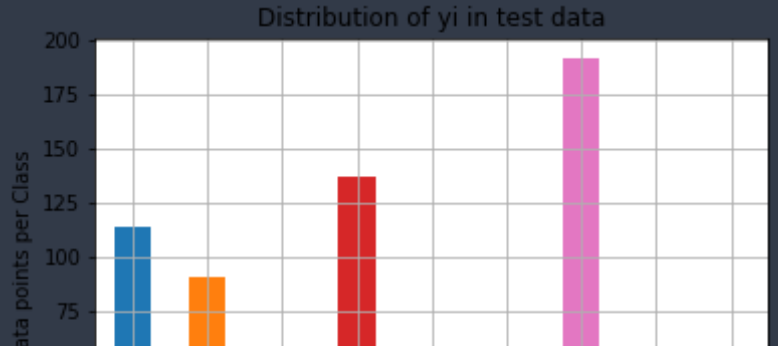
In [12]:

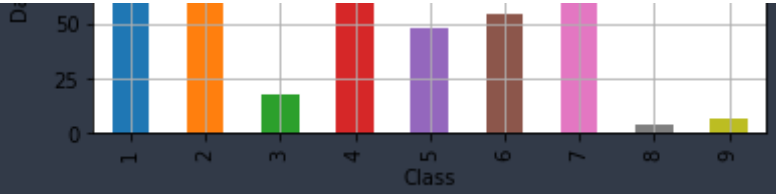
```
# it returns a dict, keys as class labels and values as the number of data points in tha
t class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


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

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

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

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

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

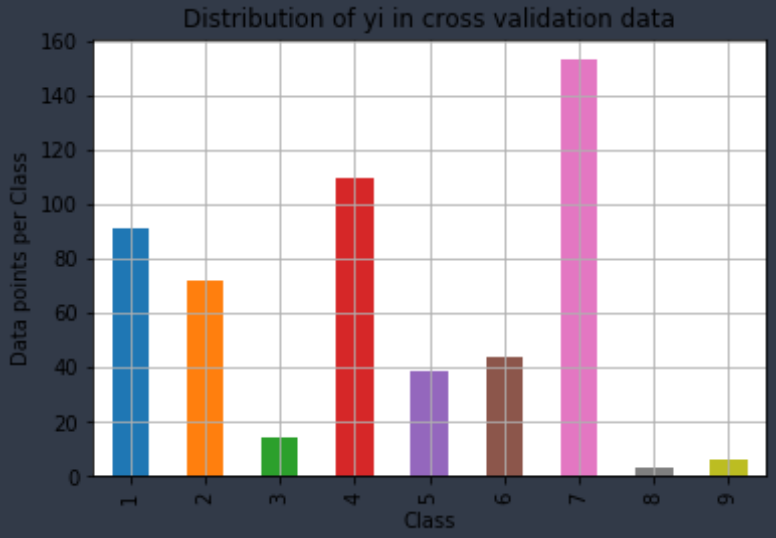



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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






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




```
Number of data points in class 7 : 153 ( 28.759 %)
```

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

## 3.2 Prediction using a 'Random' Model

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

In [13]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predict
ed 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 tw
o 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 tw
o diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

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

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

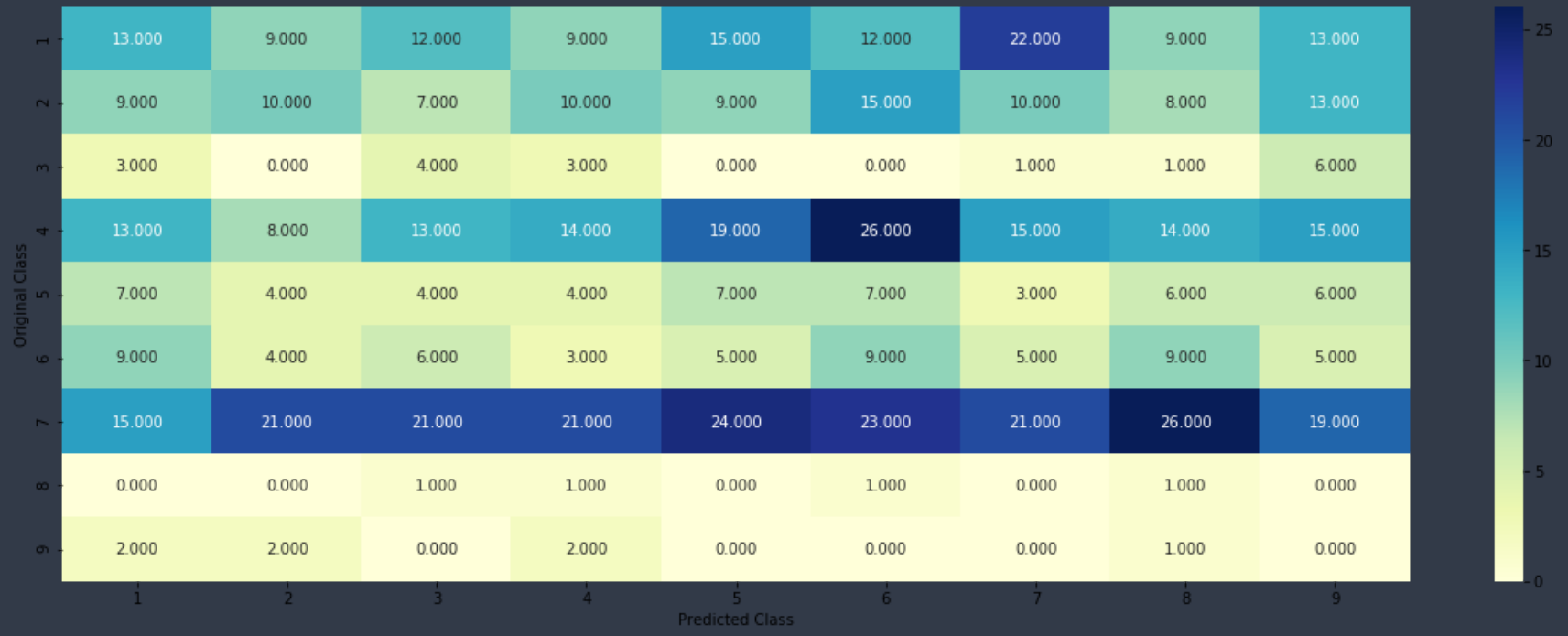
```
    plt.show()

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

In [14]:

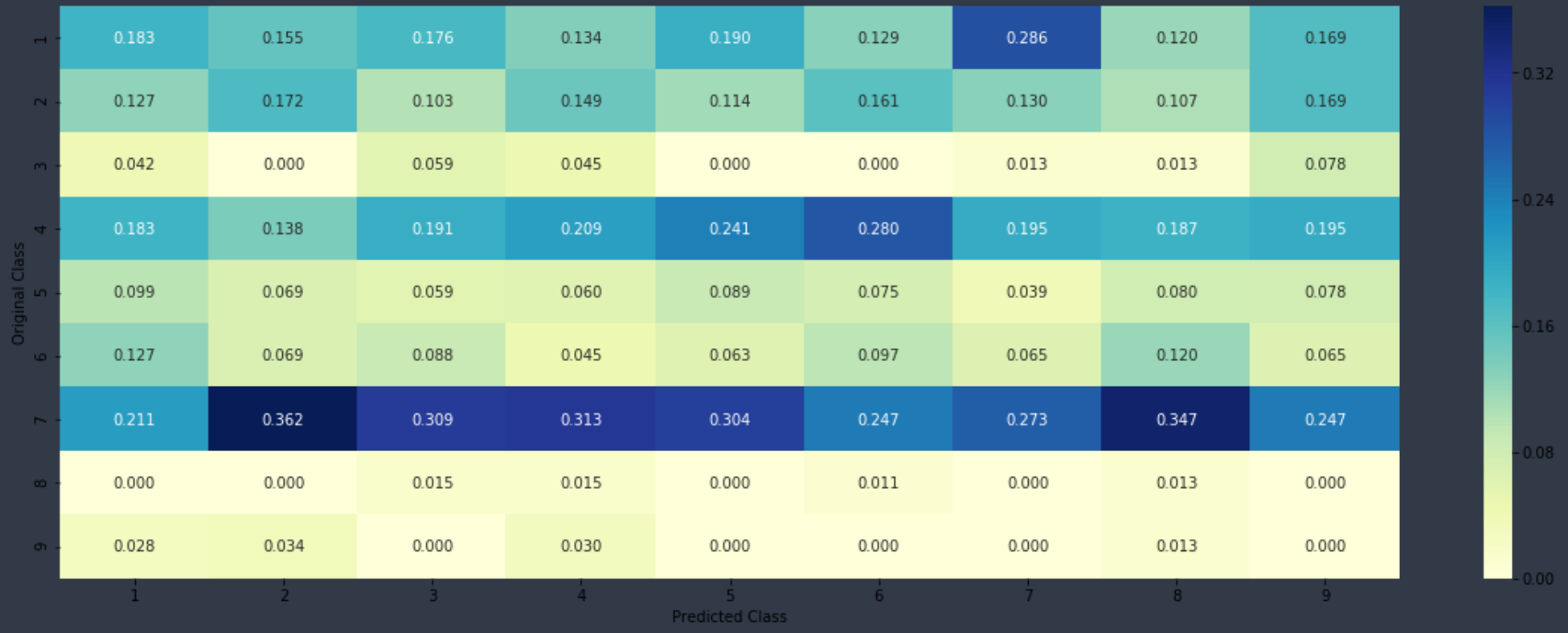
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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=1
e-15))

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

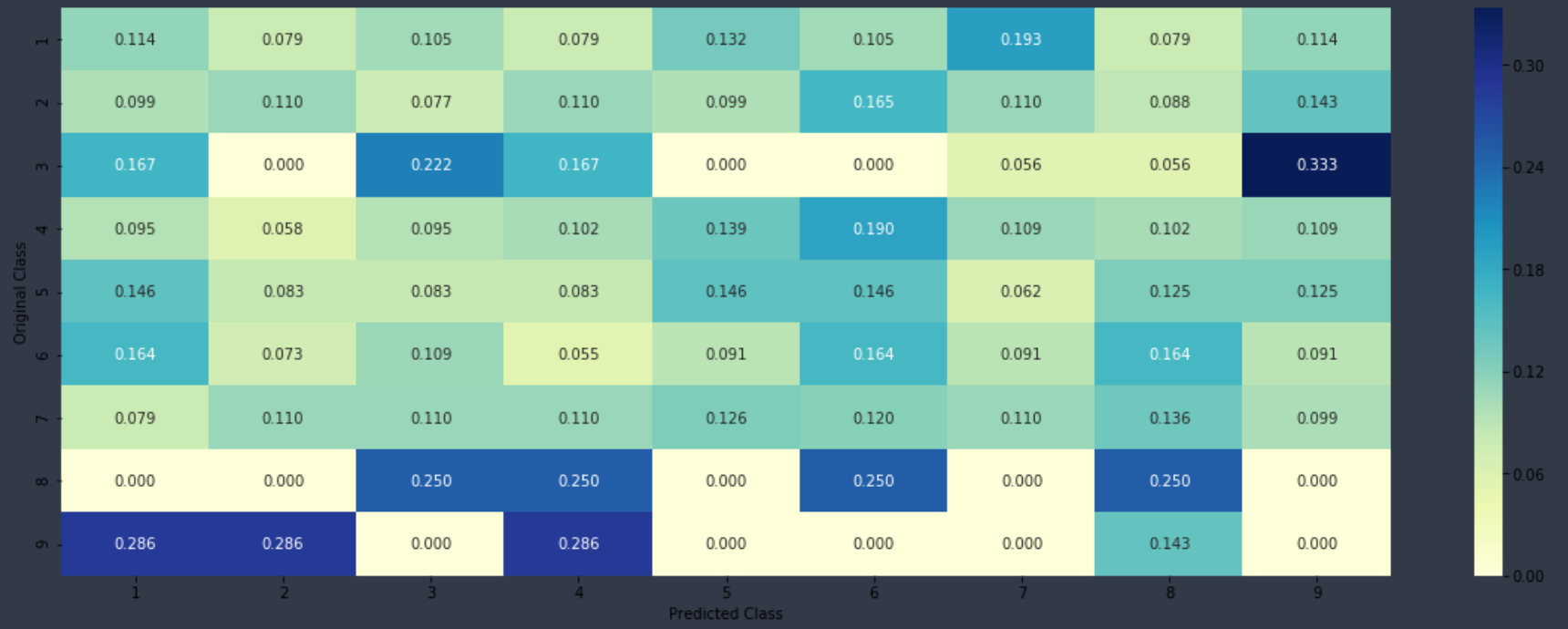
```
Log loss on Cross Validation Data using Random Model 2.477560048321117
Log loss on Test Data using Random Model 2.519542121101071
-------------------- Confusion matrix --------------------
```




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




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




## 3.3 Univariate Analysis

In [1]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data
 dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*
```

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1
3636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2
7040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.05
1020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818
177, 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, 0.060
606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.
46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.0
```

```
6289308176100628 9, 0.062893081761006289],
    #       'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07
2847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066
225165562913912, 0.066225165562913912],
    #       'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0
7333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.06
6666666666666666, 0.066666666666666666],
    #       ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value
 in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in t
he 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))
```

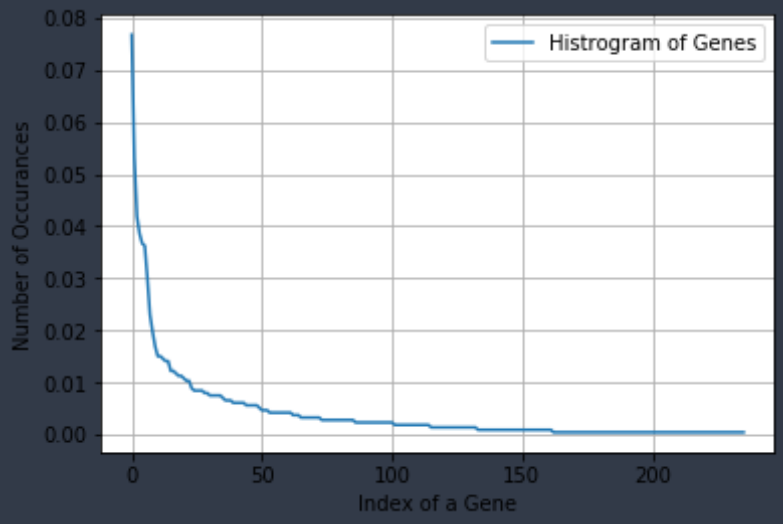
```
Number of Unique Genes : 236
BRCA1     163
TP53      113
EGFR       89
PTEN       82
KIT        78
BRCA2      77
BRAF       64
ERBB2      49
ALK        42
PDGFRA     36
Name: Gene, dtype: int64
```

In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the tra
in data, and they are distibuted as follows",)
```

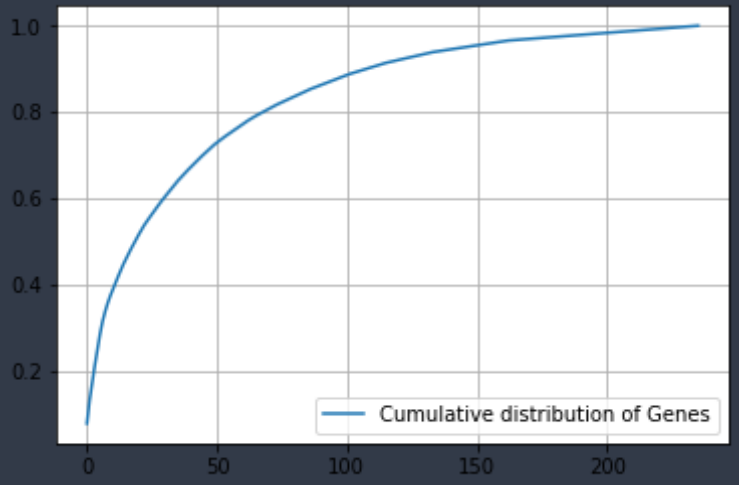
```
Ans: There are 236 different categories of genes in the train data, and they are distibuted as follows
```

In [18]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
```

```
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```




In [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```




**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 metho
d. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
train_df['Gene'].head()
```

```
1305      MLH1
2187      PTEN
742      ERBB2
42      DICER1
1628       VHL
Name: Gene, dtype: object
```

In [24]:
```
gene_vectorizer.get_feature_names()
```

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
```

```
            'araf',
            'arid2',
            'asxl1',
            'atm',
            'atrx',
            'aurka',
            'aurkb',
            'axl',
            'b2m',
            'bap1',
            'bard1',
            'bcl10',
            'bcl2',
            'bcor',
            'braf',
            'brca1',
            'brca2',
            'brd4',
            'brip1',
            'btk',
            'card11',
            'carm1',
            'casp8',
            'cbl',
            'ccnd1',
            'ccnd3',
            'ccne1',
            'cdh1',
            'cdk12',
            'cdk4',
            'cdk6',
            'cdk8',
            'cdkn1a',
            'cdkn1b',
            'cdkn2a',
            'cdkn2b',
            'cdkn2c',
            'cebpa',
            'chek2',
```

```
            'cic',
            'crebbp',
            'ctcf',
            'ctla4',
            'ctnnb1',
            'ddr2',
            'dicer1',
            'dnmt3a',
            'dnmt3b',
            'dusp4',
            'egfr',
            'eif1ax',
            'elf3',
            'ep300',
            'epas1',
            'epcam',
            'erbb2',
            'erbb3',
            'erbb4',
            'ercc2',
            'ercc4',
            'erg',
            'errfi1',
            'esr1',
            'etv1',
            'etv6',
            'ewsr1',
            'ezh2',
            'fam58a',
            'fanca',
            'fancc',
            'fat1',
            'fbxw7',
            'fgf3',
            'fgfr1',
            'fgfr2',
            'fgfr3',
            'fgfr4',
            'flt3',
```

```
            'foxa1',
            'foxl2',
            'foxo1',
            'foxp1',
            'fubp1',
            'gata3',
            'gna11',
            'gnas',
            'h3f3a',
            'hist1h1c',
            'hla',
            'hnf1a',
            'hras',
            'idh1',
            'idh2',
            'igf1r',
            'ikbke',
            'inpp4b',
            'jak1',
            'jak2',
            'jun',
            'kdm5a',
            'kdm5c',
            'kdm6a',
            'kdr',
            'keap1',
            'kit',
            'kmt2a',
            'kmt2b',
            'kmt2d',
            'knstrn',
            'kras',
            'lats2',
            'map2k1',
            'map2k2',
            'map2k4',
            'map3k1',
            'mapk1',
            'mdm2',
```

```
            'mdm4',
            'med12',
            'mef2b',
            'met',
            'mga',
            'mlh1',
            'mpl',
            'msh2',
            'msh6',
            'mtor',
            'myc',
            'mycn',
            'myd88',
            'myod1',
            'nf1',
            'nf2',
            'nfe2l2',
            'nfkbia',
            'nkx2',
            'notch1',
            'notch2',
            'npm1',
            'nras',
            'nsd1',
            'ntrk1',
            'ntrk2',
            'ntrk3',
            'nup93',
            'pak1',
            'pbrm1',
            'pdgfra',
            'pdgfrb',
            'pik3ca',
            'pik3cb',
            'pik3cd',
            'pik3r1',
            'pik3r2',
            'pim1',
            'pms1',
```

```
            'pms2',
            'pole',
            'ppm1d',
            'ppp2r1a',
            'ppp6c',
            'prdm1',
            'ptch1',
            'pten',
            'ptpn11',
            'ptprd',
            'ptprt',
            'rab35',
            'rac1',
            'rad21',
            'rad50',
            'rad51b',
            'rad51c',
            'rad51d',
            'rad54l',
            'raf1',
            'rara',
            'rasa1',
            'rb1',
            'rbm10',
            'ret',
            'rheb',
            'rhoa',
            'rictor',
            'rit1',
            'ros1',
            'rras2',
            'runx1',
            'rybp',
            'sdhb',
            'sdhc',
            'setd2',
            'sf3b1',
            'shoc2',
            'shq1',
```

```
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding metho
d. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 235)
```

**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.

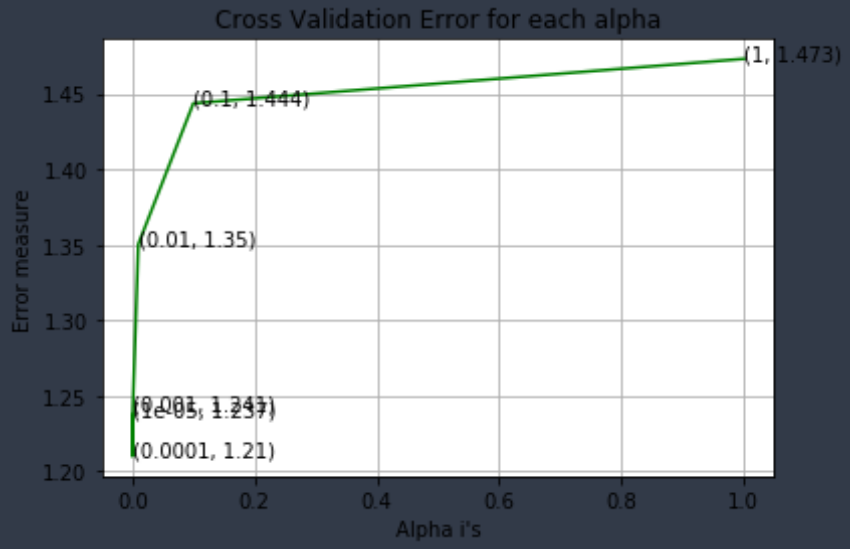
In [26]:

```
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/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t 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",cv=None)
    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, labe
ls=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
```

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

```
For values of alpha =  1e-05 The log loss is: 1.2369925559530384
For values of alpha =  0.0001 The log loss is: 1.2098300351744098
For values of alpha =  0.001 The log loss is: 1.2412133261558393
For values of alpha =  0.01 The log loss is: 1.3503540367994082
For values of alpha =  0.1 The log loss is: 1.4437025707475521
For values of alpha =  1 The log loss is: 1.473187251614279
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0115124036707572
For values of best alpha =  0.0001 The cross validation log loss is: 1.2098300351744098
For values of best alpha =  0.0001 The test log loss is: 1.161396676807217
```

**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.shape[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))
```

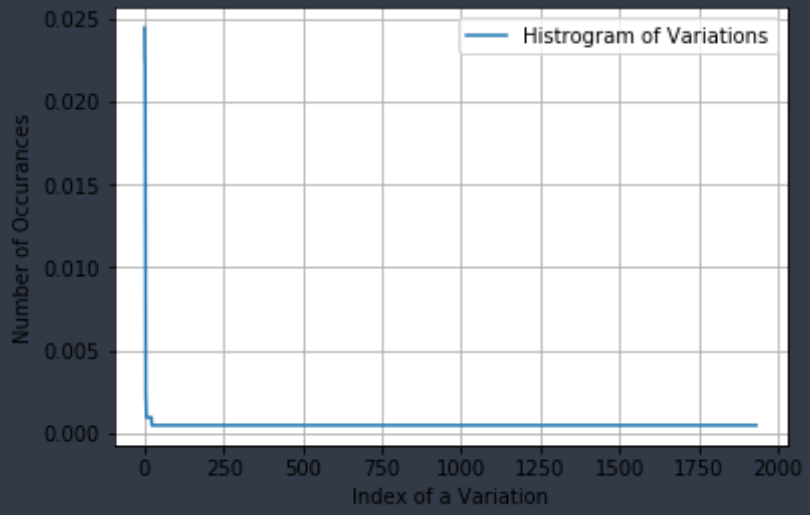
```
Number of Unique Variations : 1933
```

```
Truncating_Mutations    52
Amplification           48
Deletion                47
Fusions                 25
Overexpression           5
Q61H                     3
V321M                    2
K117N                    2
G12S                     2
Y64A                     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",)
```

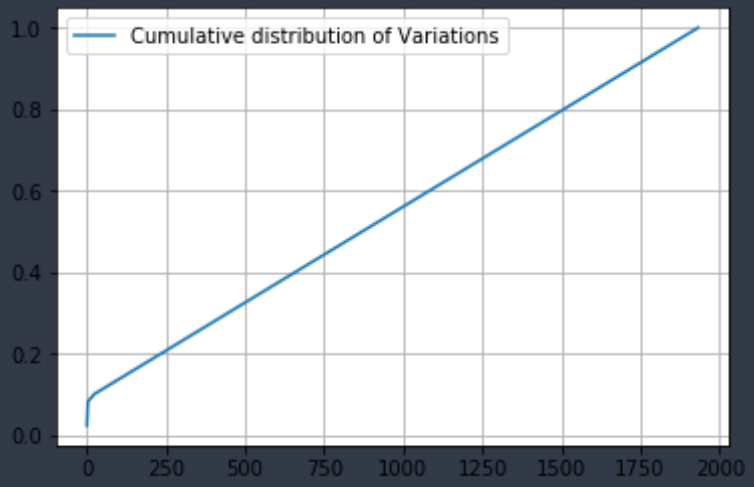
```
Ans: There are 1933 different categories of variations in the train data, and they are distibuted as follows
```

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()
```

```
[0.02448211 0.04708098 0.06920904 ... 0.99905838 0.99952919 1.        ]
```




**Q9.** How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tra
in_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 method. The shape of Variation feature:", train_variation_feature_responseCoding
.shape)
```

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

In [34]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Vari
ation'])
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 enc
oding method. The shape of Variation feature:", train_variation_feature_onehotCoding.sha
pe)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation featur
e: (2124, 1963)
```

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

Let's build a model just like the earlier!

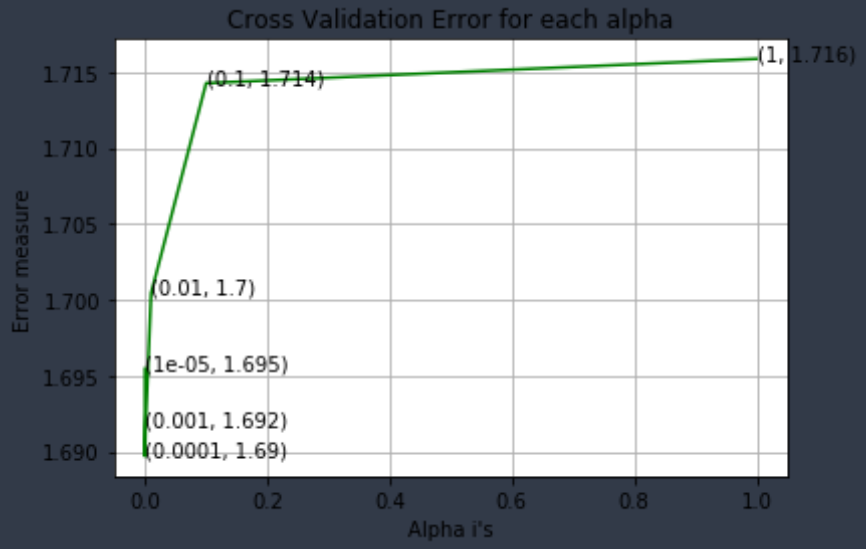
In [36]:
```
alpha = [10 ** x for x in range(-5, 1)]
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t 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",cv=None)
    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, labe
ls=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.6954679359434592
```

```
For values of alpha =  0.0001 The log loss is: 1.6896932116016865
For values of alpha =  0.001 The log loss is: 1.6916760253665526
For values of alpha =  0.01 The log loss is: 1.700435273546767
For values of alpha =  0.1 The log loss is: 1.7142689737644707
For values of alpha =  1 The log loss is: 1.715869585355236
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7728662447364719
For values of best alpha =  0.0001 The cross validation log loss is: 1.6896932116016865
For values of best alpha =  0.0001 The test log loss is: 1.7174958222247168
```

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

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

In [37]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].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_coverag
e/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cover
age/cv_df.shape[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 71 out of 665 : 10.676691729323307
2. In cross validation data 58 out of  532 : 10.902255639097744
```

## 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(wor
d,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'
].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [40]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train da
ta
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it o
ccured
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 : 54122
```

In [41]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [42]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
```

```
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [43]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feat
ure_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_respo
nseCoding.sum(axis=1)).T
```

In [44]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [45]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=T
rue))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [46]:
```
# Number of words for a given frequency.
```

```
print(Counter(sorted_text_occur))
```

```
Counter({3: 5721, 4: 3766, 5: 2888, 6: 2627, 7: 2249, 9: 2001, 8: 1908, 10: 1384, 12: 1160, 11: 1151, 15: 1069, 13: 1005, 1
6: 942, 14: 830, 18: 753, 19: 616, 20: 581, 17: 572, 21: 540, 22: 465, 24: 459, 30: 404, 27: 376, 40: 370, 25: 363, 28: 346,
23: 338, 29: 337, 26: 324, 33: 323, 32: 291, 36: 275, 35: 262, 34: 254, 55: 252, 31: 250, 44: 219, 42: 216, 39: 204, 45: 19
6, 37: 189, 48: 185, 38: 179, 41: 179, 46: 179, 60: 172, 50: 160, 47: 158, 49: 154, 43: 150, 56: 145, 54: 142, 51: 135, 52:
132, 63: 128, 53: 127, 72: 123, 65: 118, 70: 118, 57: 114, 61: 114, 62: 113, 58: 111, 66: 111, 64: 109, 59: 100, 80: 100, 8
8: 100, 69: 98, 67: 93, 77: 90, 73: 89, 71: 88, 74: 88, 78: 85, 76: 82, 79: 81, 68: 75, 81: 74, 75: 72, 86: 72, 87: 72, 84:
71, 83: 70, 91: 70, 90: 69, 96: 69, 113: 69, 82: 67, 85: 65, 100: 62, 95: 61, 110: 61, 101: 60, 120: 57, 92: 56, 94: 56, 98:
56, 102: 56, 104: 55, 99: 53, 106: 53, 107: 53, 97: 52, 112: 52, 111: 51, 132: 51, 114: 50, 93: 49, 89: 48, 127: 47, 105: 4
6, 108: 46, 109: 46, 115: 46, 128: 46, 146: 46, 130: 44, 121: 43, 140: 43, 116: 42, 123: 42, 144: 42, 135: 40, 118: 39, 103:
38, 138: 38, 117: 37, 122: 37, 126: 37, 143: 37, 145: 37, 134: 36, 137: 36, 152: 36, 124: 35, 150: 33, 159: 33, 155: 32, 15
6: 32, 162: 32, 147: 31, 131: 30, 136: 30, 141: 30, 161: 30, 125: 29, 139: 29, 194: 29, 201: 29, 210: 29, 119: 28, 133: 28,
148: 28, 149: 28, 160: 28, 163: 28, 170: 28, 174: 28, 177: 28, 193: 28, 129: 27, 181: 27, 188: 27, 211: 27, 168: 26, 190: 2
6, 142: 25, 151: 25, 153: 25, 165: 25, 166: 25, 173: 25, 186: 25, 200: 25, 164: 24, 171: 24, 154: 23, 172: 23, 175: 23, 176:
23, 182: 23, 183: 23, 187: 23, 204: 23, 205: 23, 208: 23, 215: 23, 233: 23, 180: 22, 197: 22, 192: 21, 209: 21, 212: 21, 22
3: 21, 224: 21, 293: 21, 178: 20, 216: 20, 220: 20, 242: 20, 250: 20, 271: 20, 158: 19, 179: 19, 238: 19, 240: 19, 257: 19,
157: 18, 167: 18, 202: 18, 236: 18, 239: 18, 241: 18, 246: 18, 304: 18, 196: 17, 203: 17, 219: 17, 229: 17, 249: 17, 260: 1
7, 261: 17, 263: 17, 301: 17, 313: 17, 185: 16, 189: 16, 191: 16, 198: 16, 206: 16, 213: 16, 214: 16, 226: 16, 235: 16, 237:
16, 280: 16, 287: 16, 308: 16, 199: 15, 207: 15, 217: 15, 222: 15, 227: 15, 232: 15, 248: 15, 268: 15, 278: 15, 282: 15, 31
1: 15, 354: 15, 297: 15, 169: 14, 230: 14, 243: 14, 244: 14, 247: 14, 252: 14, 264: 14, 276: 14, 298: 14, 312: 14, 323: 14,
361: 14, 218: 13, 253: 13, 274: 13, 275: 13, 288: 13, 289: 13, 291: 13, 324: 13, 327: 13, 331: 13, 340: 13, 384: 13, 195: 1
2, 231: 12, 234: 12, 245: 12, 256: 12, 258: 12, 279: 12, 299: 12, 318: 12, 341: 12, 346: 12, 351: 12, 364: 12, 390: 12, 405:
12, 458: 12, 184: 11, 221: 11, 225: 11, 254: 11, 259: 11, 262: 11, 284: 11, 285: 11, 319: 11, 328: 11, 358: 11, 371: 11, 22
8: 10, 265: 10, 267: 10, 272: 10, 277: 10, 283: 10, 286: 10, 292: 10, 294: 10, 296: 10, 320: 10, 347: 10, 366: 10, 374: 10,
394: 10, 429: 10, 450: 10, 255: 9, 269: 9, 273: 9, 281: 9, 295: 9, 300: 9, 303: 9, 305: 9, 309: 9, 321: 9, 322: 9, 326: 9, 3
36: 9, 342: 9, 344: 9, 383: 9, 415: 9, 417: 9, 440: 9, 448: 9, 463: 9, 465: 9, 468: 9, 473: 9, 251: 8, 266: 8, 270: 8, 302:
8, 314: 8, 330: 8, 332: 8, 349: 8, 352: 8, 356: 8, 357: 8, 360: 8, 365: 8, 369: 8, 377: 8, 380: 8, 382: 8, 385: 8, 389: 8, 3
93: 8, 396: 8, 414: 8, 419: 8, 433: 8, 435: 8, 446: 8, 456: 8, 466: 8, 485: 8, 492: 8, 505: 8, 526: 8, 623: 8, 679: 8, 306:
7, 307: 7, 325: 7, 329: 7, 333: 7, 343: 7, 348: 7, 359: 7, 372: 7, 373: 7, 375: 7, 387: 7, 388: 7, 391: 7, 395: 7, 399: 7, 4
03: 7, 413: 7, 420: 7, 428: 7, 460: 7, 462: 7, 464: 7, 481: 7, 488: 7, 516: 7, 524: 7, 530: 7, 594: 7, 605: 7, 660: 7, 916:
7, 315: 6, 316: 6, 317: 6, 334: 6, 338: 6, 350: 6, 355: 6, 376: 6, 381: 6, 400: 6, 421: 6, 425: 6, 442: 6, 447: 6, 449: 6, 4
53: 6, 457: 6, 471: 6, 472: 6, 480: 6, 482: 6, 484: 6, 487: 6, 496: 6, 500: 6, 515: 6, 517: 6, 522: 6, 525: 6, 529: 6, 535:
6, 553: 6, 564: 6, 572: 6, 582: 6, 587: 6, 593: 6, 598: 6, 608: 6, 613: 6, 629: 6, 630: 6, 641: 6, 658: 6, 662: 6, 665: 6, 6
80: 6, 737: 6, 827: 6, 839: 6, 951: 6, 290: 5, 310: 5, 339: 5, 345: 5, 353: 5, 363: 5, 370: 5, 378: 5, 386: 5, 392: 5, 401:
5, 402: 5, 406: 5, 408: 5, 410: 5, 411: 5, 412: 5, 418: 5, 427: 5, 434: 5, 437: 5, 438: 5, 451: 5, 455: 5, 474: 5, 475: 5, 4
90: 5, 494: 5, 498: 5, 507: 5, 511: 5, 512: 5, 514: 5, 518: 5, 520: 5, 521: 5, 523: 5, 528: 5, 536: 5, 541: 5, 545: 5, 549:
5, 551: 5, 558: 5, 562: 5, 563: 5, 565: 5, 571: 5, 575: 5, 577: 5, 585: 5, 596: 5, 607: 5, 616: 5, 625: 5, 627: 5, 628: 5, 6
51: 5, 659: 5, 678: 5, 687: 5, 690: 5, 697: 5, 700: 5, 703: 5, 706: 5, 708: 5, 711: 5, 713: 5, 717: 5, 743: 5, 783: 5, 799:
```

5, 803: 5, 818: 5, 822: 5, 870: 5, 911: 5, 934: 5, 1007: 5, 1075: 5, 1167: 5, 335: 4, 337: 4, 362: 4, 367: 4, 368: 4, 379:
4, 397: 4, 398: 4, 424: 4, 432: 4, 436: 4, 443: 4, 452: 4, 459: 4, 470: 4, 486: 4, 489: 4, 497: 4, 499: 4, 509: 4, 510: 4, 5
19: 4, 534: 4, 538: 4, 542: 4, 543: 4, 550: 4, 554: 4, 555: 4, 556: 4, 559: 4, 560: 4, 567: 4, 569: 4, 574: 4, 581: 4, 592:
4, 595: 4, 600: 4, 604: 4, 609: 4, 610: 4, 611: 4, 612: 4, 614: 4, 617: 4, 621: 4, 622: 4, 626: 4, 631: 4, 634: 4, 642: 4, 6
57: 4, 663: 4, 666: 4, 667: 4, 674: 4, 686: 4, 688: 4, 691: 4, 694: 4, 698: 4, 705: 4, 707: 4, 716: 4, 719: 4, 725: 4, 728:
4, 730: 4, 731: 4, 740: 4, 741: 4, 749: 4, 755: 4, 757: 4, 762: 4, 763: 4, 765: 4, 766: 4, 768: 4, 770: 4, 771: 4, 785: 4, 7
86: 4, 796: 4, 809: 4, 819: 4, 825: 4, 826: 4, 837: 4, 849: 4, 861: 4, 909: 4, 923: 4, 936: 4, 940: 4, 944: 4, 946: 4, 972:
4, 979: 4, 981: 4, 983: 4, 987: 4, 988: 4, 995: 4, 1057: 4, 1100: 4, 1115: 4, 1144: 4, 1152: 4, 1159: 4, 1168: 4, 1172: 4, 1
187: 4, 1207: 4, 1225: 4, 1245: 4, 1263: 4, 1275: 4, 1293: 4, 1313: 4, 1422: 4, 1430: 4, 1560: 4, 1580: 4, 1592: 4, 2462: 4,
441: 4, 601: 4, 404: 3, 422: 3, 423: 3, 426: 3, 430: 3, 431: 3, 444: 3, 445: 3, 461: 3, 467: 3, 478: 3, 495: 3, 501: 3, 503:
3, 504: 3, 527: 3, 533: 3, 537: 3, 540: 3, 544: 3, 548: 3, 552: 3, 557: 3, 566: 3, 570: 3, 580: 3, 584: 3, 588: 3, 589: 3, 5
91: 3, 606: 3, 624: 3, 632: 3, 639: 3, 644: 3, 653: 3, 655: 3, 664: 3, 668: 3, 677: 3, 681: 3, 692: 3, 701: 3, 702: 3, 715:
3, 718: 3, 724: 3, 729: 3, 735: 3, 739: 3, 746: 3, 748: 3, 751: 3, 764: 3, 769: 3, 773: 3, 774: 3, 780: 3, 782: 3, 789: 3, 7
90: 3, 794: 3, 811: 3, 812: 3, 813: 3, 814: 3, 816: 3, 817: 3, 823: 3, 829: 3, 831: 3, 835: 3, 836: 3, 838: 3, 844: 3, 856:
3, 858: 3, 874: 3, 882: 3, 887: 3, 893: 3, 894: 3, 898: 3, 901: 3, 902: 3, 905: 3, 910: 3, 913: 3, 914: 3, 922: 3, 924: 3, 9
31: 3, 943: 3, 961: 3, 974: 3, 980: 3, 989: 3, 992: 3, 997: 3, 1002: 3, 1016: 3, 1017: 3, 1021: 3, 1025: 3, 1026: 3, 1034:
3, 1039: 3, 1044: 3, 1071: 3, 1076: 3, 1083: 3, 1084: 3, 1103: 3, 1118: 3, 1128: 3, 1146: 3, 1165: 3, 1177: 3, 1197: 3, 119
8: 3, 1228: 3, 1268: 3, 1271: 3, 1286: 3, 1302: 3, 1310: 3, 1336: 3, 1343: 3, 1367: 3, 1369: 3, 1372: 3, 1393: 3, 1407: 3, 1
427: 3, 1442: 3, 1552: 3, 1554: 3, 1570: 3, 1582: 3, 1594: 3, 1612: 3, 1634: 3, 1636: 3, 1714: 3, 1754: 3, 1769: 3, 1770: 3,
1791: 3, 1795: 3, 1845: 3, 1875: 3, 1916: 3, 1977: 3, 1985: 3, 2138: 3, 2471: 3, 2531: 3, 2707: 3, 3183: 3, 3325: 3, 821: 3,
919: 3, 1319: 3, 407: 2, 409: 2, 416: 2, 439: 2, 454: 2, 469: 2, 476: 2, 479: 2, 483: 2, 491: 2, 508: 2, 513: 2, 531: 2, 54
6: 2, 547: 2, 568: 2, 573: 2, 578: 2, 583: 2, 586: 2, 597: 2, 599: 2, 602: 2, 603: 2, 618: 2, 619: 2, 635: 2, 637: 2, 638:
2, 640: 2, 643: 2, 645: 2, 646: 2, 648: 2, 650: 2, 652: 2, 654: 2, 661: 2, 675: 2, 676: 2, 696: 2, 704: 2, 709: 2, 714: 2, 7
20: 2, 721: 2, 722: 2, 723: 2, 726: 2, 736: 2, 738: 2, 742: 2, 744: 2, 752: 2, 753: 2, 754: 2, 756: 2, 758: 2, 759: 2, 760:
2, 767: 2, 772: 2, 775: 2, 776: 2, 788: 2, 793: 2, 800: 2, 802: 2, 804: 2, 807: 2, 808: 2, 815: 2, 832: 2, 833: 2, 834: 2, 8
42: 2, 845: 2, 850: 2, 851: 2, 852: 2, 853: 2, 863: 2, 866: 2, 867: 2, 869: 2, 872: 2, 873: 2, 879: 2, 880: 2, 883: 2, 885:
2, 888: 2, 889: 2, 891: 2, 892: 2, 895: 2, 899: 2, 903: 2, 918: 2, 920: 2, 921: 2, 928: 2, 929: 2, 948: 2, 950: 2, 954: 2, 9
57: 2, 958: 2, 962: 2, 968: 2, 973: 2, 978: 2, 982: 2, 991: 2, 998: 2, 1000: 2, 1001: 2, 1009: 2, 1012: 2, 1014: 2, 1015: 2,
1018: 2, 1019: 2, 1022: 2, 1023: 2, 1032: 2, 1048: 2, 1049: 2, 1051: 2, 1056: 2, 1060: 2, 1061: 2, 1062: 2, 1063: 2, 1065:
2, 1066: 2, 1067: 2, 1072: 2, 1080: 2, 1081: 2, 1085: 2, 1089: 2, 1092: 2, 1095: 2, 1096: 2, 1098: 2, 1102: 2, 1106: 2, 111
3: 2, 1120: 2, 1124: 2, 1125: 2, 1126: 2, 1127: 2, 1131: 2, 1142: 2, 1148: 2, 1149: 2, 1150: 2, 1154: 2, 1158: 2, 1161: 2, 9
355: 2, 1169: 2, 1170: 2, 1171: 2, 1173: 2, 1183: 2, 1186: 2, 1193: 2, 1204: 2, 1206: 2, 1209: 2, 1210: 2, 1211: 2, 1216: 2,
1221: 2, 1223: 2, 1224: 2, 1229: 2, 1235: 2, 1239: 2, 1242: 2, 1243: 2, 1249: 2, 1252: 2, 1254: 2, 1259: 2, 1261: 2, 1264:
2, 1265: 2, 1267: 2, 1269: 2, 1272: 2, 1280: 2, 1285: 2, 1298: 2, 1308: 2, 1321: 2, 1326: 2, 1334: 2, 1337: 2, 1338: 2, 133
9: 2, 1340: 2, 1344: 2, 1347: 2, 1348: 2, 1351: 2, 1353: 2, 1359: 2, 1360: 2, 1361: 2, 1379: 2, 1382: 2, 1389: 2, 1390: 2, 1
391: 2, 1401: 2, 1404: 2, 1423: 2, 1431: 2, 1432: 2, 1433: 2, 1437: 2, 1450: 2, 1452: 2, 1464: 2, 1466: 2, 1470: 2, 1478: 2,
1479: 2, 1480: 2, 1482: 2, 1487: 2, 1488: 2, 1489: 2, 1496: 2, 1498: 2, 1499: 2, 1500: 2, 1503: 2, 1506: 2, 1511: 2, 1512:
2, 1519: 2, 1523: 2, 1526: 2, 1527: 2, 1531: 2, 1533: 2, 1537: 2, 1538: 2, 1540: 2, 1550: 2, 1551: 2, 1555: 2, 1562: 2, 158
5: 2, 1593: 2, 1609: 2, 1617: 2, 1618: 2, 1619: 2, 1626: 2, 1656: 2, 1665: 2, 1675: 2, 1676: 2, 1684: 2, 1686: 2, 1687: 2, 1

693: 2, 1698: 2, 1710: 2, 1717: 2, 1721: 2, 1734: 2, 1739: 2, 1750: 2, 1753: 2, 1755: 2, 1766: 2, 1661: 2, 1779: 2, 1792: 2,
1801: 2, 1802: 2, 1803: 2, 1806: 2, 1816: 2, 1820: 2, 1823: 2, 1829: 2, 1844: 2, 1849: 2, 1859: 2, 1874: 2, 1876: 2, 1885:
2, 1898: 2, 1903: 2, 1923: 2, 1930: 2, 1939: 2, 1944: 2, 1963: 2, 1964: 2, 1983: 2, 1984: 2, 1989: 2, 2001: 2, 2011: 2, 201
2: 2, 2025: 2, 2043: 2, 2044: 2, 2068: 2, 2080: 2, 2088: 2, 2092: 2, 2106: 2, 2118: 2, 2126: 2, 2141: 2, 2150: 2, 2158: 2, 2
169: 2, 2191: 2, 2199: 2, 2243: 2, 2247: 2, 2248: 2, 2276: 2, 2277: 2, 2302: 2, 2304: 2, 2340: 2, 2344: 2, 2368: 2, 2375: 2,
2419: 2, 2426: 2, 1775: 2, 2463: 2, 2489: 2, 2490: 2, 2495: 2, 2501: 2, 2524: 2, 2529: 2, 2585: 2, 2594: 2, 2607: 2, 2657:
2, 2674: 2, 2693: 2, 2714: 2, 2721: 2, 2829: 2, 2866: 2, 2914: 2, 2932: 2, 2945: 2, 2973: 2, 2998: 2, 3005: 2, 3064: 2, 314
2: 2, 3159: 2, 3193: 2, 3250: 2, 3258: 2, 3276: 2, 3290: 2, 3312: 2, 3342: 2, 3376: 2, 3398: 2, 3419: 2, 3442: 2, 3501: 2, 3
604: 2, 3633: 2, 3650: 2, 3775: 2, 3776: 2, 3822: 2, 4012: 2, 4061: 2, 4095: 2, 4112: 2, 4324: 2, 4598: 2, 4599: 2, 4636: 2,
4879: 2, 5014: 2, 2227: 2, 9088: 2, 999: 2, 5506: 2, 5667: 2, 5721: 2, 5812: 2, 1027: 2, 1069: 2, 3812: 2, 1134: 2, 1190: 2,
7374: 2, 1266: 2, 1323: 2, 2705: 2, 1632: 2, 8489: 1, 8633: 1, 477: 1, 493: 1, 502: 1, 506: 1, 539: 1, 561: 1, 576: 1, 579:
1, 590: 1, 8292: 1, 615: 1, 633: 1, 25212: 1, 647: 1, 649: 1, 656: 1, 670: 1, 671: 1, 672: 1, 682: 1, 683: 1, 684: 1, 8877:
1, 693: 1, 695: 1, 699: 1, 710: 1, 712: 1, 727: 1, 734: 1, 8316: 1, 750: 1, 778: 1, 779: 1, 8976: 1, 787: 1, 791: 1, 795: 1,
797: 1, 798: 1, 805: 1, 810: 1, 820: 1, 9013: 1, 824: 1, 840: 1, 841: 1, 843: 1, 846: 1, 847: 1, 848: 1, 854: 1, 855: 1, 85
7: 1, 860: 1, 862: 1, 864: 1, 868: 1, 871: 1, 876: 1, 881: 1, 884: 1, 886: 1, 896: 1, 9089: 1, 900: 1, 906: 1, 908: 1, 912:
1, 915: 1, 917: 1, 17303: 1, 925: 1, 926: 1, 927: 1, 932: 1, 933: 1, 9127: 1, 937: 1, 941: 1, 942: 1, 945: 1, 949: 1, 952:
1, 953: 1, 956: 1, 959: 1, 960: 1, 963: 1, 965: 1, 966: 1, 970: 1, 975: 1, 976: 1, 977: 1, 984: 1, 986: 1, 990: 1, 993: 1, 9
96: 1, 17383: 1, 1003: 1, 1006: 1, 1011: 1, 9212: 1, 1024: 1, 17411: 1, 9220: 1, 1031: 1, 1036: 1, 1037: 1, 1038: 1, 1040:
1, 1041: 1, 1043: 1, 1045: 1, 1046: 1, 1047: 1, 1050: 1, 1052: 1, 1053: 1, 1055: 1, 1059: 1, 1064: 1, 1068: 1, 9261: 1, 107
0: 1, 1073: 1, 1074: 1, 9269: 1, 1086: 1, 1087: 1, 9280: 1, 1090: 1, 1101: 1, 1104: 1, 9297: 1, 1109: 1, 1110: 1, 1111: 1, 1
112: 1, 1116: 1, 1130: 1, 8221: 1, 1133: 1, 9326: 1, 1135: 1, 1136: 1, 1138: 1, 1140: 1, 1145: 1, 1147: 1, 1151: 1, 1153: 1,
1156: 1, 1157: 1, 1162: 1, 1166: 1, 1174: 1, 1176: 1, 1178: 1, 1179: 1, 1180: 1, 1181: 1, 1182: 1, 1184: 1, 1185: 1, 1188:
1, 1189: 1, 9382: 1, 1194: 1, 1200: 1, 1202: 1, 1205: 1, 1212: 1, 1213: 1, 1215: 1, 1218: 1, 1219: 1, 1220: 1, 17606: 1, 122
6: 1, 1227: 1, 1230: 1, 1231: 1, 1236: 1, 1238: 1, 1248: 1, 1255: 1, 1257: 1, 1260: 1, 66802: 1, 1270: 1, 1273: 1, 1276: 1,
1278: 1, 1279: 1, 1281: 1, 1282: 1, 1283: 1, 1284: 1, 1289: 1, 1291: 1, 1295: 1, 1297: 1, 1299: 1, 1304: 1, 1305: 1, 1306:
1, 1307: 1, 1311: 1, 1315: 1, 1316: 1, 1317: 1, 1318: 1, 9511: 1, 1322: 1, 17707: 1, 1324: 1, 1325: 1, 1327: 1, 1328: 1, 132
9: 1, 1330: 1, 1333: 1, 1335: 1, 1341: 1, 1342: 1, 1345: 1, 1346: 1, 9541: 1, 1350: 1, 1352: 1, 1355: 1, 9549: 1, 1358: 1, 1
362: 1, 1364: 1, 1365: 1, 1366: 1, 1368: 1, 1370: 1, 1371: 1, 1375: 1, 9568: 1, 1377: 1, 1378: 1, 1381: 1, 1384: 1, 1385: 1,
1387: 1, 1388: 1, 1396: 1, 1399: 1, 1400: 1, 1402: 1, 1403: 1, 1409: 1, 1412: 1, 1416: 1, 1417: 1, 1420: 1, 1424: 1, 1425:
1, 1428: 1, 1434: 1, 1438: 1, 1439: 1, 1440: 1, 1445: 1, 9640: 1, 1449: 1, 1451: 1, 1458: 1, 34227: 1, 1460: 1, 17846: 1, 14
65: 1, 1467: 1, 1468: 1, 1471: 1, 1473: 1, 34242: 1, 1475: 1, 1477: 1, 1481: 1, 1483: 1, 1484: 1, 9677: 1, 1486: 1, 8233: 1,
24600: 1, 1495: 1, 1497: 1, 1502: 1, 1505: 1, 1508: 1, 1510: 1, 1513: 1, 1520: 1, 1524: 1, 1528: 1, 1532: 1, 1539: 1, 50693:
1, 1543: 1, 1544: 1, 1545: 1, 1546: 1, 1547: 1, 42509: 1, 1558: 1, 1559: 1, 1561: 1, 1565: 1, 1567: 1, 1568: 1, 1627: 1, 157
4: 1, 1575: 1, 1577: 1, 1581: 1, 1583: 1, 17970: 1, 1588: 1, 1595: 1, 1597: 1, 1601: 1, 1602: 1, 1605: 1, 1606: 1, 1607: 1,
1608: 1, 1610: 1, 9805: 1, 1616: 1, 1621: 1, 1623: 1, 1624: 1, 8463: 1, 1628: 1, 1631: 1, 18016: 1, 1635: 1, 1637: 1, 1639:
1, 1641: 1, 1642: 1, 1647: 1, 1648: 1, 1649: 1, 1650: 1, 1653: 1, 42602: 1, 18047: 1, 1664: 1, 1670: 1, 1671: 1, 1673: 1, 16
74: 1, 1677: 1, 1678: 1, 1688: 1, 1690: 1, 1691: 1, 1694: 1, 1699: 1, 1702: 1, 1703: 1, 1704: 1, 1705: 1, 1713: 1, 1718: 1,
1719: 1, 1722: 1, 4384: 1, 1738: 1, 1740: 1, 1741: 1, 9939: 1, 1749: 1, 1756: 1, 1757: 1, 1758: 1, 1759: 1, 1762: 1, 1763:
1, 1767: 1, 1768: 1, 1778: 1, 1786: 1, 1787: 1, 1788: 1, 1799: 1, 1804: 1, 1807: 1, 1811: 1, 1819: 1, 1822: 1, 1826: 1, 183

0: 1, 1831: 1, 1832: 1, 1833: 1, 1834: 1, 1835: 1, 1841: 1, 1842: 1, 1843: 1, 10039: 1, 1848: 1, 18234: 1, 1854: 1, 1860: 1, 1861: 1, 1862: 1, 1863: 1, 1864: 1, 1867: 1, 1868: 1, 1869: 1, 1870: 1, 1871: 1, 10065: 1, 1878: 1, 10072: 1, 1881: 1, 1882: 1, 1884: 1, 1887: 1, 10080: 1, 1889: 1, 1892: 1, 1905: 1, 1907: 1, 1909: 1, 1910: 1, 18297: 1, 1914: 1, 10107: 1, 1918: 1, 8196: 1, 8512: 1, 1926: 1, 1929: 1, 1931: 1, 1933: 1, 1934: 1, 1942: 1, 1943: 1, 1951: 1, 1952: 1, 1955: 1, 1957: 1, 1958: 1, 1959: 1, 1960: 1, 1961: 1, 1967: 1, 10160: 1, 1970: 1, 1971: 1, 1973: 1, 1974: 1, 1975: 1, 1979: 1, 1980: 1, 1981: 1, 1987: 1, 10180: 1, 1997: 1, 2002: 1, 2004: 1, 2008: 1, 2013: 1, 2014: 1, 2015: 1, 2031: 1, 2033: 1, 2035: 1, 2036: 1, 2038: 1, 2040: 1, 2041: 1, 2045: 1, 2046: 1, 2047: 1, 2048: 1, 2049: 1, 2051: 1, 2055: 1, 2058: 1, 2059: 1, 2060: 1, 2061: 1, 2062: 1, 2065: 1, 2066: 1, 2071: 1, 2077: 1, 2078: 1, 18466: 1, 2083: 1, 2084: 1, 2086: 1, 2089: 1, 2091: 1, 9906: 1, 2094: 1, 2095: 1, 2096: 1, 2099: 1, 2101: 1, 10295: 1, 2110: 1, 2111: 1, 2124: 1, 2125: 1, 2127: 1, 2129: 1, 2131: 1, 2132: 1, 2133: 1, 2137: 1, 10331: 1, 2142: 1, 2143: 1, 2149: 1, 2151: 1, 10345: 1, 2155: 1, 9619: 1, 2160: 1, 2162: 1, 2166: 1, 2167: 1, 2171: 1, 10364: 1, 2173: 1, 2175: 1, 2181: 1, 2184: 1, 2189: 1, 2192: 1, 2194: 1, 2197: 1, 2198: 1, 2200: 1, 2201: 1, 2203: 1, 2206: 1, 2207: 1, 2209: 1, 10402: 1, 8257: 1, 2217: 1, 18604: 1, 2224: 1, 26803: 1, 2228: 1, 2231: 1, 2233: 1, 2234: 1, 26812: 1, 2237: 1, 2239: 1, 2240: 1, 2241: 1, 2244: 1, 2245: 1, 2250: 1, 2259: 1, 2260: 1, 2261: 1, 17806: 1, 2264: 1, 2265: 1, 2266: 1, 2267: 1, 2280: 1, 2282: 1, 2283: 1, 2292: 1, 2294: 1, 2299: 1, 2300: 1, 2301: 1, 2305: 1, 2306: 1, 2312: 1, 2319: 1, 2321: 1, 2323: 1, 10518: 1, 10520: 1, 2333: 1, 10533: 1, 2342: 1, 2345: 1, 2347: 1, 2351: 1, 2356: 1, 2358: 1, 2359: 1, 2360: 1, 2361: 1, 2366: 1, 2369: 1, 2371: 1, 2374: 1, 2376: 1, 2379: 1, 2382: 1, 2392: 1, 2393: 1, 2394: 1, 2396: 1, 2397: 1, 18782: 1, 2400: 1, 2402: 1, 2405: 1, 2406: 1, 2407: 1, 2408: 1, 2409: 1, 2411: 1, 2413: 1, 2414: 1, 2417: 1, 2425: 1, 10620: 1, 2429: 1, 27007: 1, 2434: 1, 2435: 1, 2437: 1, 2438: 1, 2440: 1, 2447: 1, 2453: 1, 2457: 1, 18845: 1, 2464: 1, 2469: 1, 2477: 1, 2478: 1, 2480: 1, 2482: 1, 2488: 1, 2493: 1, 2494: 1, 2496: 1, 2507: 1, 2508: 1, 16802: 1, 2513: 1, 2518: 1, 2522: 1, 2533: 1, 2535: 1, 2539: 1, 8616: 1, 2546: 1, 2549: 1, 2561: 1, 2562: 1, 2564: 1, 2570: 1, 2571: 1, 2575: 1, 10770: 1, 2579: 1, 2581: 1, 2584: 1, 10778: 1, 2590: 1, 2593: 1, 2595: 1, 10788: 1, 2600: 1, 2603: 1, 2606: 1, 68145: 1, 2611: 1, 2612: 1, 2622: 1, 8629: 1, 2625: 1, 2628: 1, 2631: 1, 2634: 1, 25016: 1, 2646: 1, 2649: 1, 2655: 1, 2658: 1, 2660: 1, 2661: 1, 10854: 1, 2666: 1, 2667: 1, 10861: 1, 2673: 1, 2677: 1, 2684: 1, 2687: 1, 2691: 1, 2699: 1, 2701: 1, 2702: 1, 2703: 1, 10897: 1, 2706: 1, 2711: 1, 2713: 1, 2715: 1, 2726: 1, 2727: 1, 2730: 1, 2732: 1, 2735: 1, 35508: 1, 35509: 1, 2743: 1, 2745: 1, 2756: 1, 2763: 1, 2765: 1, 10961: 1, 2771: 1, 2780: 1, 2782: 1, 2785: 1, 10978: 1, 2793: 1, 19178: 1, 2801: 1, 2807: 1, 2813: 1, 2820: 1, 27398: 1, 2826: 1, 2827: 1, 2830: 1, 2822: 1, 2835: 1, 2839: 1, 2849: 1, 2852: 1, 2853: 1, 2854: 1, 2864: 1, 2867: 1, 2870: 1, 2786: 1, 2882: 1, 2890: 1, 1847: 1, 2895: 1, 2897: 1, 2900: 1, 2901: 1, 2905: 1, 2907: 1, 1850: 1, 8677: 1, 2912: 1, 2920: 1, 2925: 1, 2927: 1, 27509: 1, 33257: 1, 2936: 1, 2939: 1, 2942: 1, 2943: 1, 8683: 1, 27525: 1, 1462: 1, 2953: 1, 2911: 1, 2960: 1, 2961: 1, 11156: 1, 2968: 1, 2975: 1, 19365: 1, 2988: 1, 2991: 1, 2992: 1, 2996: 1, 3007: 1, 3014: 1, 68551: 1, 3016: 1, 3017: 1, 3020: 1, 3031: 1, 3038: 1, 3043: 1, 11236: 1, 3048: 1, 3066: 1, 3075: 1, 3076: 1, 3082: 1, 3086: 1, 3088: 1, 3093: 1, 3097: 1, 3099: 1, 3102: 1, 3104: 1, 3107: 1, 3122: 1, 3127: 1, 3130: 1, 3131: 1, 3134: 1, 1888: 1, 19530: 1, 3154: 1, 11353: 1, 3165: 1, 3168: 1, 19558: 1, 3176: 1, 3179: 1, 3184: 1, 3187: 1, 3192: 1, 3194: 1, 9381: 1, 3202: 1, 3206: 1, 3219: 1, 3222: 1, 3223: 1, 3227: 1, 3229: 1, 3232: 1, 19621: 1, 3238: 1, 3239: 1, 3244: 1, 3249: 1, 3259: 1, 3260: 1, 3262: 1, 19651: 1, 3278: 1, 3284: 1, 3285: 1, 11478: 1, 3288: 1, 3293: 1, 8293: 1, 3297: 1, 1915: 1, 3310: 1, 10110: 1, 3333: 1, 3334: 1, 1921: 1, 3336: 1, 8260: 1, 3348: 1, 16942: 1, 3351: 1, 3353: 1, 3367: 1, 3372: 1, 3378: 1, 3383: 1, 3386: 1, 3295: 1, 3388: 1, 11581: 1, 3391: 1, 3393: 1, 3402: 1, 3404: 1, 3405: 1, 3414: 1, 8297: 1, 3416: 1, 3418: 1, 3422: 1, 3423: 1, 3426: 1, 3430: 1, 3437: 1, 3440: 1, 4155: 1, 8766: 1, 3446: 1, 3449: 1, 3458: 1, 3463: 1, 3465: 1, 3466: 1, 3472: 1, 3473: 1, 3475: 1, 3476: 1, 3483: 1, 3485: 1, 11688: 1, 3498: 1, 3500: 1, 3502: 1, 3516: 1, 3523: 1, 19916: 1, 3533: 1, 3534: 1, 3540: 1, 36309: 1, 19930: 1, 3548: 1, 3550: 1, 3554: 1, 3556: 1, 3560: 1, 3562: 1, 3566: 1, 3568: 1, 11762: 1, 3573: 1, 3576: 1, 3

581: 1, 3585: 1, 3586: 1, 18123: 1, 3595: 1, 3599: 1, 3610: 1, 3621: 1, 3625: 1, 10162: 1, 3638: 1, 3639: 1, 3644: 1, 3648:
1, 20035: 1, 3652: 1, 3653: 1, 3663: 1, 3674: 1, 11868: 1, 3679: 1, 11878: 1, 3687: 1, 3690: 1, 3695: 1, 3696: 1, 3703: 1, 3
707: 1, 3708: 1, 3711: 1, 3713: 1, 3714: 1, 3715: 1, 28292: 1, 3720: 1, 3721: 1, 3726: 1, 3737: 1, 3748: 1, 3753: 1, 3762:
1, 11958: 1, 3768: 1, 3772: 1, 3779: 1, 17014: 1, 3782: 1, 11981: 1, 3790: 1, 3795: 1, 3805: 1, 3807: 1, 12000: 1, 3809: 1,
3811: 1, 36580: 1, 10193: 1, 10194: 1, 3829: 1, 3833: 1, 3838: 1, 3843: 1, 3848: 1, 3849: 1, 3855: 1, 3865: 1, 1493: 1, 3665
4: 1, 3892: 1, 3893: 1, 3894: 1, 3910: 1, 3912: 1, 3915: 1, 3916: 1, 3918: 1, 3921: 1, 12117: 1, 3927: 1, 3931: 1, 3932: 1,
3958: 1, 3959: 1, 3962: 1, 3971: 1, 3976: 1, 1663: 1, 3982: 1, 12179: 1, 41050: 1, 12953: 1, 3997: 1, 3998: 1, 12203: 1, 401
3: 1, 8317: 1, 12213: 1, 4027: 1, 4034: 1, 4041: 1, 4047: 1, 4049: 1, 4056: 1, 4057: 1, 4067: 1, 12263: 1, 12272: 1, 8873:
1, 4089: 1, 4093: 1, 4102: 1, 4104: 1, 3415: 1, 4109: 1, 4110: 1, 685: 1, 4114: 1, 4122: 1, 4124: 1, 4128: 1, 4132: 1, 4134:
1, 4142: 1, 4151: 1, 118843: 1, 10250: 1, 4158: 1, 28737: 1, 4163: 1, 4171: 1, 4173: 1, 4179: 1, 4185: 1, 4188: 1, 4195: 1,
4196: 1, 4200: 1, 4201: 1, 12396: 1, 4206: 1, 45174: 1, 4225: 1, 4231: 1, 4242: 1, 4244: 1, 4245: 1, 4249: 1, 4255: 1, 1244
9: 1, 4259: 1, 4265: 1, 4272: 1, 4275: 1, 4276: 1, 4278: 1, 4284: 1, 12477: 1, 4299: 1, 4300: 1, 4304: 1, 12499: 1, 4311: 1,
4342: 1, 4345: 1, 4350: 1, 4351: 1, 4353: 1, 4357: 1, 12556: 1, 20751: 1, 4368: 1, 4371: 1, 4374: 1, 12574: 1, 4383: 1, 1257
6: 1, 4389: 1, 4393: 1, 4394: 1, 66978: 1, 20808: 1, 4435: 1, 12628: 1, 4450: 1, 4461: 1, 12655: 1, 745: 1, 4474: 1, 4476:
1, 12670: 1, 4479: 1, 4485: 1, 4491: 1, 4492: 1, 4493: 1, 4496: 1, 12689: 1, 4504: 1, 4505: 1, 12700: 1, 4528: 1, 12723: 1,
4532: 1, 12730: 1, 12732: 1, 4545: 1, 4563: 1, 4564: 1, 4568: 1, 20956: 1, 4577: 1, 12779: 1, 4594: 1, 4600: 1, 4609: 1, 116
92: 1, 4620: 1, 4626: 1, 4631: 1, 4640: 1, 4643: 1, 8966: 1, 4647: 1, 4649: 1, 4655: 1, 12848: 1, 4661: 1, 21047: 1, 4688:
1, 4694: 1, 4707: 1, 12902: 1, 9705: 1, 4742: 1, 4749: 1, 4761: 1, 4764: 1, 12973: 1, 4793: 1, 12987: 1, 4804: 1, 4805: 1, 1
1724: 1, 4815: 1, 4820: 1, 4830: 1, 4832: 1, 4848: 1, 4850: 1, 13050: 1, 13056: 1, 4877: 1, 4880: 1, 4896: 1, 21294: 1, 492
4: 1, 13118: 1, 4938: 1, 4940: 1, 4963: 1, 4970: 1, 4971: 1, 4973: 1, 4980: 1, 5000: 1, 5001: 1, 5002: 1, 5009: 1, 13205: 1,
9029: 1, 13235: 1, 5051: 1, 5056: 1, 5061: 1, 5079: 1, 5080: 1, 5096: 1, 5097: 1, 13291: 1, 5100: 1, 5104: 1, 5114: 1, 5124:
1, 5127: 1, 5128: 1, 5131: 1, 5139: 1, 5143: 1, 5157: 1, 5159: 1, 29742: 1, 5173: 1, 5176: 1, 5190: 1, 21347: 1, 4399: 1, 52
19: 1, 5226: 1, 5232: 1, 5236: 1, 5239: 1, 5240: 1, 5241: 1, 5255: 1, 5265: 1, 5270: 1, 13470: 1, 5294: 1, 26826: 1, 5314:
1, 5316: 1, 13520: 1, 13530: 1, 5346: 1, 13541: 1, 5356: 1, 5367: 1, 897: 1, 5395: 1, 17287: 1, 5427: 1, 5430: 1, 17293: 1,
8365: 1, 13650: 1, 5462: 1, 16549: 1, 5479: 1, 38265: 1, 5504: 1, 5508: 1, 13703: 1, 5519: 1, 21926: 1, 5557: 1, 1549: 1, 55
67: 1, 5583: 1, 5584: 1, 13778: 1, 5593: 1, 5606: 1, 935: 1, 5615: 1, 5616: 1, 5622: 1, 5626: 1, 5633: 1, 5649: 1, 5651: 1,
13865: 1, 5676: 1, 13871: 1, 22082: 1, 5701: 1, 13924: 1, 5734: 1, 5740: 1, 5786: 1, 5802: 1, 5803: 1, 5805: 1, 43293: 1, 58
10: 1, 5823: 1, 14020: 1, 46840: 1, 2341: 1, 5878: 1, 5880: 1, 5901: 1, 5903: 1, 5939: 1, 22330: 1, 5947: 1, 5949: 1, 5961:
1, 43321: 1, 5981: 1, 5992: 1, 14187: 1, 6003: 1, 9193: 1, 6040: 1, 6043: 1, 6057: 1, 6063: 1, 22468: 1, 6091: 1, 6101: 1, 1
020: 1, 6126: 1, 47090: 1, 6141: 1, 6149: 1, 50178: 1, 14352: 1, 6161: 1, 6167: 1, 1028: 1, 6175: 1, 6176: 1, 6181: 1, 6185:
1, 14403: 1, 6212: 1, 3766: 1, 6219: 1, 1571: 1, 17422: 1, 6232: 1, 6253: 1, 6261: 1, 14468: 1, 6308: 1, 6316: 1, 6321: 1, 1
4515: 1, 6353: 1, 6364: 1, 2428: 1, 6380: 1, 153850: 1, 6397: 1, 14620: 1, 11997: 1, 6450: 1, 6452: 1, 6456: 1, 6458: 1, 646
7: 1, 6468: 1, 39242: 1, 6479: 1, 6500: 1, 6501: 1, 3815: 1, 6513: 1, 42498: 1, 22900: 1, 6522: 1, 1088: 1, 6549: 1, 6553:
1, 2459: 1, 6567: 1, 6581: 1, 14780: 1, 14791: 1, 6604: 1, 6618: 1, 6620: 1, 6626: 1, 6633: 1, 6640: 1, 14841: 1, 6651: 1, 6
668: 1, 6671: 1, 6674: 1, 6691: 1, 63932: 1, 14962: 1, 6773: 1, 9773: 1, 1132: 1, 6799: 1, 6802: 1, 6804: 1, 6808: 1, 15013:
1, 6842: 1, 6844: 1, 6846: 1, 2509: 1, 6875: 1, 6878: 1, 6889: 1, 6895: 1, 2139: 1, 6902: 1, 6921: 1, 15127: 1, 6943: 1, 698
2: 1, 10723: 1, 15192: 1, 7018: 1, 7046: 1, 7051: 1, 80786: 1, 7061: 1, 7084: 1, 7087: 1, 33951: 1, 7110: 1, 7125: 1, 15326:
1, 7135: 1, 7153: 1, 7058: 1, 7179: 1, 7180: 1, 7197: 1, 8423: 1, 7226: 1, 7233: 1, 15439: 1, 15442: 1, 7252: 1, 7257: 1, 72
59: 1, 7275: 1, 7288: 1, 7311: 1, 7319: 1, 7323: 1, 2586: 1, 1222: 1, 10782: 1, 7354: 1, 7358: 1, 7369: 1, 2596: 1, 7404: 1,

```
15606: 1, 15645: 1, 7479: 1, 1613: 1, 7498: 1, 7511: 1, 15731: 1, 7544: 1, 7556: 1, 7584: 1, 7586: 1, 7606: 1, 2641: 1, 766
0: 1, 15857: 1, 40445: 1, 2647: 1, 7694: 1, 7696: 1, 56852: 1, 7701: 1, 7714: 1, 40484: 1, 24110: 1, 56203: 1, 7779: 1, 6513
3: 1, 24224: 1, 16034: 1, 32419: 1, 7876: 1, 7877: 1, 7882: 1, 9507: 1, 7916: 1, 7922: 1, 16118: 1, 7934: 1, 7950: 1, 7966:
1, 7993: 1, 7995: 1, 24383: 1, 16213: 1, 8033: 1, 8051: 1, 8053: 1, 16250: 1, 8060: 1, 19099: 1, 5564: 1, 8117: 1, 8120: 1,
8129: 1, 24520: 1, 8142: 1, 8147: 1, 8148: 1, 8156: 1, 8161: 1, 8166: 1, 5458: 1, 8184: 1, 8189: 1})
```

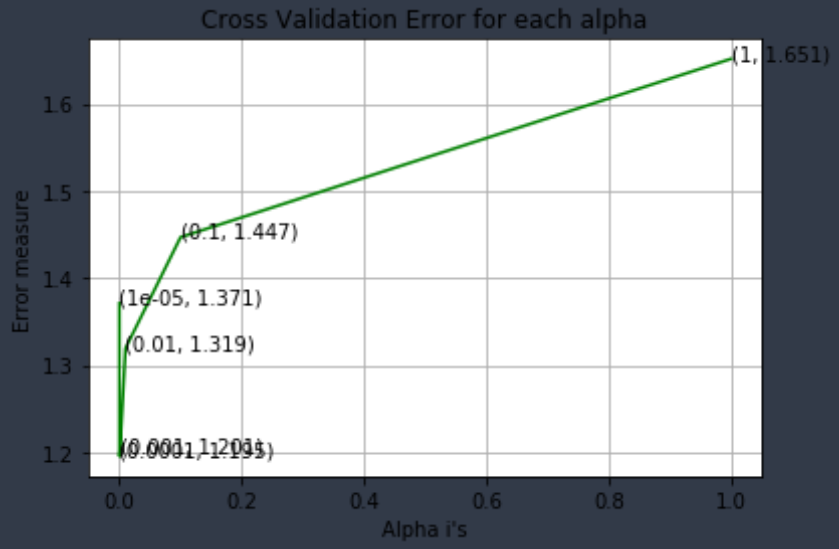
In [47]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t 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",cv=None)
    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, labe
ls=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
```

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

```
For values of alpha =  1e-05 The log loss is: 1.3708827096403124
For values of alpha =  0.0001 The log loss is: 1.1952055477059038
For values of alpha =  0.001 The log loss is: 1.2014094226724097
For values of alpha =  0.01 The log loss is: 1.3191684788067473
For values of alpha =  0.1 The log loss is: 1.4466843714587327
For values of alpha =  1 The log loss is: 1.6514503621481518
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7403367030256387
For values of best alpha =  0.0001 The cross validation log loss is: 1.1952055477059038
For values of best alpha =  0.0001 The test log loss is: 1.1568206246149109
```

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

**Ans.** Yes, it seems like!

In [48]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [49]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train dat
a")
```

```
96.466 % of word of test data appeared in train data
97.79 % of word of Cross Validation appeared in train data
```

# 4. Machine Learning Models

In [50]:
```
#Data preparation for ML models.

#Misc. functionns for ML models
```

```
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv =None)
    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 ea
ch class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y
.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [51]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [52]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)
```

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

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

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

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

## Stacking the three types of features

In [53]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_fe
ature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
re_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one
hotCoding))

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

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding
)).tocsr()
```

```
test_y = np.array(list(test_df['Class']))

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_varia
tion_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variatio
n_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feat
ure_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_re
sponseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_respo
nseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCod
ing))
```

In [54]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 56320)
```

```
(number of data points * number of features) in test data =  (665, 56320)
(number of data points * number of features) in cross validation data = (532, 56320)
```

In [55]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

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

# 4.1. Base Line Model

## 4.1.1. Naive Bayes

### 4.1.1.1. Hyper parameter tuning

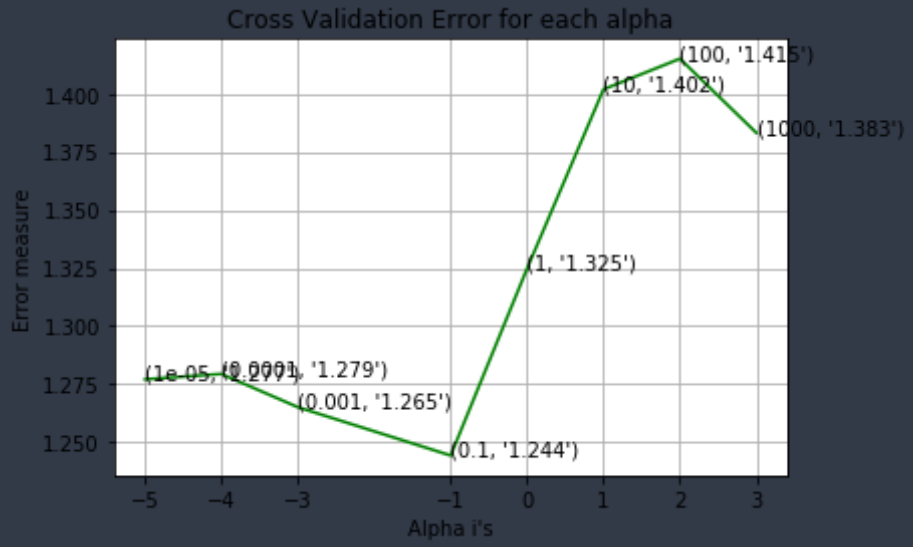
In [56]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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",cv=None)
    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 esti
mates
    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",cv=None)
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_los
s(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 i
s:",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 : 1.2770776594800948
for alpha = 0.0001
Log Loss : 1.279384160837367
for alpha = 0.001
Log Loss : 1.2650879755061992
for alpha = 0.1
Log Loss : 1.244273826698502
for alpha = 1
Log Loss : 1.3247750382370953
for alpha = 10
Log Loss : 1.4019245153917734
for alpha = 100
Log Loss : 1.4151140763632757
for alpha = 1000
Log Loss : 1.3831813821302377
```




```
For values of best alpha =  0.1 The train log loss is: 0.905931901065226
For values of best alpha =  0.1 The cross validation log loss is: 1.244273826698502
For values of best alpha =  0.1 The test log loss is: 1.2883410580414763
```

### 4.1.1.2. Testing the model with best hyper paramters

In [57]:

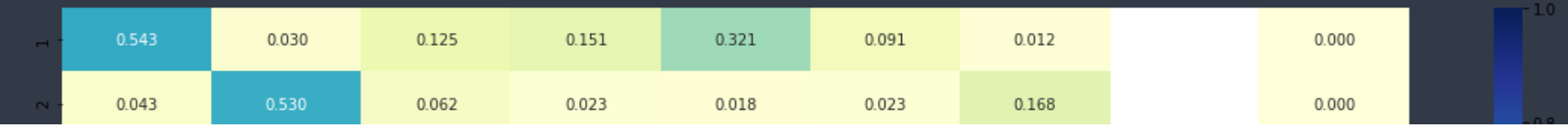
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
```

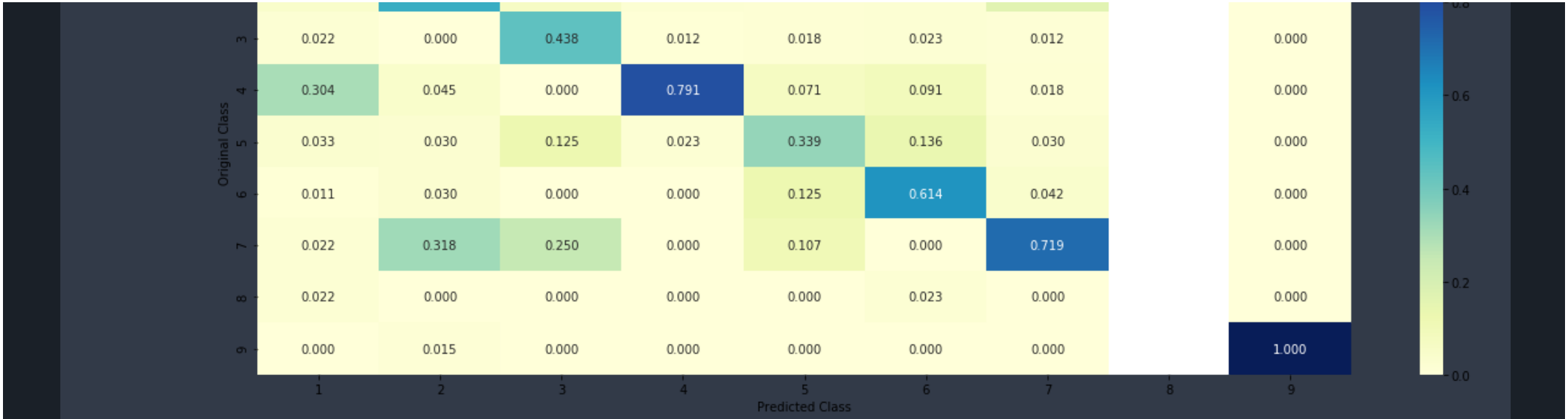
```
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


# 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",cv=None)
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_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

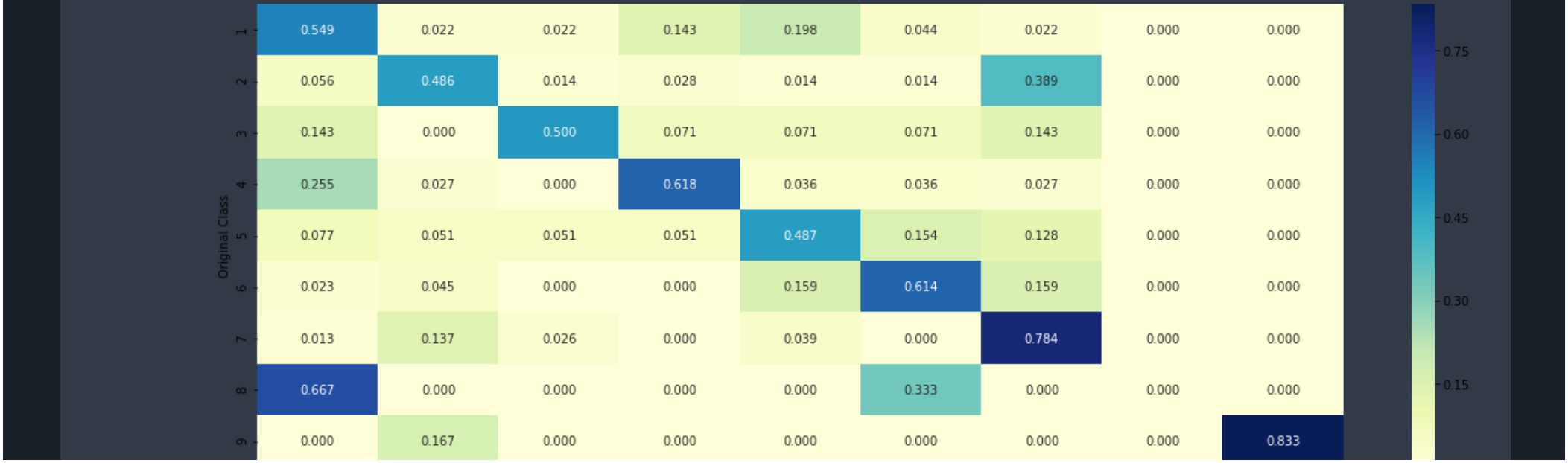
```
Log Loss : 1.244273826698502
Number of missclassified point : 0.37781954887218044
-------------------- Confusion matrix --------------------
```

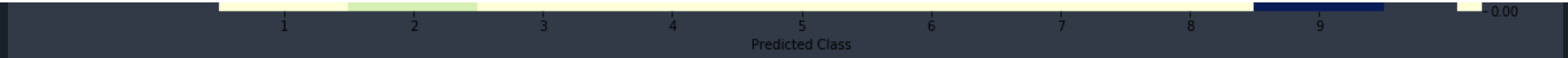

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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






### 4.1.1.3. Feature Importance, Correctly classified point

In [58]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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 : 4
Predicted Class Probabilities: [[0.2318 0.0712 0.0132 0.4797 0.0312 0.0332 0.1331 0.0047 0.002 ]]
Actual Class : 1
--------------------------------------------------
11 Text feature [function] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [mammalian] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [acid] present in test data point [True]
16 Text feature [functional] present in test data point [True]
17 Text feature [experiments] present in test data point [True]
18 Text feature [proteins] present in test data point [True]
19 Text feature [activity] present in test data point [True]
20 Text feature [amino] present in test data point [True]
22 Text feature [results] present in test data point [True]
25 Text feature [critical] present in test data point [True]

27 Text feature [ability] present in test data point [True]
```

```
29 Text feature [type] present in test data point [True]
30 Text feature [whereas] present in test data point [True]
32 Text feature [terminal] present in test data point [True]
33 Text feature [determined] present in test data point [True]
34 Text feature [phosphatase] present in test data point [True]
35 Text feature [indicated] present in test data point [True]
36 Text feature [suppressor] present in test data point [True]
38 Text feature [wild] present in test data point [True]
40 Text feature [affect] present in test data point [True]
41 Text feature [transfection] present in test data point [True]
42 Text feature [indicate] present in test data point [True]
43 Text feature [transfected] present in test data point [True]
48 Text feature [co] present in test data point [True]
52 Text feature [purified] present in test data point [True]
56 Text feature [vivo] present in test data point [True]
58 Text feature [related] present in test data point [True]
59 Text feature [determine] present in test data point [True]
60 Text feature [whether] present in test data point [True]
61 Text feature [bind] present in test data point [True]
63 Text feature [hcl] present in test data point [True]
64 Text feature [buffer] present in test data point [True]
69 Text feature [containing] present in test data point [True]
71 Text feature [two] present in test data point [True]
72 Text feature [mm] present in test data point [True]
73 Text feature [vector] present in test data point [True]
74 Text feature [effect] present in test data point [True]
75 Text feature [assay] present in test data point [True]
76 Text feature [tris] present in test data point [True]
77 Text feature [shown] present in test data point [True]
78 Text feature [cannot] present in test data point [True]
79 Text feature [microscopy] present in test data point [True]
81 Text feature [although] present in test data point [True]
82 Text feature [see] present in test data point [True]
83 Text feature [three] present in test data point [True]
85 Text feature [sds] present in test data point [True]
86 Text feature [system] present in test data point [True]
87 Text feature [either] present in test data point [True]
88 Text feature [therefore] present in test data point [True]

90 Text feature [nacl] present in test data point [True]
```

```
92 Text feature [putative] present in test data point [True]
93 Text feature [predicted] present in test data point [True]
94 Text feature [acids] present in test data point [True]
97 Text feature [tested] present in test data point [True]
99 Text feature [substitutions] present in test data point [True]
Out of the top  100  features  57 are present in query point
```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:
```
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_onehotCodi
ng[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: [[0.0773 0.0721 0.0132 0.109  0.0315 0.0332 0.6571 0.0047 0.002 ]]
Actual Class : 7
--------------------------------------------------
15 Text feature [kinase] present in test data point [True]
16 Text feature [activating] present in test data point [True]
17 Text feature [presence] present in test data point [True]
20 Text feature [inhibitor] present in test data point [True]
23 Text feature [treated] present in test data point [True]
24 Text feature [activation] present in test data point [True]
25 Text feature [however] present in test data point [True]
26 Text feature [shown] present in test data point [True]
27 Text feature [independent] present in test data point [True]
28 Text feature [expressing] present in test data point [True]
```

```
29 Text feature [also] present in test data point [True]
30 Text feature [showed] present in test data point [True]
33 Text feature [well] present in test data point [True]
34 Text feature [found] present in test data point [True]
35 Text feature [factor] present in test data point [True]
36 Text feature [potential] present in test data point [True]
37 Text feature [cell] present in test data point [True]
38 Text feature [previously] present in test data point [True]
39 Text feature [cells] present in test data point [True]
40 Text feature [10] present in test data point [True]
41 Text feature [compared] present in test data point [True]
42 Text feature [obtained] present in test data point [True]
43 Text feature [studies] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [addition] present in test data point [True]
46 Text feature [similar] present in test data point [True]
47 Text feature [phosphorylation] present in test data point [True]
48 Text feature [growth] present in test data point [True]
50 Text feature [may] present in test data point [True]
52 Text feature [1a] present in test data point [True]
53 Text feature [total] present in test data point [True]
55 Text feature [approved] present in test data point [True]
56 Text feature [12] present in test data point [True]
57 Text feature [observed] present in test data point [True]
58 Text feature [inhibition] present in test data point [True]
59 Text feature [described] present in test data point [True]
63 Text feature [reported] present in test data point [True]
65 Text feature [concentrations] present in test data point [True]
66 Text feature [identified] present in test data point [True]
67 Text feature [various] present in test data point [True]
69 Text feature [mutation] present in test data point [True]
70 Text feature [interestingly] present in test data point [True]
72 Text feature [including] present in test data point [True]
75 Text feature [although] present in test data point [True]
77 Text feature [either] present in test data point [True]
79 Text feature [sensitive] present in test data point [True]
80 Text feature [using] present in test data point [True]
84 Text feature [report] present in test data point [True]
85 Text feature [15] present in test data point [True]
```

```
86 Text feature [three] present in test data point [True]
87 Text feature [recent] present in test data point [True]
89 Text feature [figure] present in test data point [True]
90 Text feature [25] present in test data point [True]
93 Text feature [show] present in test data point [True]
94 Text feature [small] present in test data point [True]
95 Text feature [enhanced] present in test data point [True]
97 Text feature [occur] present in test data point [True]
98 Text feature [two] present in test data point [True]
99 Text feature [hours] present in test data point [True]
Out of the top  100  features  59 are present in query point
```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

In [60]:

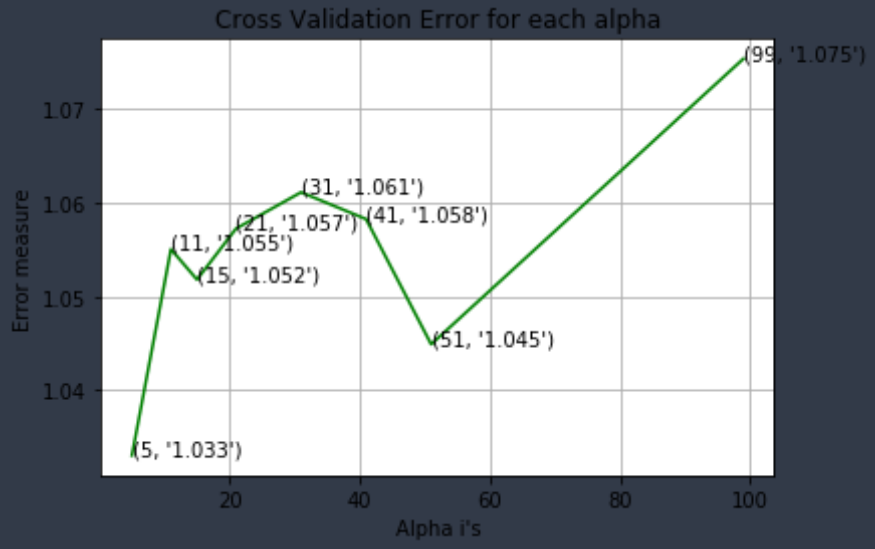
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
 p=2,
# metric='minkowski',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

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

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    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",cv=None)
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_los
s(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 i
s:",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 : 1.033054534454444
for alpha = 11
Log Loss : 1.0550532057037931
for alpha = 15
Log Loss : 1.0518099583483347
for alpha = 21
Log Loss : 1.0572404734914724
for alpha = 31
Log Loss : 1.0611182618836492
for alpha = 41
Log Loss : 1.0582569782885856
for alpha = 51
Log Loss : 1.04491076715046
for alpha = 99
Log Loss : 1.0753611320852825
```



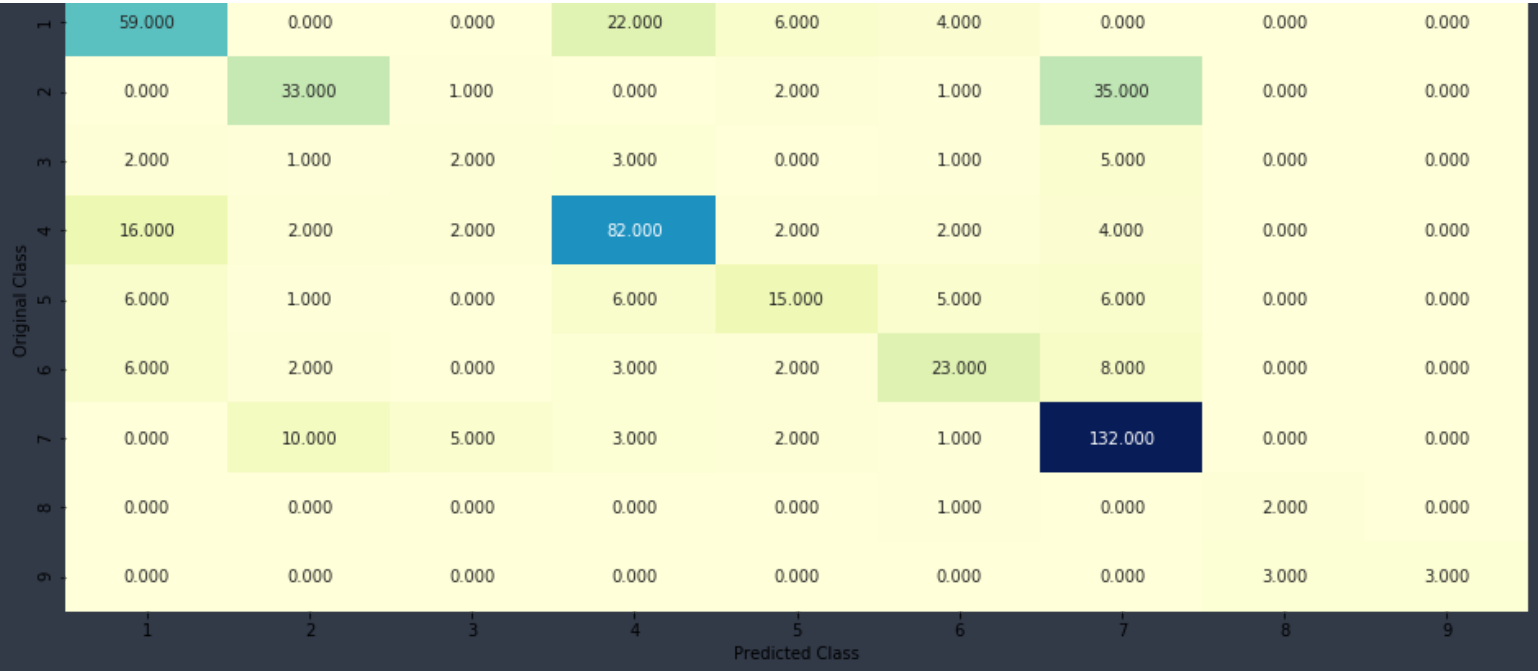
```
For values of best alpha =  5 The train log loss is: 0.49246298354621826
For values of best alpha =  5 The cross validation log loss is: 1.033054534454444
For values of best alpha =  5 The test log loss is: 1.0185157649358427
```

## 4.2.2. Testing the model with best hyper paramters

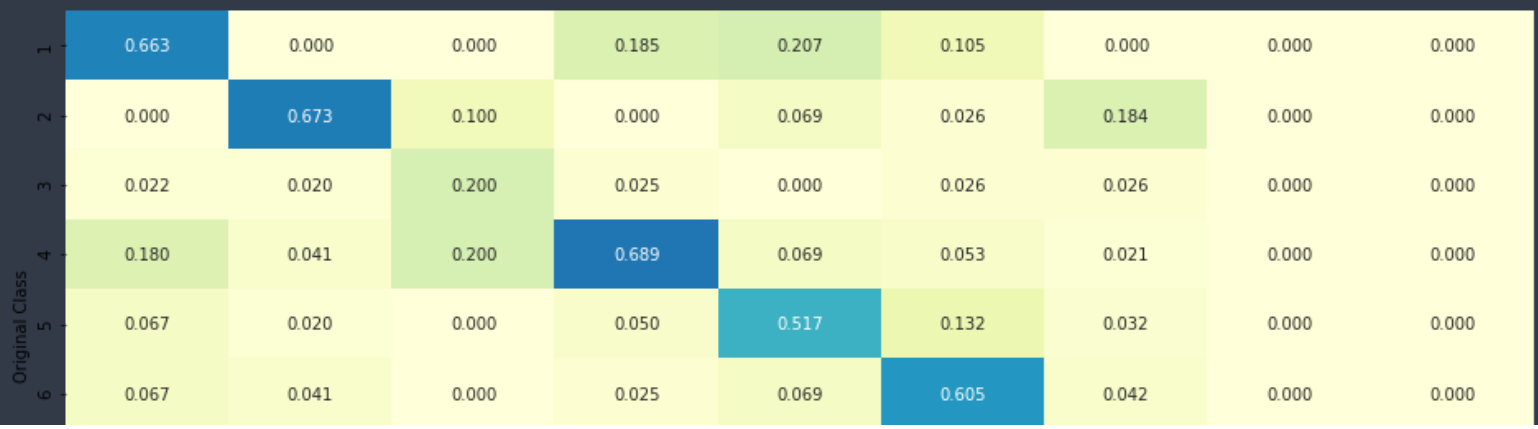
In [61]:

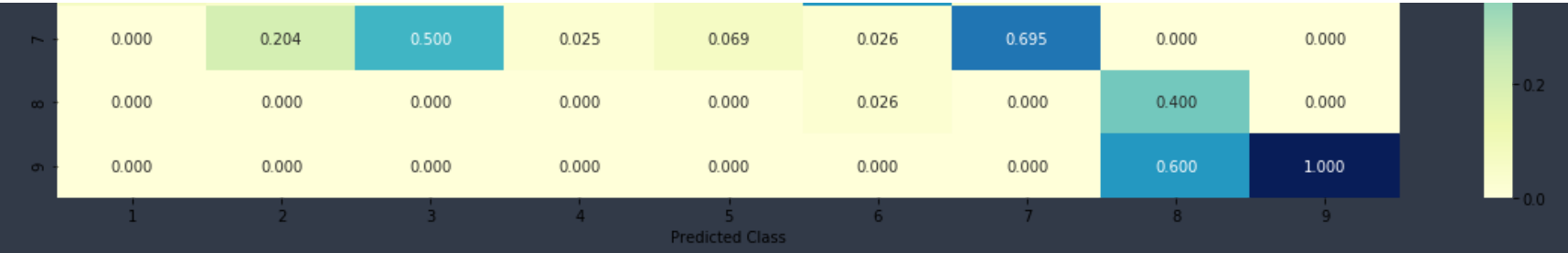
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
 p=2,
# metric='minkowski',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

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

```
Log loss : 1.033054534454444
Number of mis-classified points : 0.3402255639097744
------------------- Confusion matrix -------------------
```

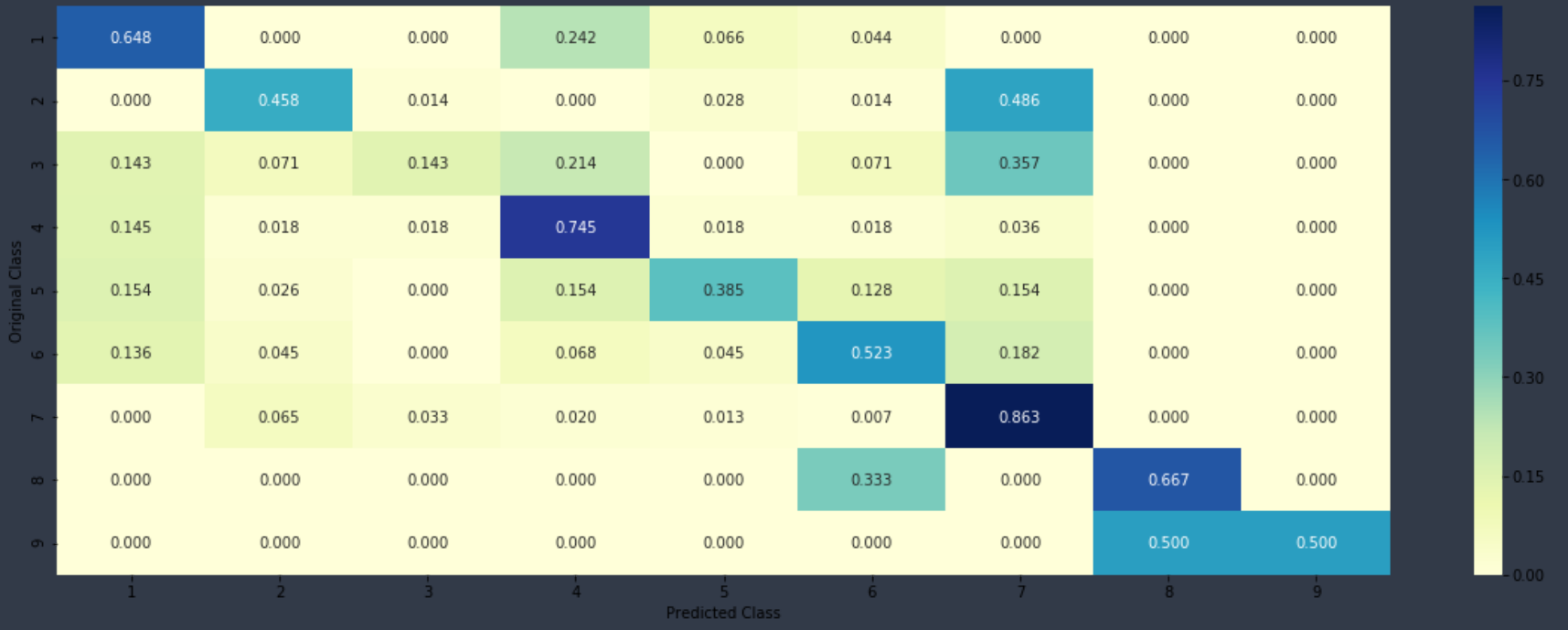



---------- Precision matrix (Columm Sum=1) ----------






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




## 4.2.3.Sample Query point -1

In [62]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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 classe
s",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 1
The  5  nearest neighbours of the test points belongs to classes [4 4 1 1 1]
Fequency of nearest points : Counter({1: 3, 4: 2})
```

### 4.2.4. Sample Query Point-2

In [63]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv = None)
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
```

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [7 7 7 7 2]
Fequency of nearest points : Counter({7: 4, 2: 1})
```

# 4.3. Logistic Regression

## 4.3.1. With Class balancing

### 4.3.1.1. Hyper paramter tuning

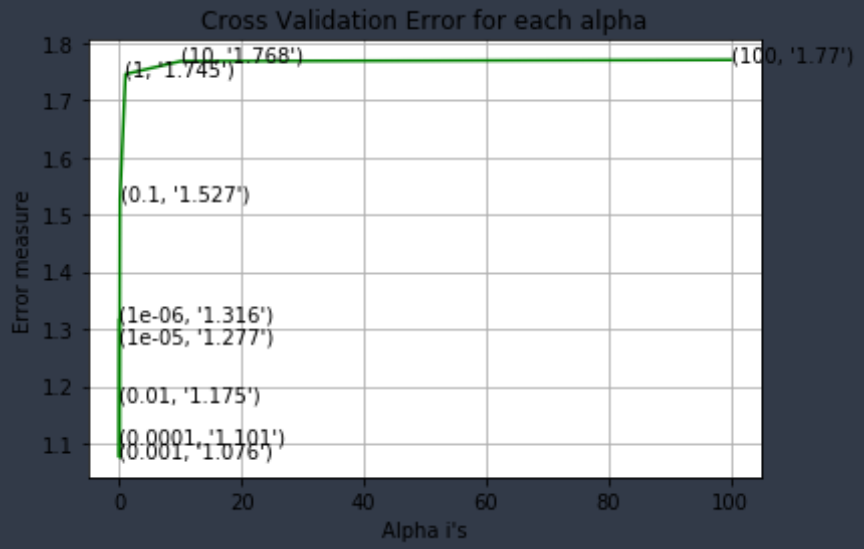
In [64]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/ge
ometric-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', rand
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",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 : 1.3158696399509813
for alpha = 1e-05
Log Loss : 1.2772585138745403
for alpha = 0.0001
Log Loss : 1.101071012631821
for alpha = 0.001
Log Loss : 1.0760042491515471
for alpha = 0.01
Log Loss : 1.1745641920274856
for alpha = 0.1
Log Loss : 1.5273458793318677
for alpha = 1
Log Loss : 1.745047099998296
for alpha = 10
Log Loss : 1.7675334745788802
for alpha = 100
Log Loss : 1.7699793145970224
```




```
For values of best alpha =  0.001 The train log loss is: 0.5735563847733652
For values of best alpha =  0.001 The cross validation log loss is: 1.0760042491515471
For values of best alpha =  0.001 The test log loss is: 1.0505131650688446
```

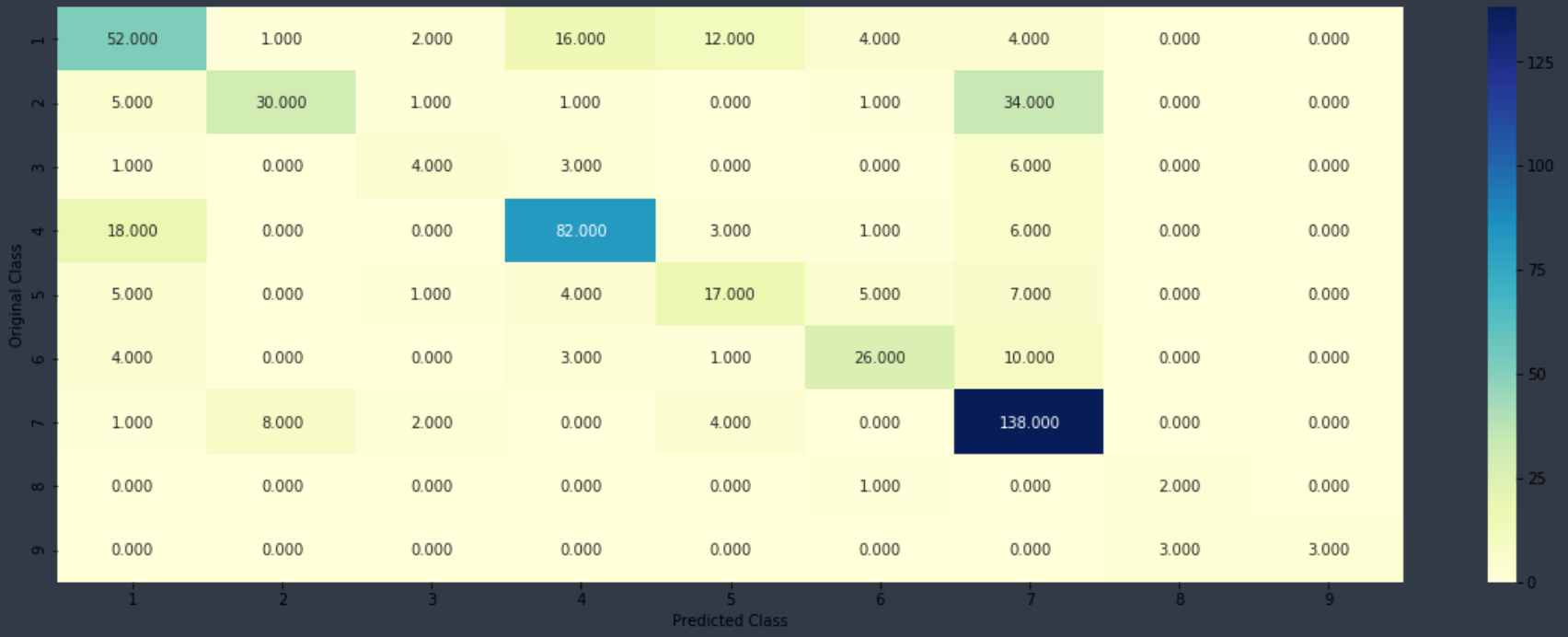
### 4.3.1.2. Testing the model with best hyper paramters

In [65]:

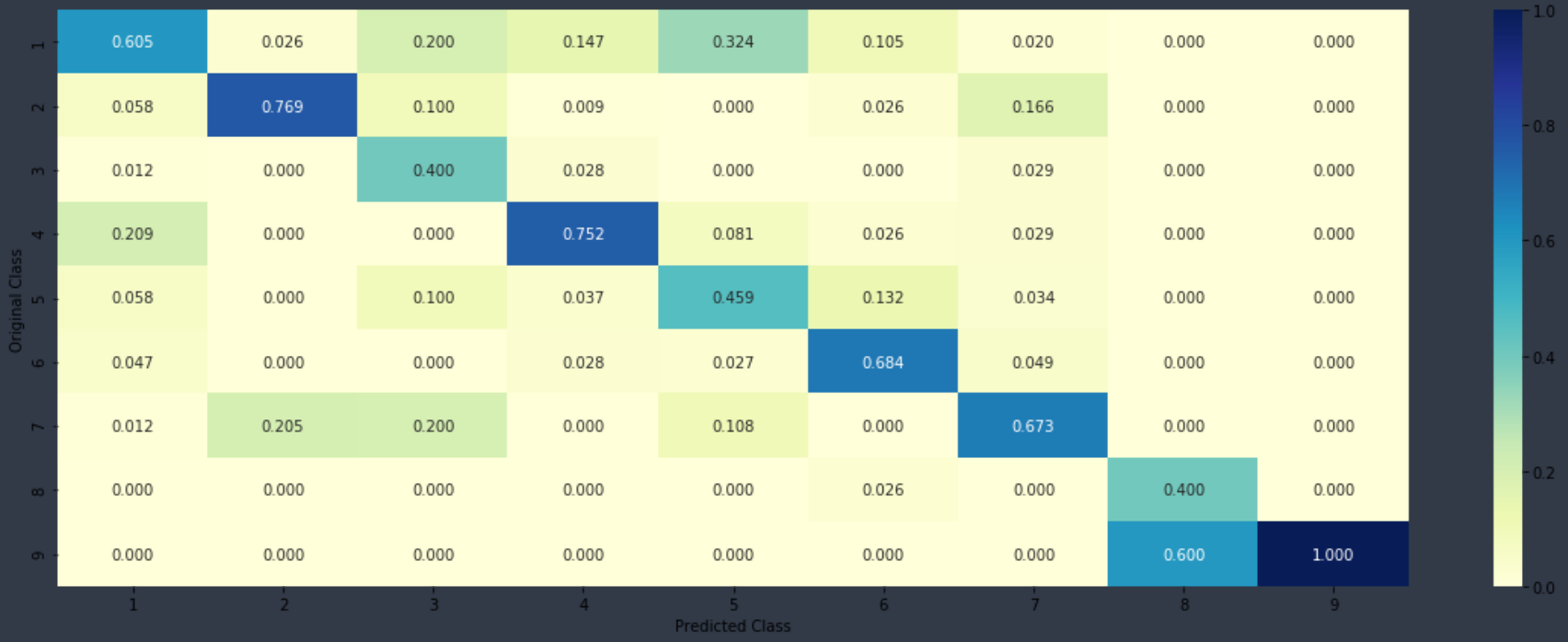
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

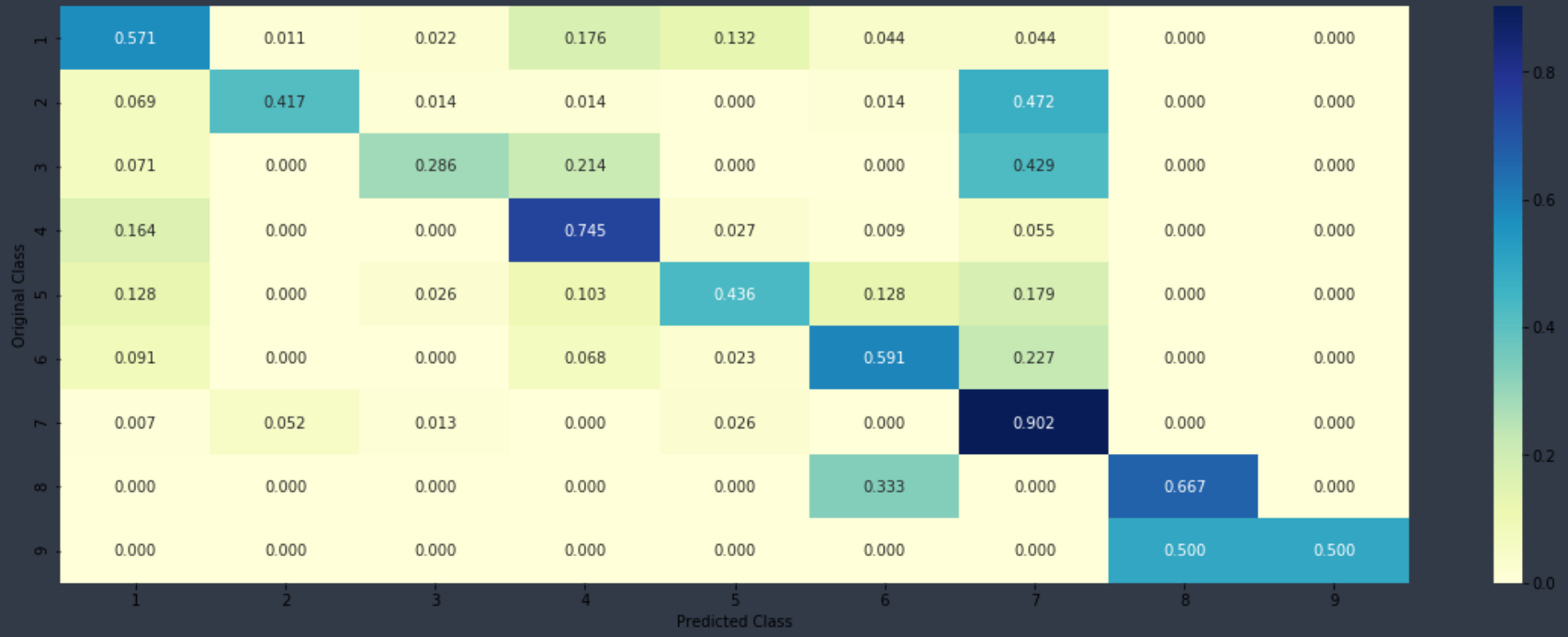
```
Log loss : 1.0760042491515471
Number of mis-classified points : 0.33458646616541354
------------------- Confusion matrix --------------------
```




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




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




### 4.3.1.3. Feature Importance

In [66]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
```

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

#### 4.3.1.3.1. Correctly Classified point

In [67]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', 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_onehotCodi
ng[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 : 4
Predicted Class Probabilities: [[3.028e-01 1.230e-02 1.400e-03 6.404e-01 8.100e-03 3.000e-03 2.760e-02
  4.100e-03 4.000e-04]]
Actual Class : 1
--------------------------------------------------
216 Text feature [microscopy] present in test data point [True]
232 Text feature [1631] present in test data point [True]
237 Text feature [1558] present in test data point [True]
256 Text feature [suppressor] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

#### 4.3.1.3.2. Incorrectly Classified point

In [68]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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: [[0.0632 0.1947 0.0108 0.0895 0.036  0.0316 0.565  0.0061 0.0032]]
Actual Class : 7
--------------------------------------------------
34 Text feature [miliary] present in test data point [True]
61 Text feature [ligand] present in test data point [True]
79 Text feature [tk] present in test data point [True]
114 Text feature [y1068] present in test data point [True]
124 Text feature [egfrs] present in test data point [True]
125 Text feature [preoperatively] present in test data point [True]
```

```
150 Text feature [phospho] present in test data point [True]
177 Text feature [receptors] present in test data point [True]
224 Text feature [activation] present in test data point [True]
225 Text feature [activating] present in test data point [True]
255 Text feature [technology] present in test data point [True]
271 Text feature [egf] present in test data point [True]
307 Text feature [phosphorylation] present in test data point [True]
373 Text feature [tyrosine] present in test data point [True]
387 Text feature [kinase] present in test data point [True]
473 Text feature [expressing] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

## 4.3.2. Without Class balancing

### 4.3.2.1. Hyper paramter tuning

In [69]:

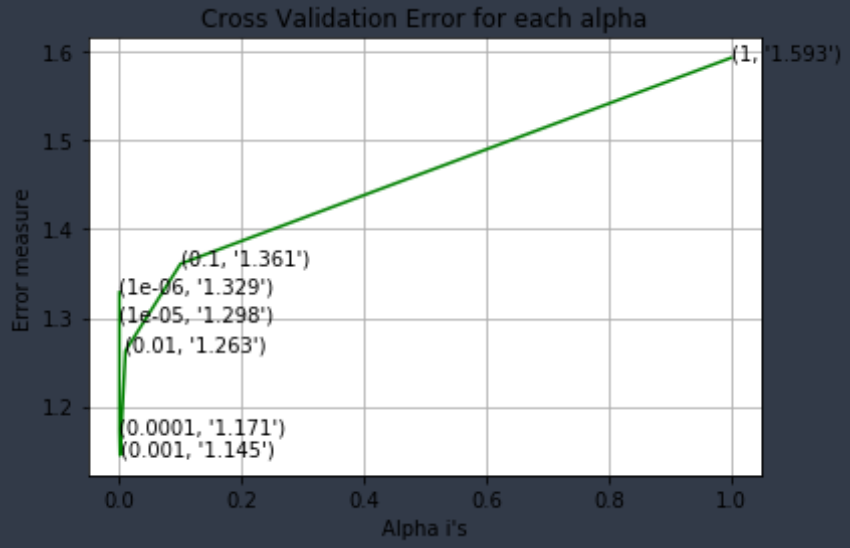
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

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



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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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",cv = None)
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_los
s(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 i
s:",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 : 1.3285368574207634
for alpha = 1e-05
Log Loss : 1.2978618779176583
for alpha = 0.0001
Log Loss : 1.1711312033811103
for alpha = 0.001
Log Loss : 1.1449867667026763
for alpha = 0.01
Log Loss : 1.2628228752844737
for alpha = 0.1
Log Loss : 1.3608134033919923
for alpha = 1
Log Loss : 1.5929319866470932
```



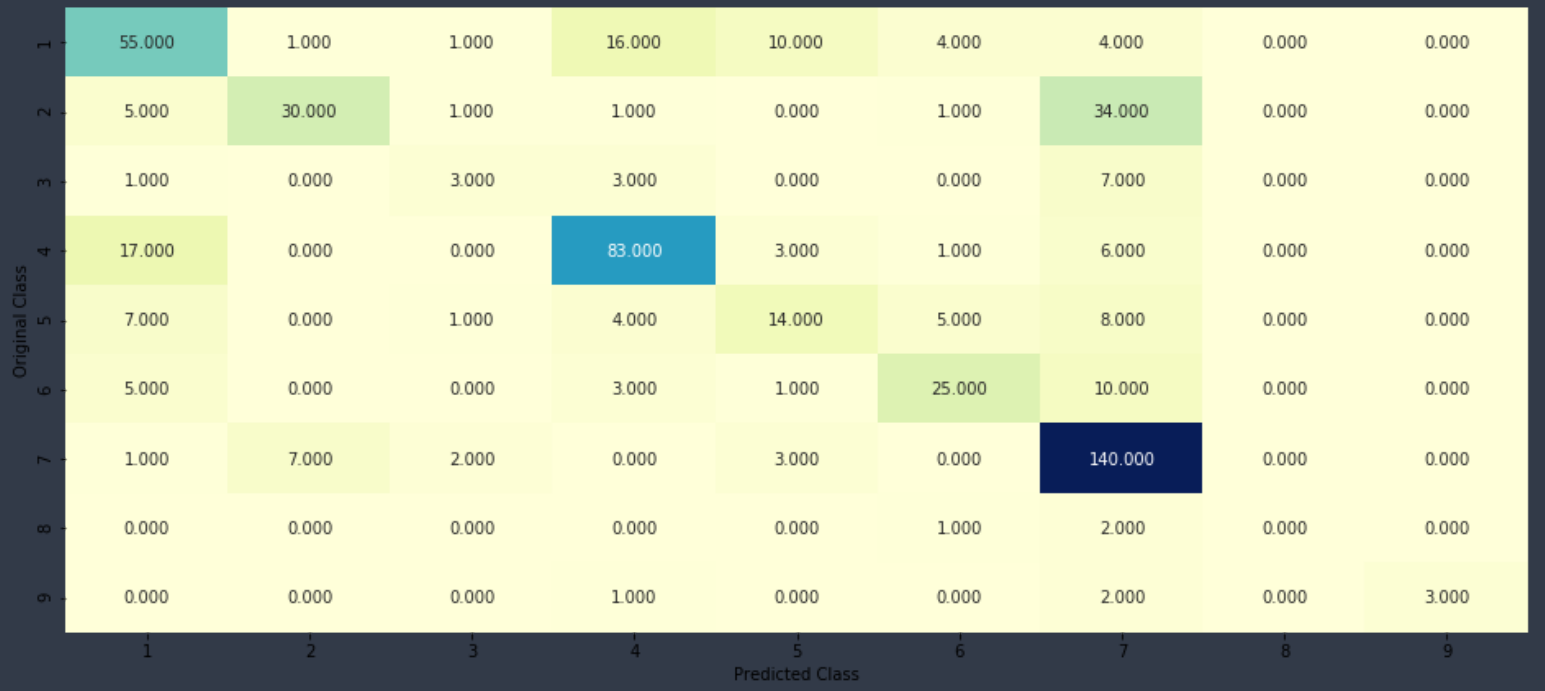

```
For values of best alpha =  0.001 The train log loss is: 0.5757651731961222
For values of best alpha =  0.001 The cross validation log loss is: 1.1449867667026763
```

```
For values of best alpha =  0.001 The test log loss is: 1.074768794830581
```

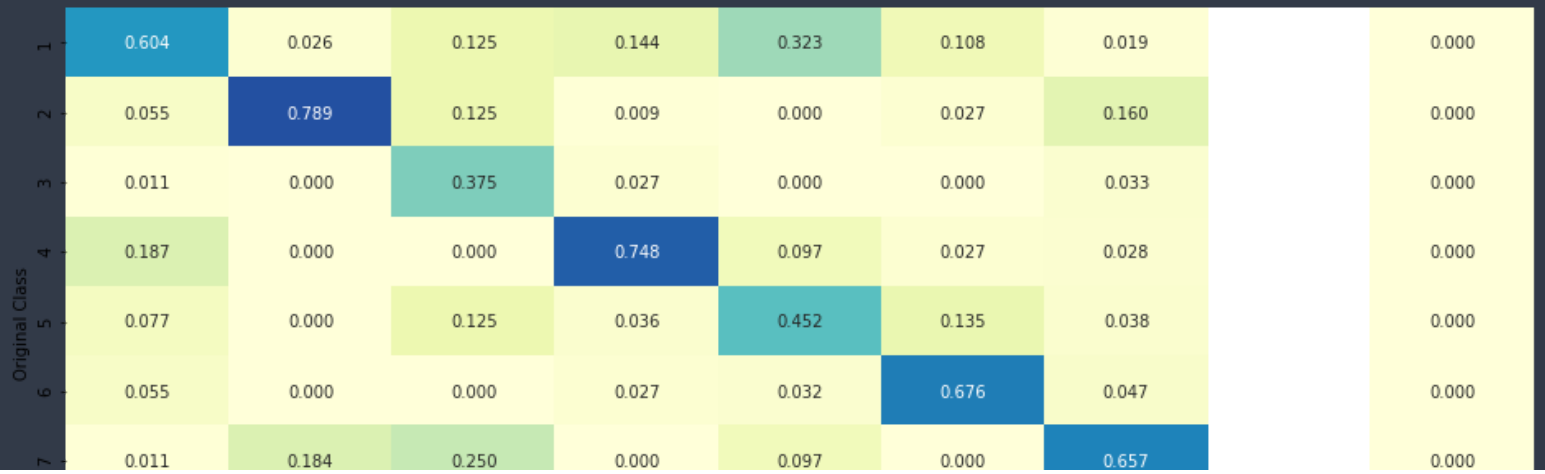
### 4.3.2.2. Testing model with best hyper parameters

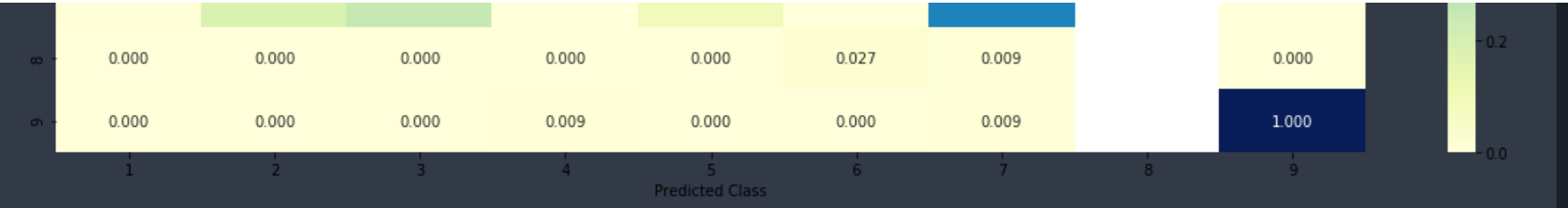
In [70]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.1449867667026763
Number of mis-classified points : 0.33646616541353386
------------------- Confusion matrix -------------------
```

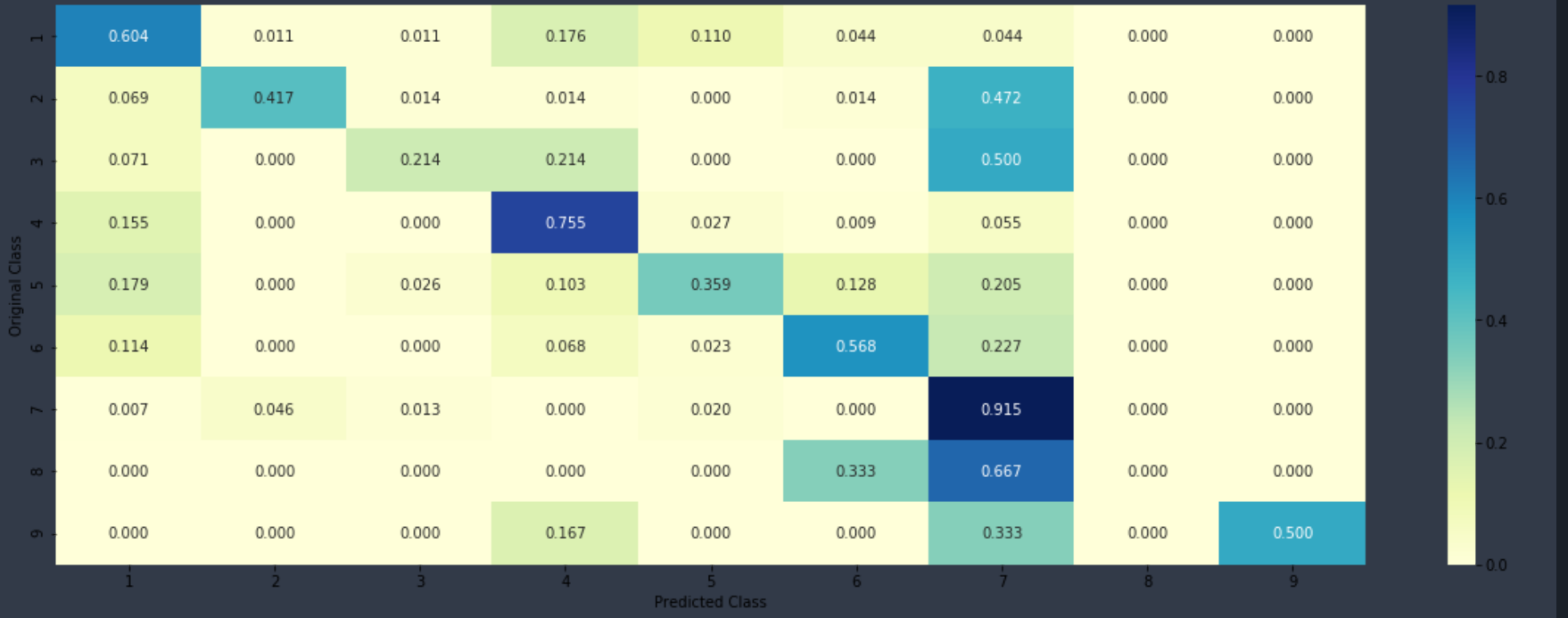



---------- Precision matrix (Columm Sum=1) ----------






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




### 4.3.2.3. Feature Importance, Correctly Classified point

In [71]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
```

```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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 : 4
Predicted Class Probabilities: [[0.3012 0.0122 0.0013 0.6358 0.0072 0.0028 0.0355 0.004  0.    ]]
Actual Class : 1
--------------------------------------------------
283 Text feature [1631] present in test data point [True]
301 Text feature [1558] present in test data point [True]
323 Text feature [microscopy] present in test data point [True]
342 Text feature [suppressor] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

#### 4.3.2.4. Feature Importance, Inorrectly Classified point

In [72]:

```
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_onehotCodi
ng[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: [[0.0605 0.1945 0.0111 0.0906 0.0365 0.0318 0.567  0.0064 0.0015]]
Actual Class : 7
--------------------------------------------------
84 Text feature [miliary] present in test data point [True]
154 Text feature [ligand] present in test data point [True]
184 Text feature [tk] present in test data point [True]
205 Text feature [y1068] present in test data point [True]
269 Text feature [preoperatively] present in test data point [True]
271 Text feature [activating] present in test data point [True]
335 Text feature [technology] present in test data point [True]
342 Text feature [phospho] present in test data point [True]
357 Text feature [receptors] present in test data point [True]
381 Text feature [egfrs] present in test data point [True]
418 Text feature [activation] present in test data point [True]
443 Text feature [phosphorylation] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

# 4.4. Linear Support Vector Machines

## 4.4.1. Hyper paramter tuning

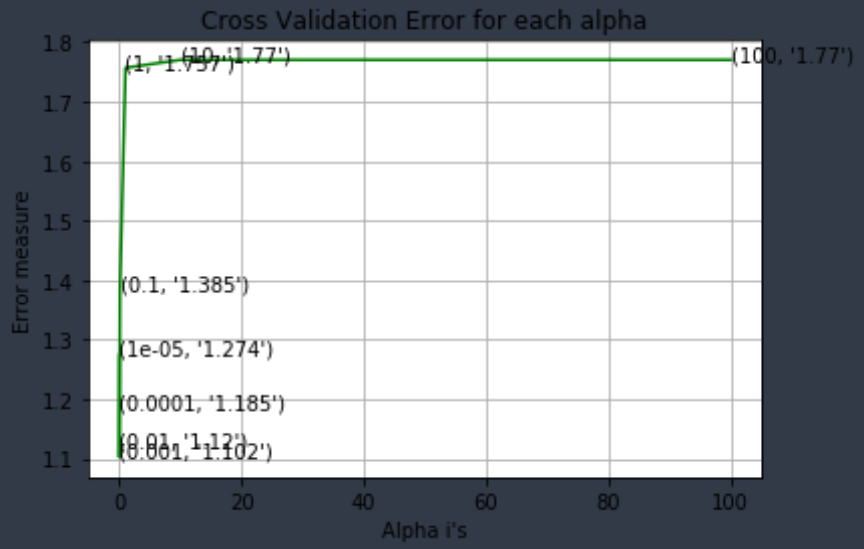
In [73]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.o
rg/stable/modules/generated/sklearn.svm.SVC.html

# ----------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabilit
y=False, tol=0.001,
```

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/ma
thematical-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', r
andom_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv= None)
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-
15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.2741786542464353
for C = 0.0001
Log Loss : 1.185328490563755
for C = 0.001
Log Loss : 1.102357423660535
for C = 0.01
Log Loss : 1.1195526273243634
for C = 0.1
Log Loss : 1.38542084915679
for C = 1
Log Loss : 1.7569648052226707
for C = 10
Log Loss : 1.7704360715371192
for C = 100
Log Loss : 1.7704358887259548
```




```
For values of best alpha =  0.001 The train log loss is: 0.5878105661869157
For values of best alpha =  0.001 The cross validation log loss is: 1.102357423660535
For values of best alpha =  0.001 The test log loss is: 1.1225112701015525
```

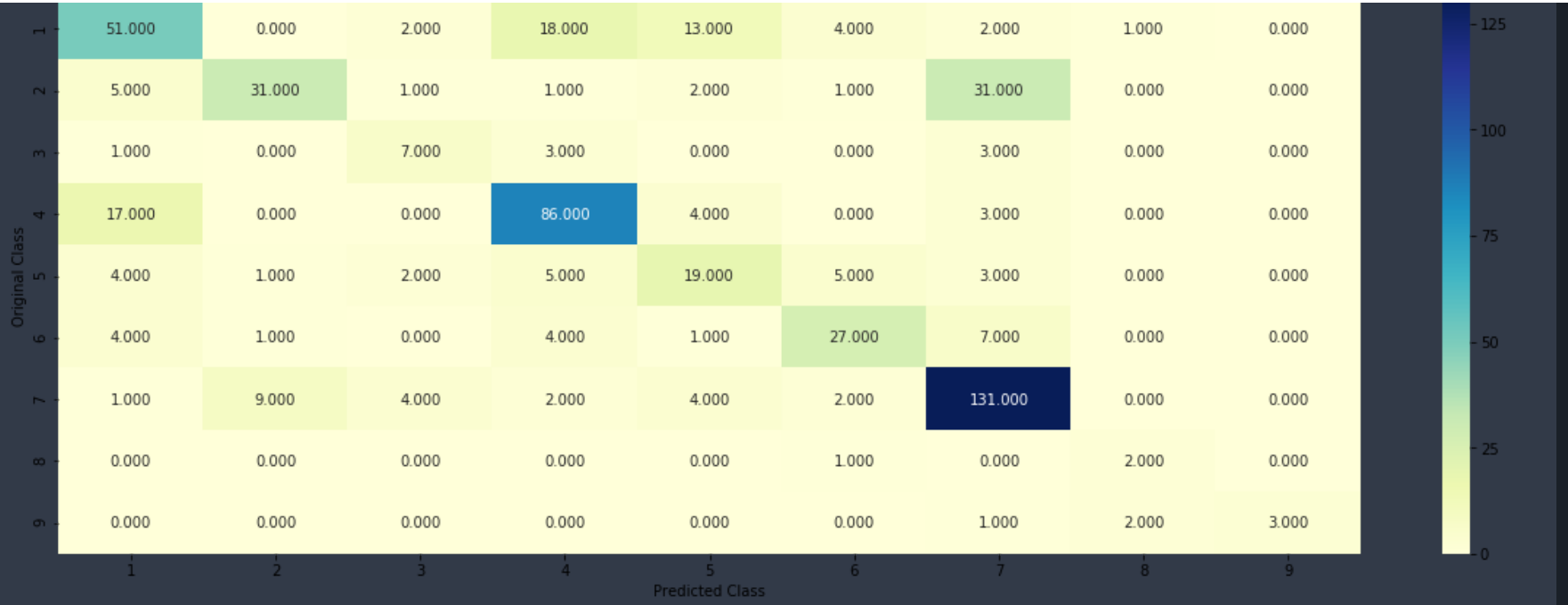
## 4.4.2. Testing model with best hyper parameters

In [74]:

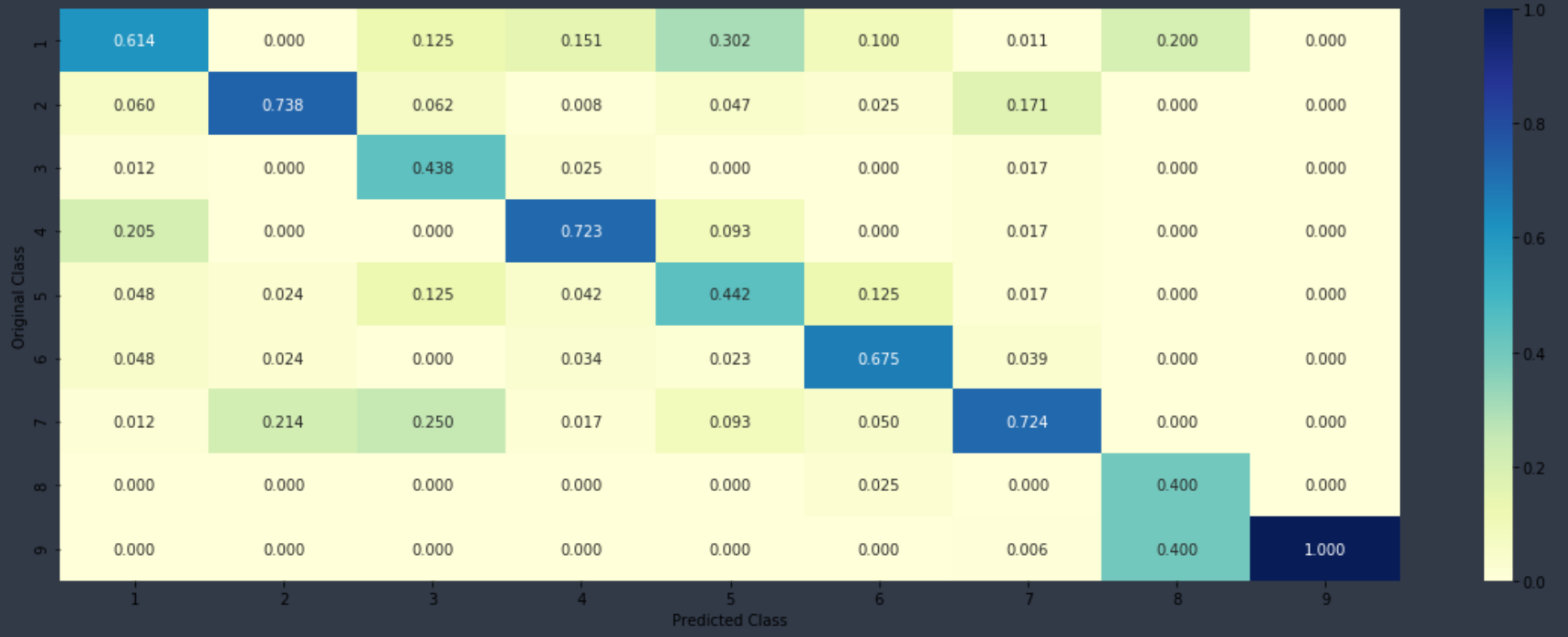
```
# read more about support vector machines with linear kernals here http://scikit-learn.o
rg/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

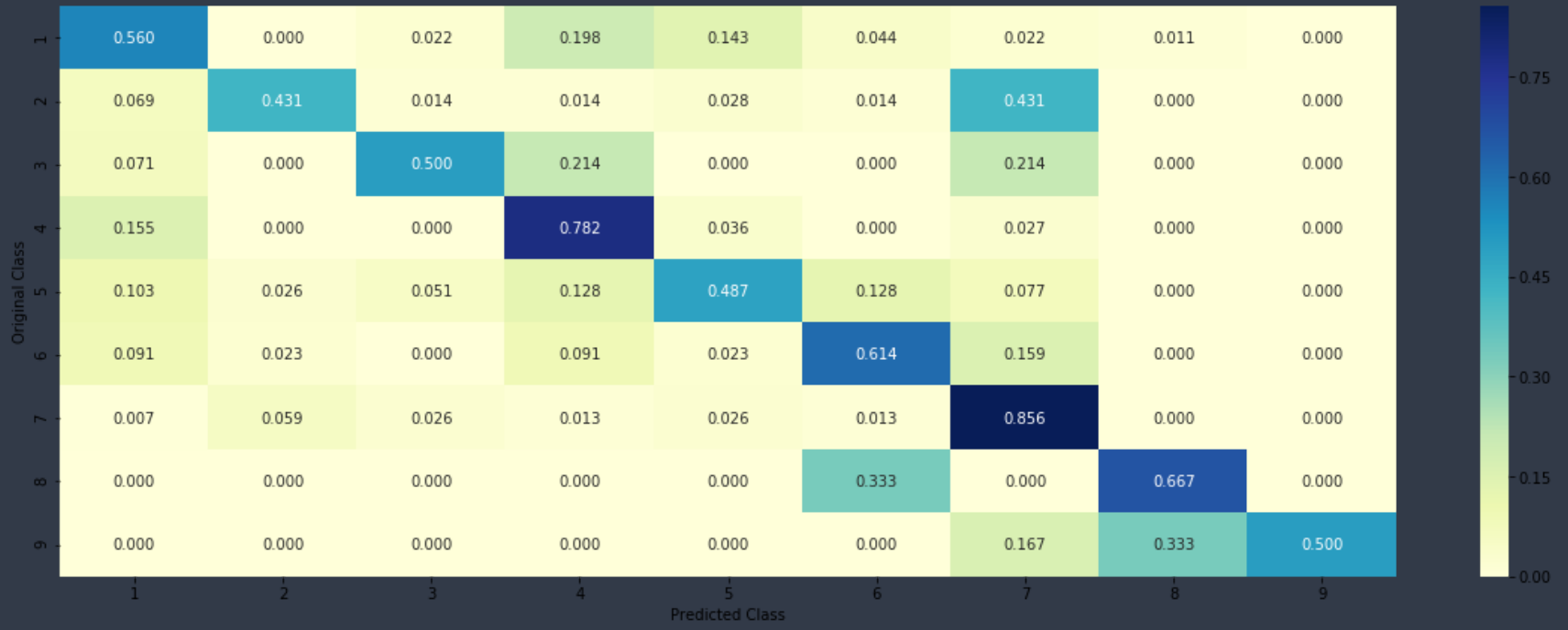
```
Log loss : 1.102357423660535
Number of mis-classified points : 0.32894736842105265
-------------------- Confusion matrix --------------------
```




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




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




## 4.3.3. Feature Importance

### 4.3.3.1. For Correctly classified point

In [75]:

```
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_onehotCodi
ng[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 : 4
Predicted Class Probabilities: [[0.2818 0.0369 0.008  0.5759 0.0279 0.0105 0.052  0.004  0.003 ]]
Actual Class : 1
--------------------------------------------------
306 Text feature [1631] present in test data point [True]
346 Text feature [1558] present in test data point [True]
421 Text feature [1753] present in test data point [True]
429 Text feature [py99] present in test data point [True]
433 Text feature [1100] present in test data point [True]
438 Text feature [y1571h] present in test data point [True]
475 Text feature [xpress] present in test data point [True]
476 Text feature [pervanadate] present in test data point [True]
489 Text feature [1134] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

#### 4.3.3.2. For Incorrectly classified point

In [76]:
```
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_onehotCodi
ng[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: [[0.0964 0.1478 0.014  0.1128 0.0454 0.0439 0.5298 0.0054 0.0046]]
Actual Class : 7
--------------------------------------------------
223 Text feature [miliary] present in test data point [True]
325 Text feature [tarceva] present in test data point [True]
360 Text feature [y1068] present in test data point [True]
436 Text feature [technology] present in test data point [True]
458 Text feature [tk] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

# 4.5 Random Forest Classifier

## 4.5.1. Hyper paramter tuning (With One hot Encoding)

In [77]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
one, 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/
# --------------------------------

from sklearn.ensemble import RandomForestClassifier
# 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, rand
om_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv =None)
        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_erro
r_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', ma
```

```
x_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",cv =None)
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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2839512073781536
for n_estimators = 100 and max depth =  10
Log Loss : 1.2178312594550817
for n_estimators = 200 and max depth =  5
Log Loss : 1.262209206116554
for n_estimators = 200 and max depth =  10
Log Loss : 1.20412219606452
for n_estimators = 500 and max depth =  5
Log Loss : 1.251506756807322
for n_estimators = 500 and max depth =  10
Log Loss : 1.1911951337755293
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2482308336098311
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1880874447365672
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2464167851786507
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1886953923846035
For values of best estimator =  1000 The train log loss is: 0.7152950491866529
```

```
For values of best estimator =  1000 The cross validation log loss is: 1.1880874447365672
For values of best estimator =  1000 The test log loss is: 1.1637889799271286
```

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

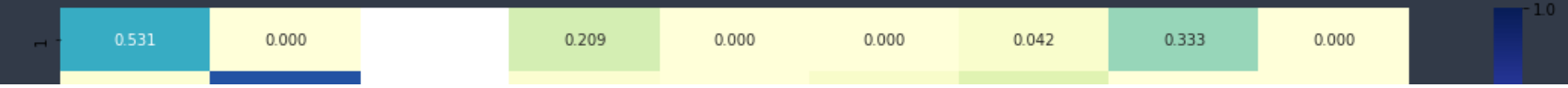
In [78]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
one, 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/ra
ndom-forest-and-their-construction-2/
# ------------------------------
```

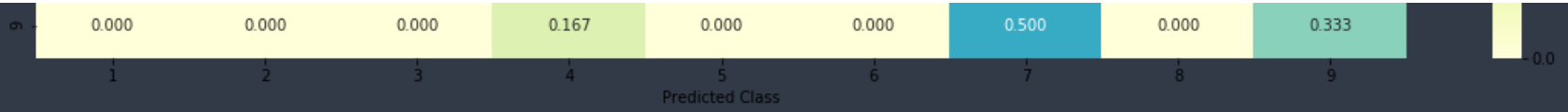
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y,
clf)
```

```
Log loss : 1.1880874447365672
Number of mis-classified points : 0.40977443609022557
-------------------- 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 [79]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_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_onehotCodi
ng[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1914 0.0404 0.0193 0.5462 0.0479 0.0433 0.1007 0.0061 0.0047]]
```

```
Actual Class : 1
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [function] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
8 Text feature [activated] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
13 Text feature [functional] present in test data point [True]
14 Text feature [kinases] present in test data point [True]
15 Text feature [growth] present in test data point [True]
19 Text feature [akt] present in test data point [True]
20 Text feature [variants] present in test data point [True]
21 Text feature [loss] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [cells] present in test data point [True]
25 Text feature [missense] present in test data point [True]
34 Text feature [inhibition] present in test data point [True]
35 Text feature [phosphorylation] present in test data point [True]
39 Text feature [neutral] present in test data point [True]
41 Text feature [expressing] present in test data point [True]
53 Text feature [variant] present in test data point [True]
55 Text feature [cell] present in test data point [True]
56 Text feature [downstream] present in test data point [True]
59 Text feature [protein] present in test data point [True]
66 Text feature [pathogenic] present in test data point [True]
70 Text feature [serum] present in test data point [True]
72 Text feature [patients] present in test data point [True]
75 Text feature [expression] present in test data point [True]
78 Text feature [phospho] present in test data point [True]
81 Text feature [frameshift] present in test data point [True]
82 Text feature [proteins] present in test data point [True]
83 Text feature [inhibited] present in test data point [True]
86 Text feature [truncating] present in test data point [True]
88 Text feature [starved] present in test data point [True]
91 Text feature [clinical] present in test data point [True]
97 Text feature [phosphatase] present in test data point [True]
```

```
99 Text feature [days] present in test data point [True]
Out of the top  100  features  38 are present in query point
```

### 4.5.3.2. Inorrectly Classified point

In [80]:
```
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_onehotCodi
ng[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0621 0.2532 0.0199 0.0745 0.0471 0.0407 0.4912 0.0068 0.0043]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [function] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
12 Text feature [receptor] present in test data point [True]
13 Text feature [functional] present in test data point [True]
15 Text feature [growth] present in test data point [True]
17 Text feature [oncogenic] present in test data point [True]
20 Text feature [variants] present in test data point [True]
```

```
23 Text feature [signaling] present in test data point [True]
24 Text feature [cells] present in test data point [True]
25 Text feature [missense] present in test data point [True]
27 Text feature [therapeutic] present in test data point [True]
29 Text feature [treated] present in test data point [True]
34 Text feature [inhibition] present in test data point [True]
35 Text feature [phosphorylation] present in test data point [True]
37 Text feature [months] present in test data point [True]
41 Text feature [expressing] present in test data point [True]
43 Text feature [resistance] present in test data point [True]
44 Text feature [lines] present in test data point [True]
48 Text feature [nsclc] present in test data point [True]
51 Text feature [sensitivity] present in test data point [True]
53 Text feature [variant] present in test data point [True]
55 Text feature [cell] present in test data point [True]
57 Text feature [tkis] present in test data point [True]
58 Text feature [atp] present in test data point [True]
59 Text feature [protein] present in test data point [True]
60 Text feature [therapy] present in test data point [True]
63 Text feature [autophosphorylation] present in test data point [True]
70 Text feature [serum] present in test data point [True]
71 Text feature [median] present in test data point [True]
72 Text feature [patients] present in test data point [True]
75 Text feature [expression] present in test data point [True]
77 Text feature [egfr] present in test data point [True]
78 Text feature [phospho] present in test data point [True]
80 Text feature [amplification] present in test data point [True]
82 Text feature [proteins] present in test data point [True]
85 Text feature [ligand] present in test data point [True]
89 Text feature [trial] present in test data point [True]
90 Text feature [mutant] present in test data point [True]
91 Text feature [clinical] present in test data point [True]
92 Text feature [tki] present in test data point [True]
93 Text feature [survival] present in test data point [True]
94 Text feature [extracellular] present in test data point [True]
98 Text feature [receptors] present in test data point [True]
Out of the top  100  features  48 are present in query point
```

## 4.5.3. Hyper paramter tuning (With Response Coding)

In [81]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
one, 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/ra
ndom-forest-and-their-construction-2/
# ------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
```

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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
om_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_erro
r_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', ma
x_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:",l
og_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1223033367028816
```

```
for n_estimators = 10 and max depth =  3
Log Loss : 1.650200139295415
for n_estimators = 10 and max depth =  5
Log Loss : 1.4877140985099533
for n_estimators = 10 and max depth =  10
Log Loss : 1.82201302266505
for n_estimators = 50 and max depth =  2
Log Loss : 1.5643639844050643
for n_estimators = 50 and max depth =  3
Log Loss : 1.442254974742197
for n_estimators = 50 and max depth =  5
Log Loss : 1.5051511858190947
for n_estimators = 50 and max depth =  10
Log Loss : 1.7394058722739667
for n_estimators = 100 and max depth =  2
Log Loss : 1.4710969059317156
for n_estimators = 100 and max depth =  3
Log Loss : 1.4979160390639916
for n_estimators = 100 and max depth =  5
Log Loss : 1.3917815719910214
for n_estimators = 100 and max depth =  10
Log Loss : 1.7801669093883972
for n_estimators = 200 and max depth =  2
Log Loss : 1.5253607177293593
for n_estimators = 200 and max depth =  3
Log Loss : 1.478743075116895
for n_estimators = 200 and max depth =  5
Log Loss : 1.3684142016356005
for n_estimators = 200 and max depth =  10
Log Loss : 1.7999313498505427
for n_estimators = 500 and max depth =  2
Log Loss : 1.6424170441705308
for n_estimators = 500 and max depth =  3
Log Loss : 1.5398225973020423
for n_estimators = 500 and max depth =  5
Log Loss : 1.4014256758800347
for n_estimators = 500 and max depth =  10
Log Loss : 1.8062430810842292

for n_estimators = 1000 and max depth =  2
```

```
Log Loss : 1.6297672480970207
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5539038578046147
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3642629933195005
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7821531467694767
For values of best alpha =  1000 The train log loss is: 0.05142137426698682
For values of best alpha =  1000 The cross validation log loss is: 1.3642629933195005
For values of best alpha =  1000 The test log loss is: 1.3636899590326306
```

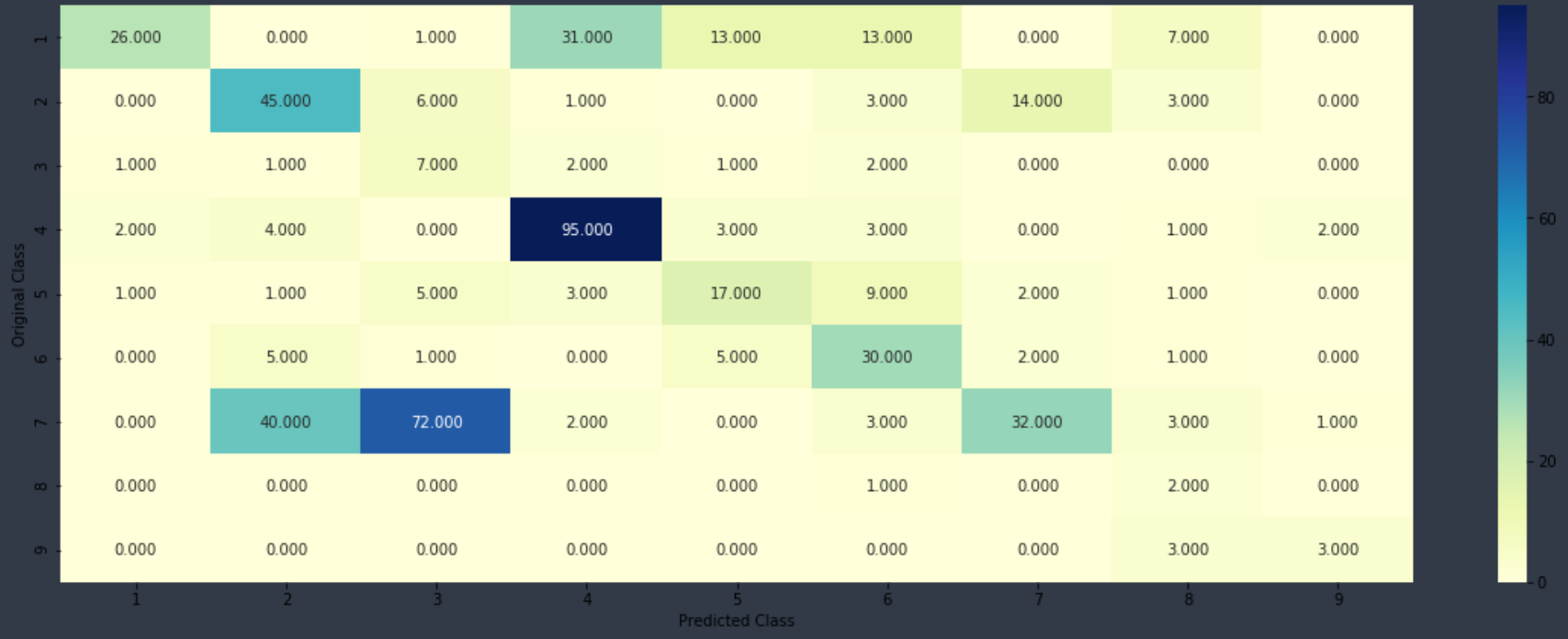
## 4.5.4. Testing model with best hyper parameters (Response Coding)

In [82]:

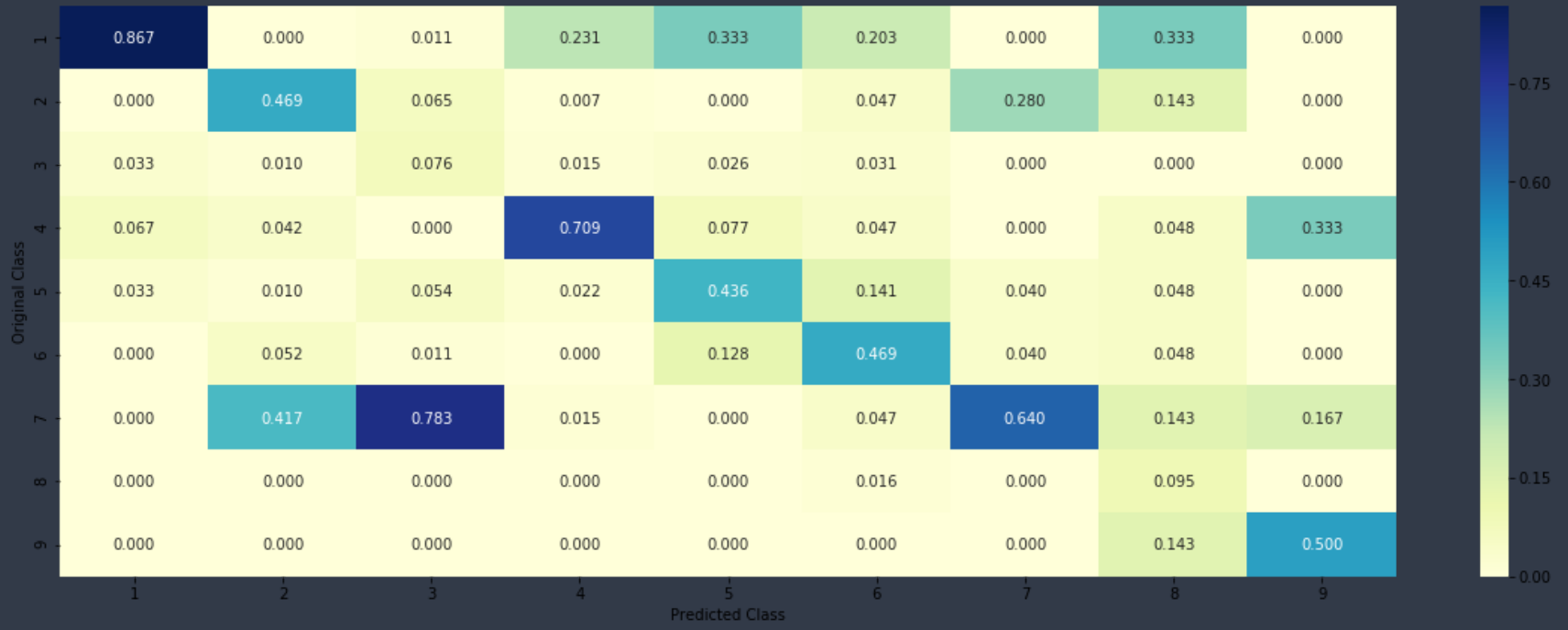
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
one, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
None, min_impurity_decrease=0.0,
# min_impurity_split=None, 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)
```

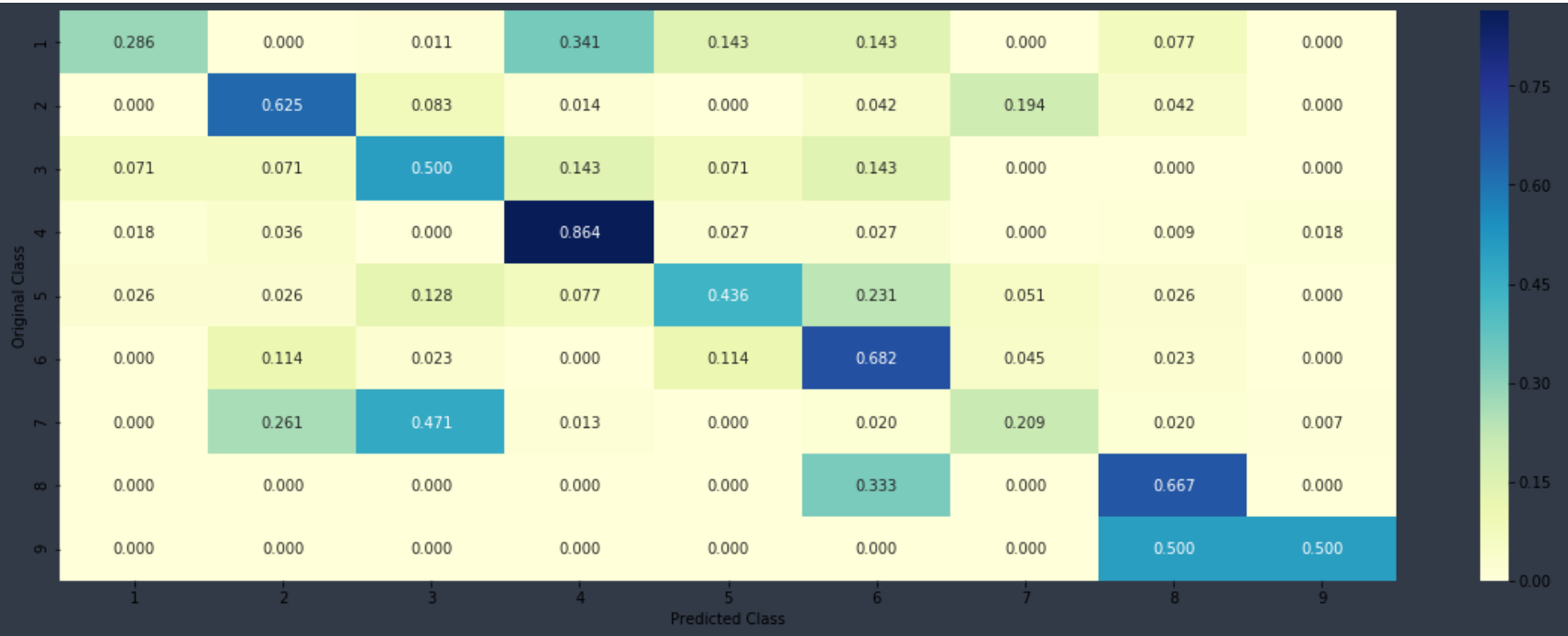
```
Log loss : 1.3642629933195005
Number of mis-classified points : 0.5169172932330827
-------------------- Confusion matrix --------------------
```



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




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




## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point

In [83]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', ma
x_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_responseCo
ding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1494 0.0181 0.143  0.495  0.0452 0.0939 0.0096 0.0256 0.0203]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
```

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

#### 4.5.5.2. Incorrectly Classified point

In [84]:
```
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_responseCo
ding[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: [[0.0124 0.4414 0.1536 0.0175 0.0271 0.0555 0.2381 0.0385 0.0158]]
Actual Class : 7
--------------------------------------------------
```

```
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
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 [85]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
```

```
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t Descent.
# predict(X)    Predict class labels for samples in X.

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


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

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

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


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

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

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

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

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


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', ran
dom_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid",cv=None)

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', rando
m_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid",cv=None)


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(
cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pro
ba(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_oneh
otCoding))))
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_classifie
r=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_lo
ss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.07
Support vector machines : Log Loss: 1.76
Naive Bayes : Log Loss: 1.27
------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.520
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.109
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.198
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.506
```

## 4.7.2 testing the model with the best hyper parameters

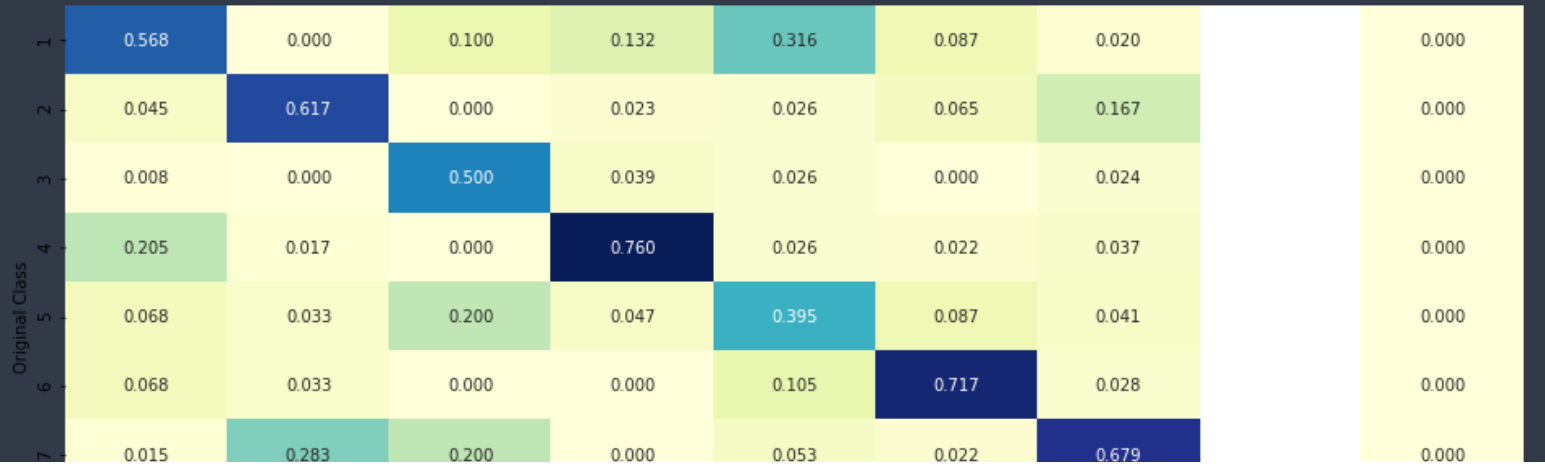
In [86]:

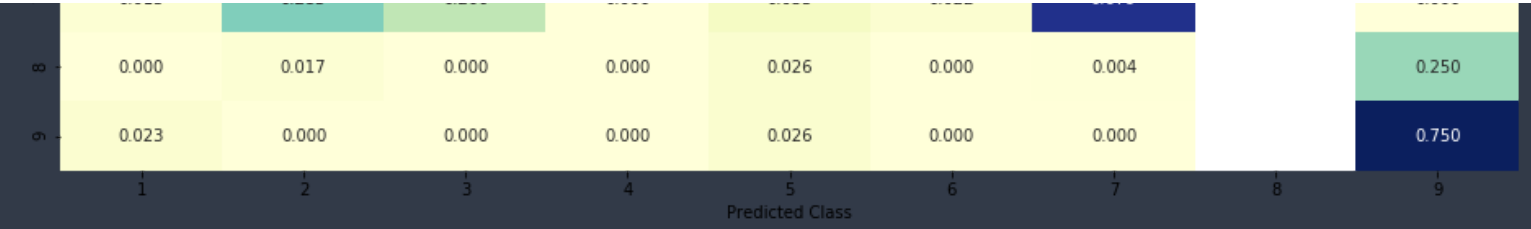
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr
, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

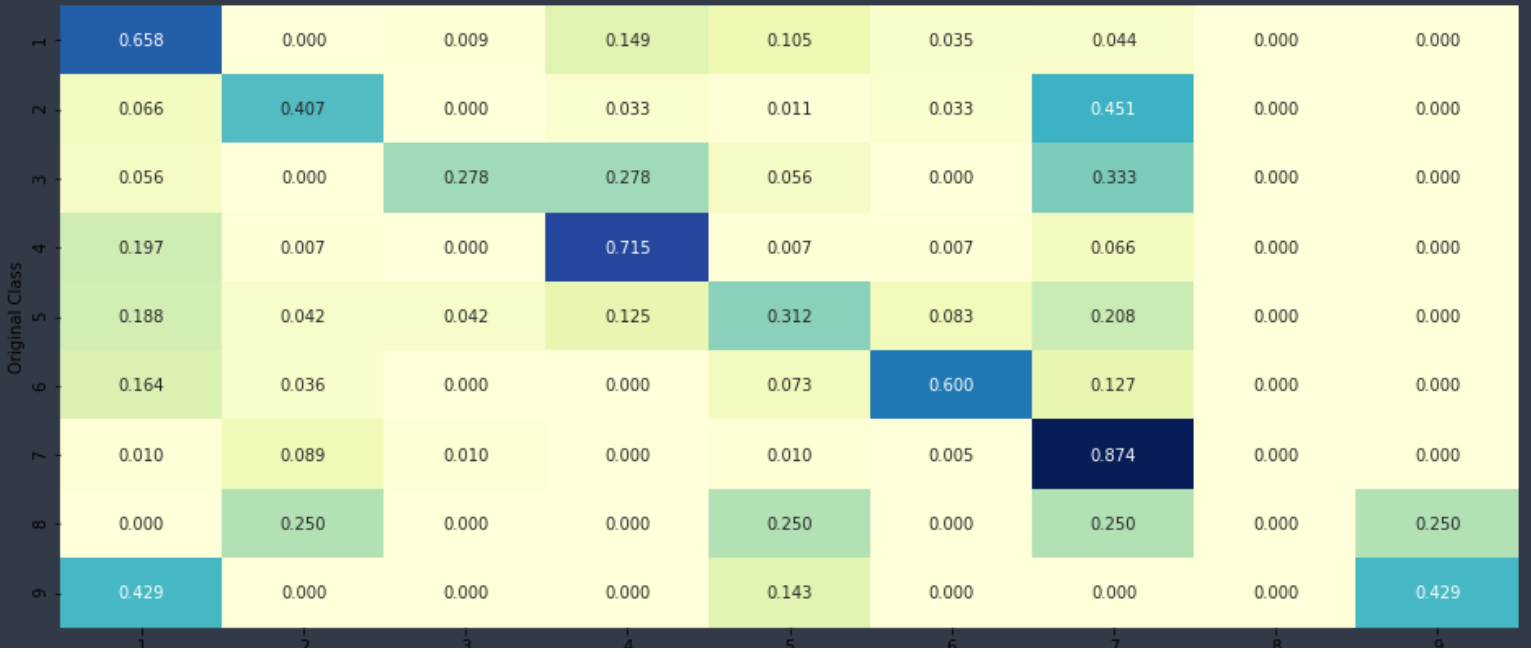
```
Log loss (train) on the stacking classifier : 0.6491011020777427
Log loss (CV) on the stacking classifier : 1.1089663153222316
Log loss (test) on the stacking classifier : 1.1077181052418779
Number of missclassified point : 0.34887218045112783
-------------------- Confusion matrix --------------------
```


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






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

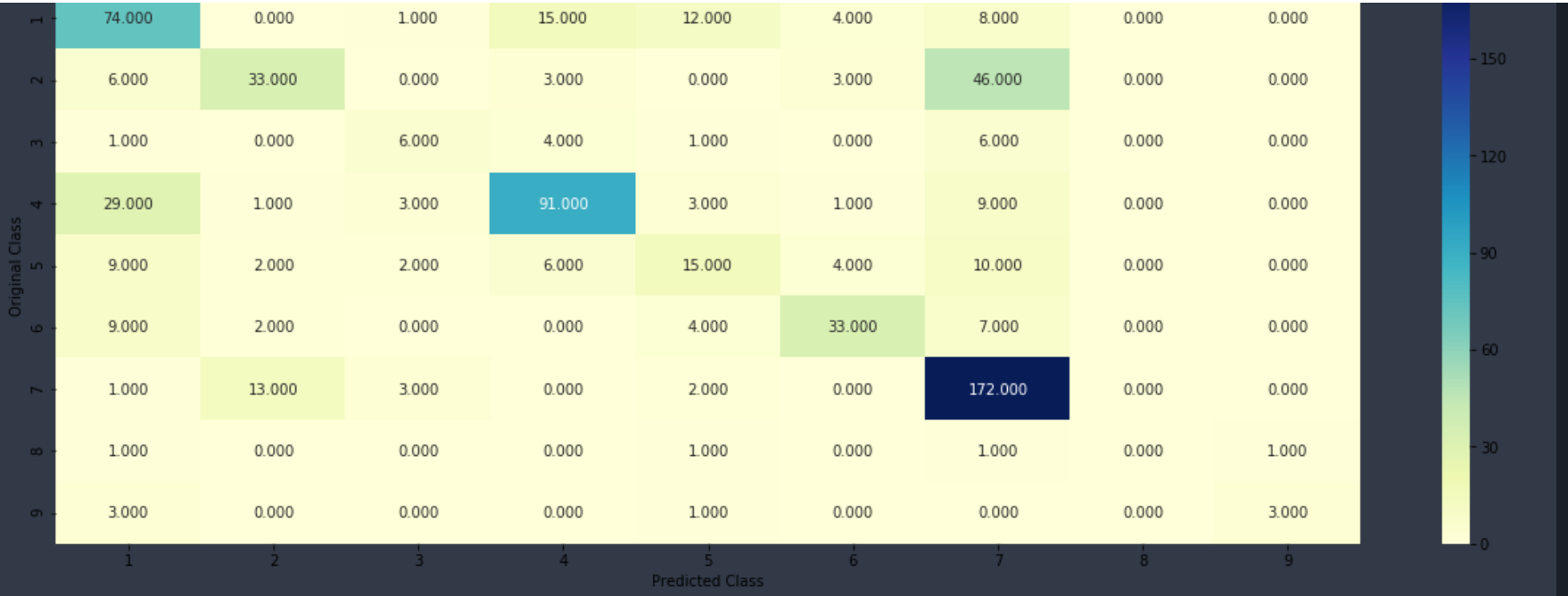


Predicted Class

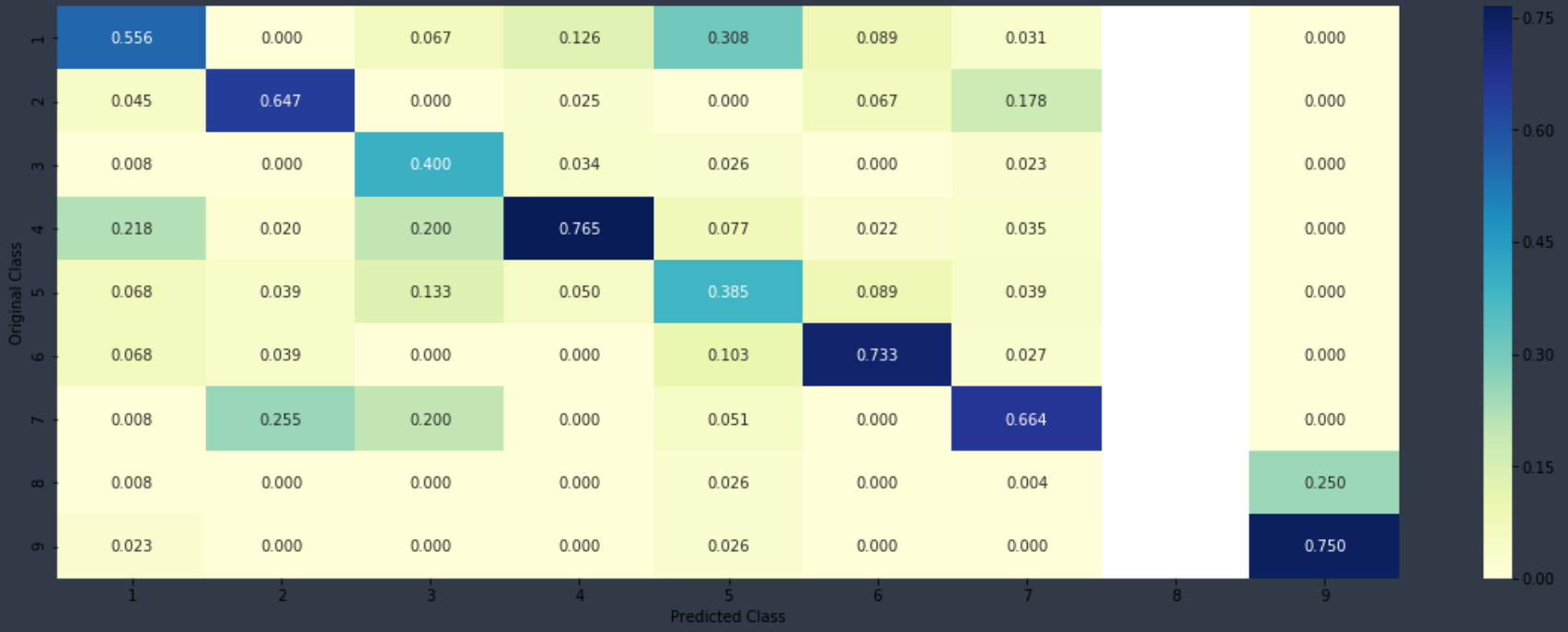
## 4.7.3 Maximum Voting classifier

In [87]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

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))
```

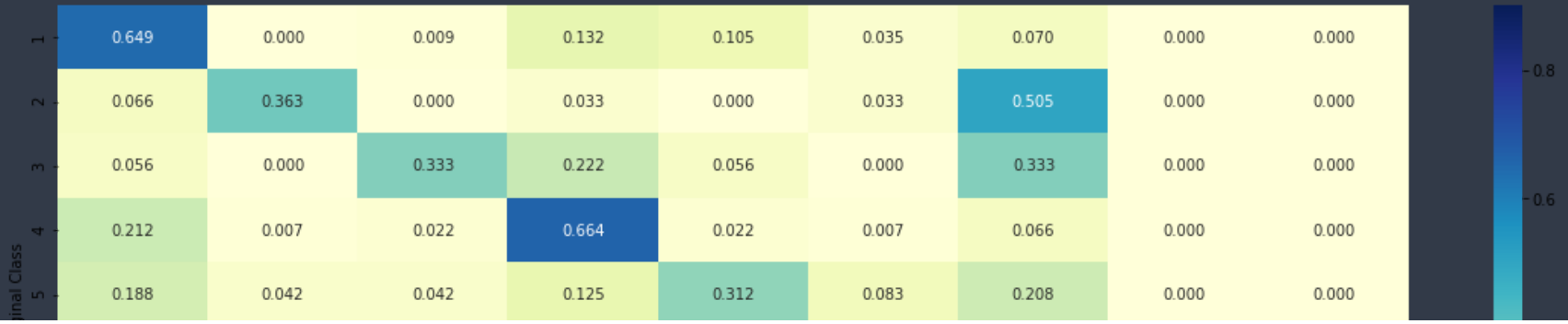
```
Log loss (train) on the VotingClassifier : 0.9100030774427197
Log loss (CV) on the VotingClassifier : 1.1929132582731894
Log loss (test) on the VotingClassifier : 1.2058254674211573
Number of missclassified point : 0.35789473684210527
------------------- Confusion matrix --------------------
```

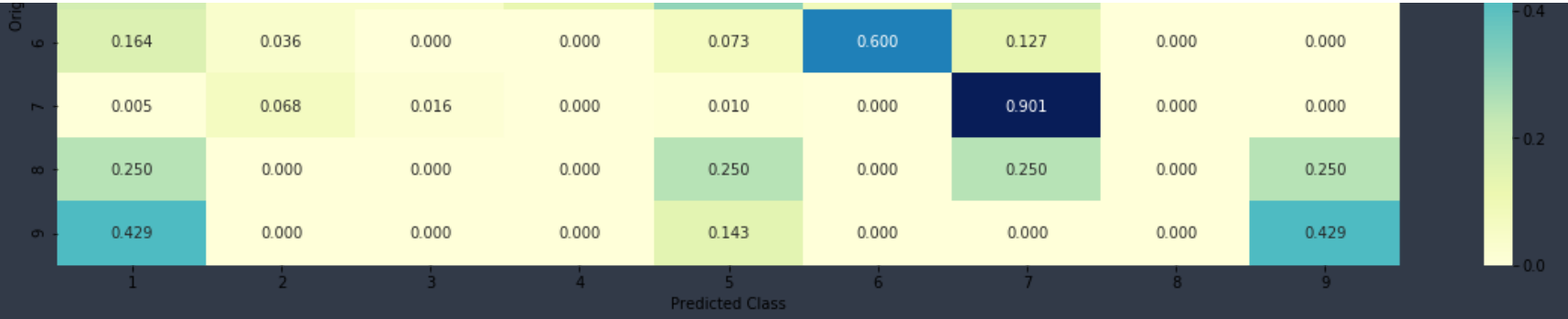



---------- Precision matrix (Columm Sum=1) ----------




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






# 5. Assignments

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

# 1. Using TF-IDF features

In [88]:
```
# building a tfidf-CountVectorizer with all the words that occured minimum 3 times in tr
ain data
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = tfidf_text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
```

```
train_text_features= tfidf_text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it o
ccured
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 [89]:
```
# 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 = tfidf_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 = tfidf_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 [90]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_fe
ature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
```

```
re_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one
hotCoding))

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

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

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

In [91]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3198)
(number of data points * number of features) in test data =  (665, 3198)
(number of data points * number of features) in cross validation data = (532, 3198)
```

# Base line models

In [92]:

```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

```
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word
,yes_no))

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

## KNN

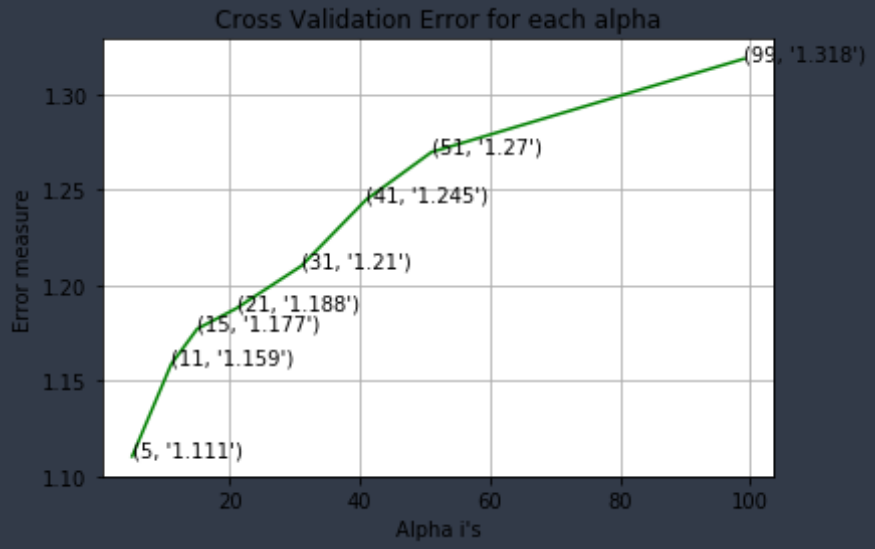
In [93]:

```
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",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 = 5
Log Loss : 1.110701052180169
for alpha = 11
Log Loss : 1.1587978557688752
for alpha = 15
Log Loss : 1.1772544987624372
for alpha = 21
Log Loss : 1.1879402137407125
for alpha = 31
Log Loss : 1.210117097586449
```

```
for alpha = 41
Log Loss : 1.2448359315028281
for alpha = 51
Log Loss : 1.2695530389173733
for alpha = 99
Log Loss : 1.3182994022031123
```

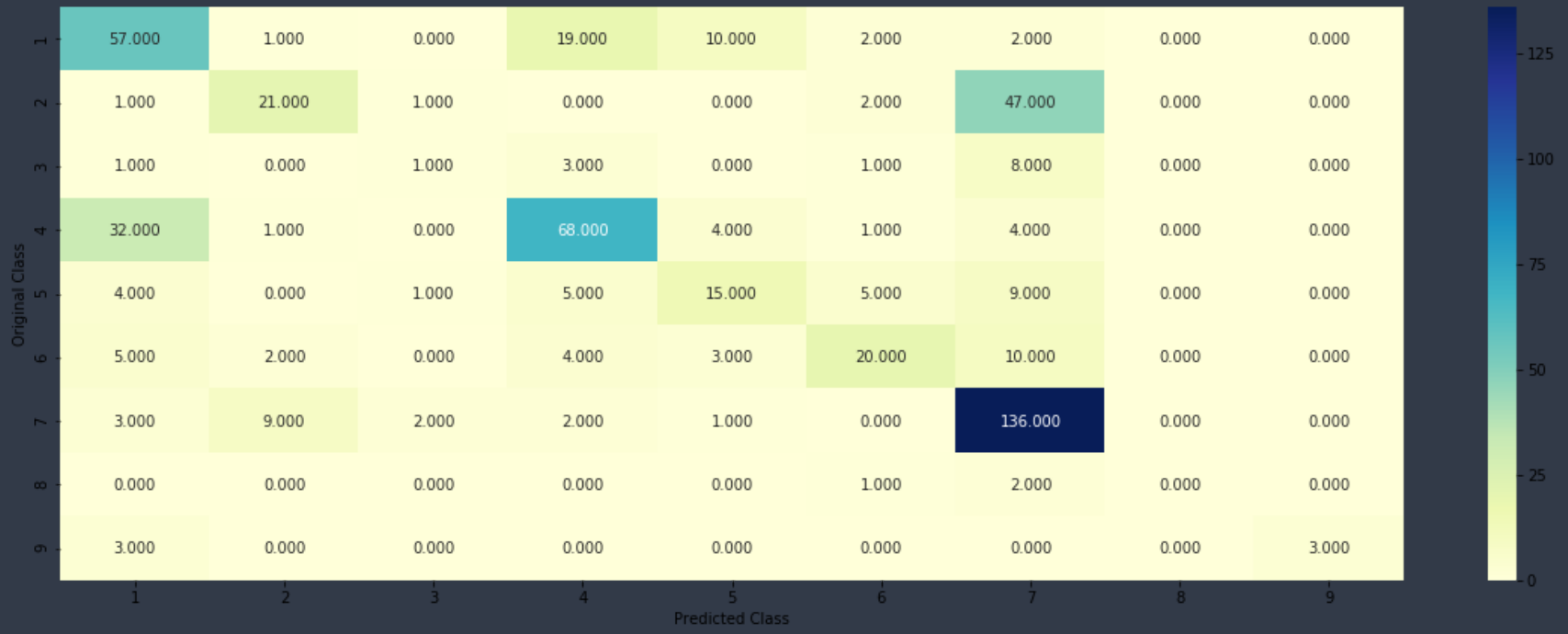



```
For values of best alpha =  5 The train log loss is: 0.8981075660581528
For values of best alpha =  5 The cross validation log loss is: 1.110701052180169
For values of best alpha =  5 The test log loss is: 1.0978665280576856
```

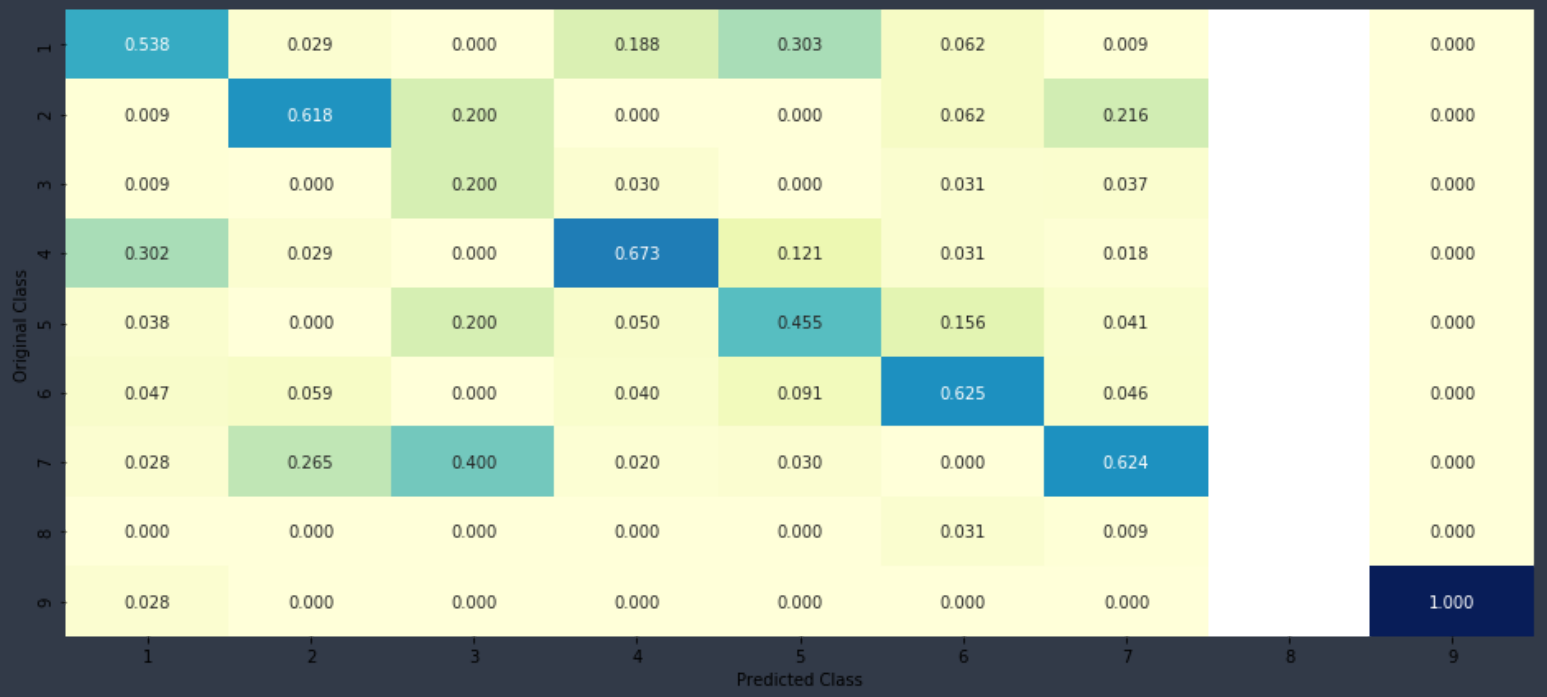
## Testing on te best Hyperparameter

In [94]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y
, clf)
```

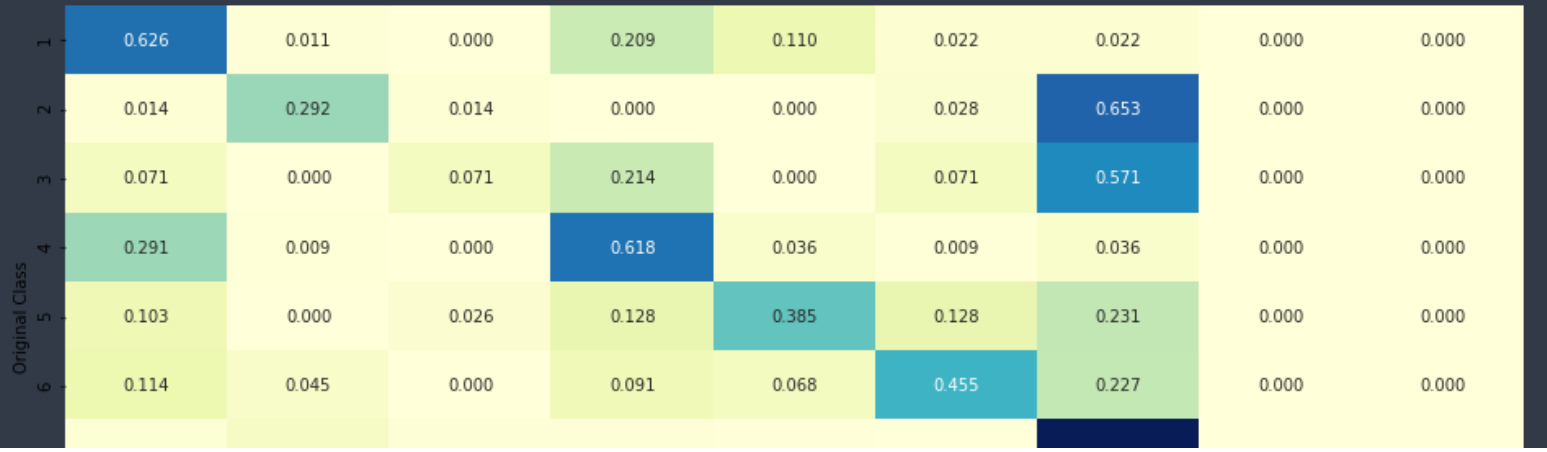
```
Log loss : 1.110701052180169
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```

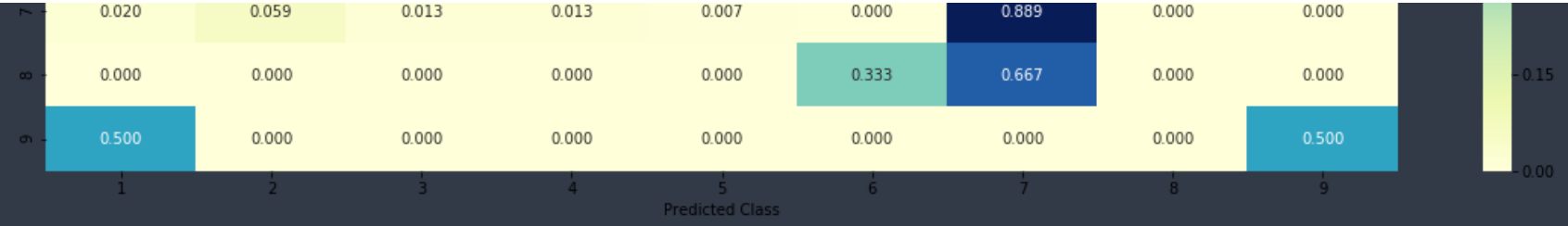



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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






## Sample query point : 1

In [95]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv = None)
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 12

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

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

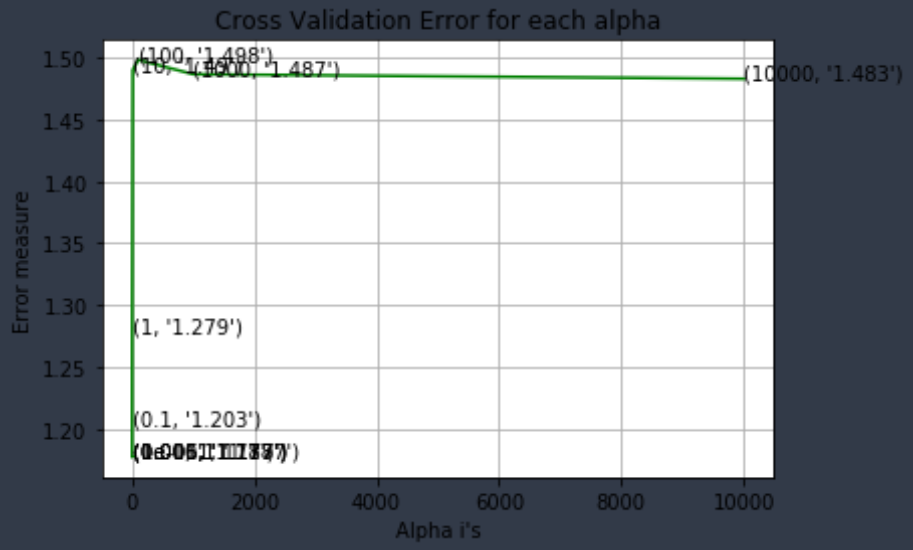
# Naive Bayes

In [96]:

```
alpha = [10**i for i in range(-6,5) ]
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",cv=5)
    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 esti
mates
    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 = MultinomialNB(alpha=alpha[best_alpha])
```

```
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",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 : 1.1770449345658636
for alpha = 1e-05
Log Loss : 1.1769437366366404
for alpha = 0.0001
Log Loss : 1.1770292573290886
for alpha = 0.001
Log Loss : 1.1779363543650392
for alpha = 0.01
Log Loss : 1.1779414573780072
for alpha = 0.1
Log Loss : 1.202589657761211
for alpha = 1
Log Loss : 1.278868474221302
for alpha = 10
Log Loss : 1.4900289091540542
for alpha = 100
Log Loss : 1.498189844078447
for alpha = 1000
Log Loss : 1.4865390157495149
for alpha = 10000
Log Loss : 1.4828841133294703
```




```
For values of best alpha =  1e-05 The train log loss is: 0.44454678199345393
For values of best alpha =  1e-05 The cross validation log loss is: 1.1769437366366404
For values of best alpha =  1e-05 The test log loss is: 1.1635278731450238
```

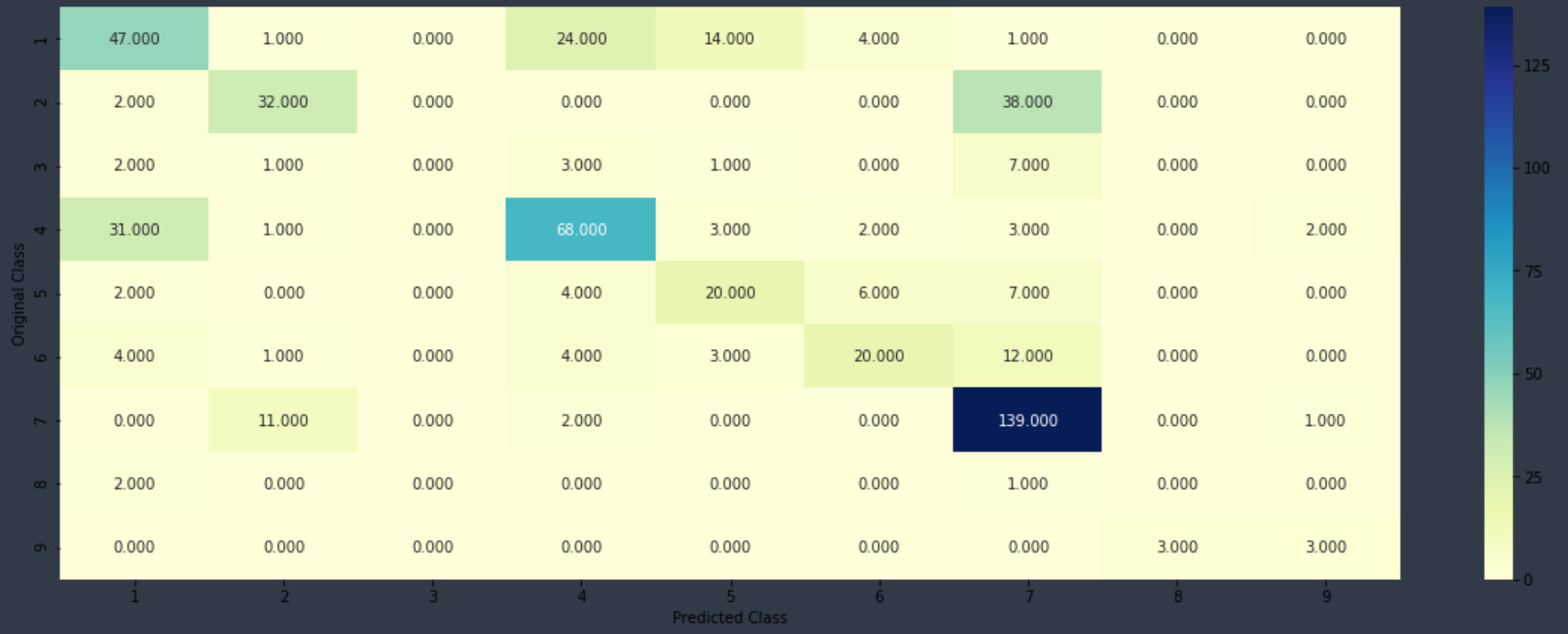
## Testing on the best Hyperparameter

In [97]:

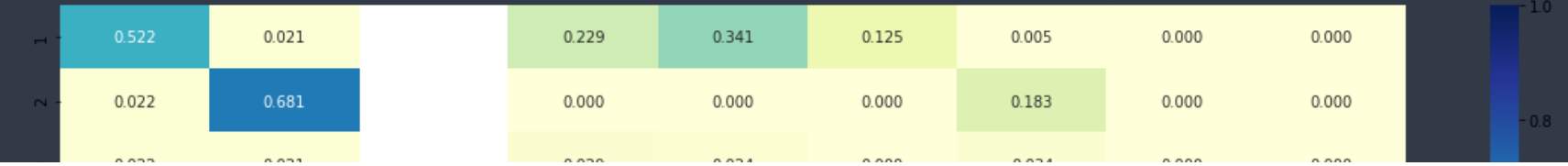
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
```

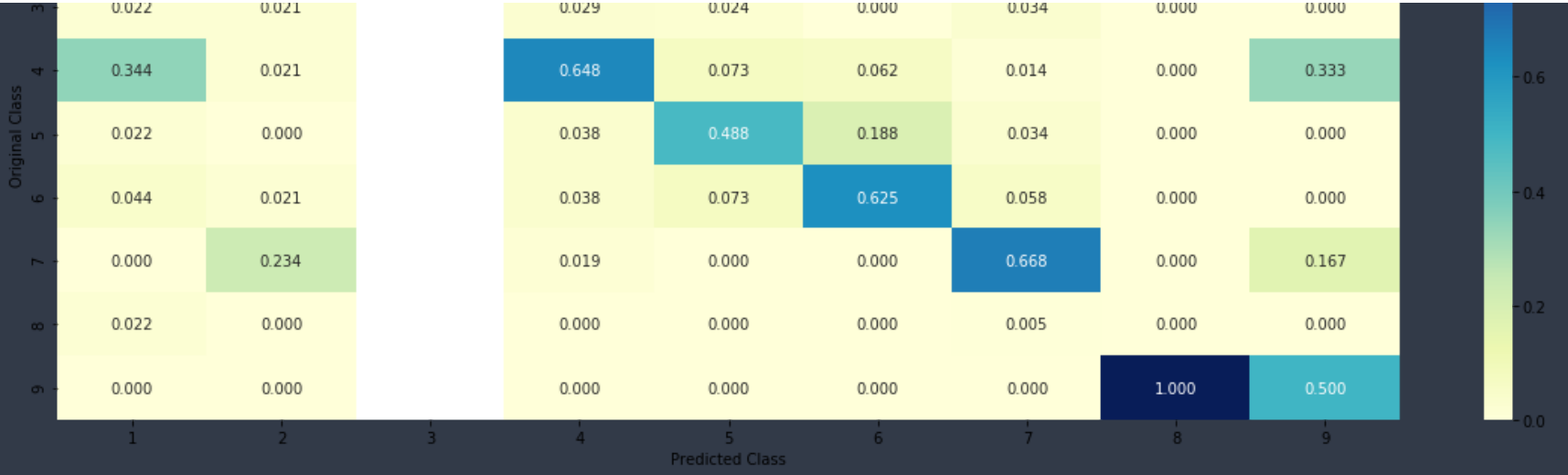
```
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.1838601734089547
Number of missclassified point : 0.3815789473684211
-------------------- Confusion matrix --------------------
```

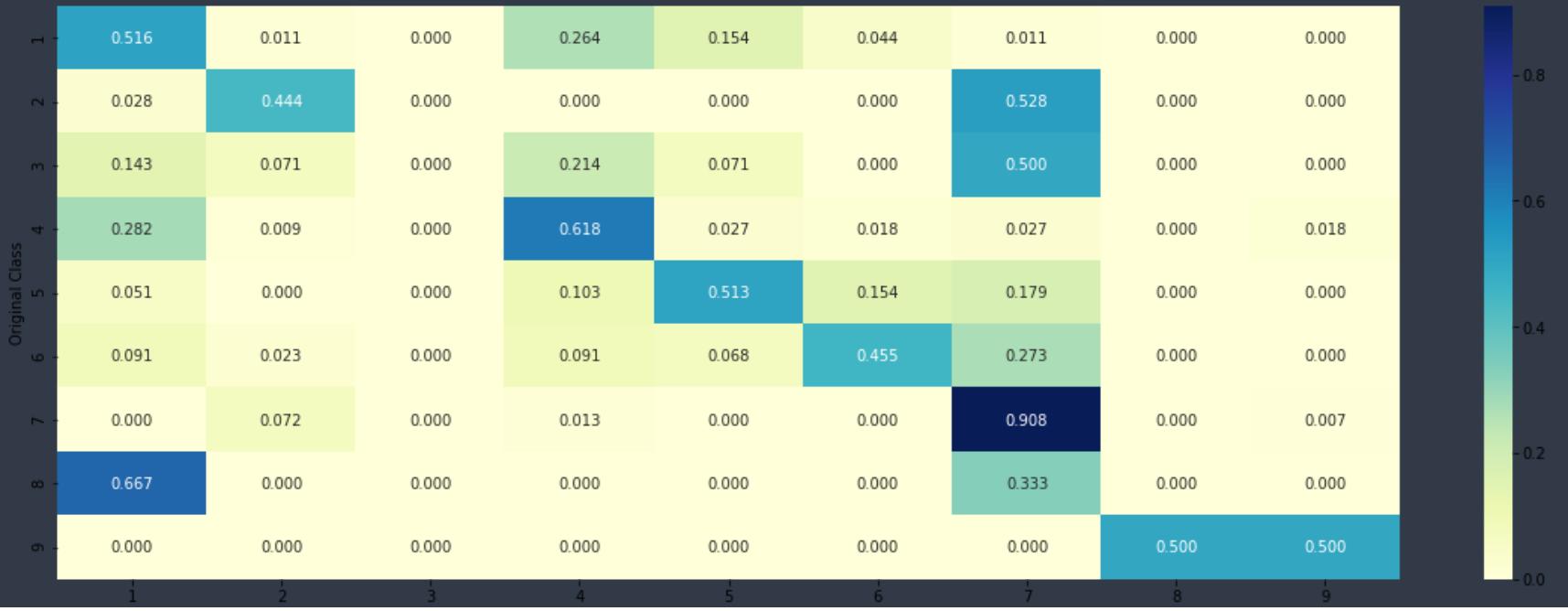



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



Predicted Class

## Important features of predicted points

In [98]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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 : 4
Predicted Class Probabilities: [[0.1066 0.0425 0.0109 0.6813 0.03   0.0309 0.093  0.0034 0.0012]]
Actual Class : 1
--------------------------------------------------
11 Text feature [activity] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [function] present in test data point [True]
14 Text feature [proteins] present in test data point [True]
15 Text feature [results] present in test data point [True]
16 Text feature [acid] present in test data point [True]
17 Text feature [missense] present in test data point [True]
19 Text feature [experiments] present in test data point [True]
20 Text feature [amino] present in test data point [True]
21 Text feature [whereas] present in test data point [True]
22 Text feature [functional] present in test data point [True]
24 Text feature [shown] present in test data point [True]
25 Text feature [mutations] present in test data point [True]
26 Text feature [also] present in test data point [True]
```

```
Text feature [two] present in test data point [True]
Text feature [type] present in test data point [True]
Text feature [may] present in test data point [True]
Text feature [whether] present in test data point [True]
Text feature [wild] present in test data point [True]
Text feature [related] present in test data point [True]
Text feature [reduced] present in test data point [True]
Text feature [described] present in test data point [True]
Text feature [three] present in test data point [True]
Text feature [vitro] present in test data point [True]
Text feature [indicated] present in test data point [True]
Text feature [important] present in test data point [True]
Text feature [determined] present in test data point [True]
Text feature [therefore] present in test data point [True]
Text feature [mammalian] present in test data point [True]
Text feature [suggesting] present in test data point [True]
Text feature [suppressor] present in test data point [True]
Text feature [indicate] present in test data point [True]
Text feature [ability] present in test data point [True]
Text feature [either] present in test data point [True]
Text feature [although] present in test data point [True]
Text feature [previously] present in test data point [True]
Text feature [see] present in test data point [True]
Text feature [one] present in test data point [True]
Text feature [analysis] present in test data point [True]
Text feature [containing] present in test data point [True]
Text feature [30] present in test data point [True]
Text feature [associated] present in test data point [True]
Text feature [discussion] present in test data point [True]
Text feature [show] present in test data point [True]
Text feature [purified] present in test data point [True]
Text feature [buffer] present in test data point [True]
Text feature [terminal] present in test data point [True]
Text feature [determine] present in test data point [True]
Text feature [several] present in test data point [True]
Text feature [assay] present in test data point [True]
Text feature [introduction] present in test data point [True]
Text feature [similar] present in test data point [True]
Text feature [expressed] present in test data point [True]
```

```
73 Text feature [substitutions] present in test data point [True]
74 Text feature [effect] present in test data point [True]
76 Text feature [addition] present in test data point [True]
77 Text feature [loss] present in test data point [True]
78 Text feature [bind] present in test data point [True]
79 Text feature [10] present in test data point [True]
80 Text feature [mutation] present in test data point [True]
81 Text feature [effects] present in test data point [True]
83 Text feature [mm] present in test data point [True]
84 Text feature [levels] present in test data point [True]
85 Text feature [suggested] present in test data point [True]
86 Text feature [using] present in test data point [True]
87 Text feature [could] present in test data point [True]
88 Text feature [vivo] present in test data point [True]
89 Text feature [suggest] present in test data point [True]
90 Text feature [critical] present in test data point [True]
91 Text feature [50] present in test data point [True]
92 Text feature [lower] present in test data point [True]
93 Text feature [phosphatase] present in test data point [True]
94 Text feature [however] present in test data point [True]
95 Text feature [tagged] present in test data point [True]
96 Text feature [due] present in test data point [True]
97 Text feature [affect] present in test data point [True]
98 Text feature [used] present in test data point [True]
Out of the top  100  features  77 are present in query point
```

# Logistic Regression with Class Balancing

## Hyperparameter Tuning

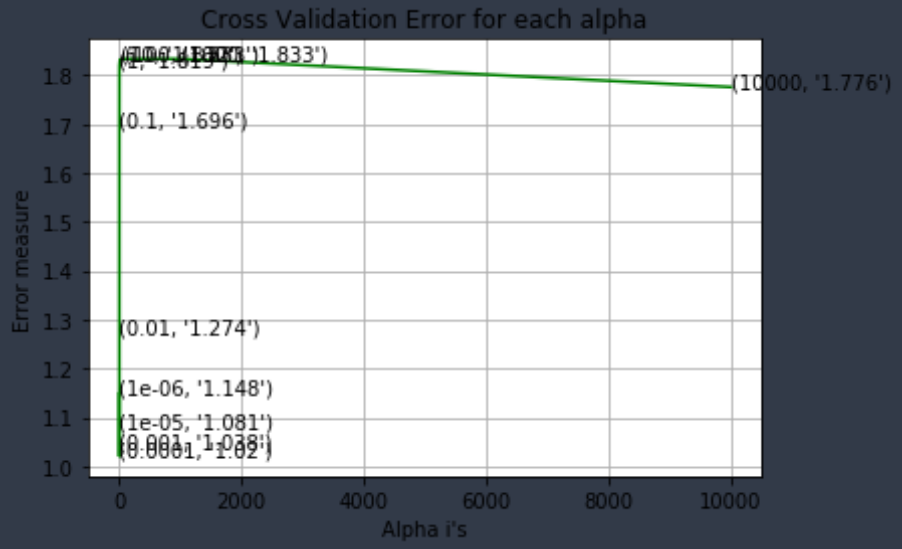
In [99]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
```

```
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",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 : 1.1483813396394302
for alpha = 1e-05
Log Loss : 1.080590316516733
for alpha = 0.0001
Log Loss : 1.020338696584149
for alpha = 0.001
Log Loss : 1.0376117684782866
for alpha = 0.01
Log Loss : 1.2738359674090283
for alpha = 0.1
Log Loss : 1.695814111124099
for alpha = 1
Log Loss : 1.8190567772972264
for alpha = 10
Log Loss : 1.8318428079300775
for alpha = 100
Log Loss : 1.8333961457620223
for alpha = 1000
Log Loss : 1.833305106494954
for alpha = 10000
Log Loss : 1.775690810242407
```




```
For values of best alpha =  0.0001 The train log loss is: 0.39807205200028006
For values of best alpha =  0.0001 The cross validation log loss is: 1.020338696584149
For values of best alpha =  0.0001 The test log loss is: 0.9755494609684295
```

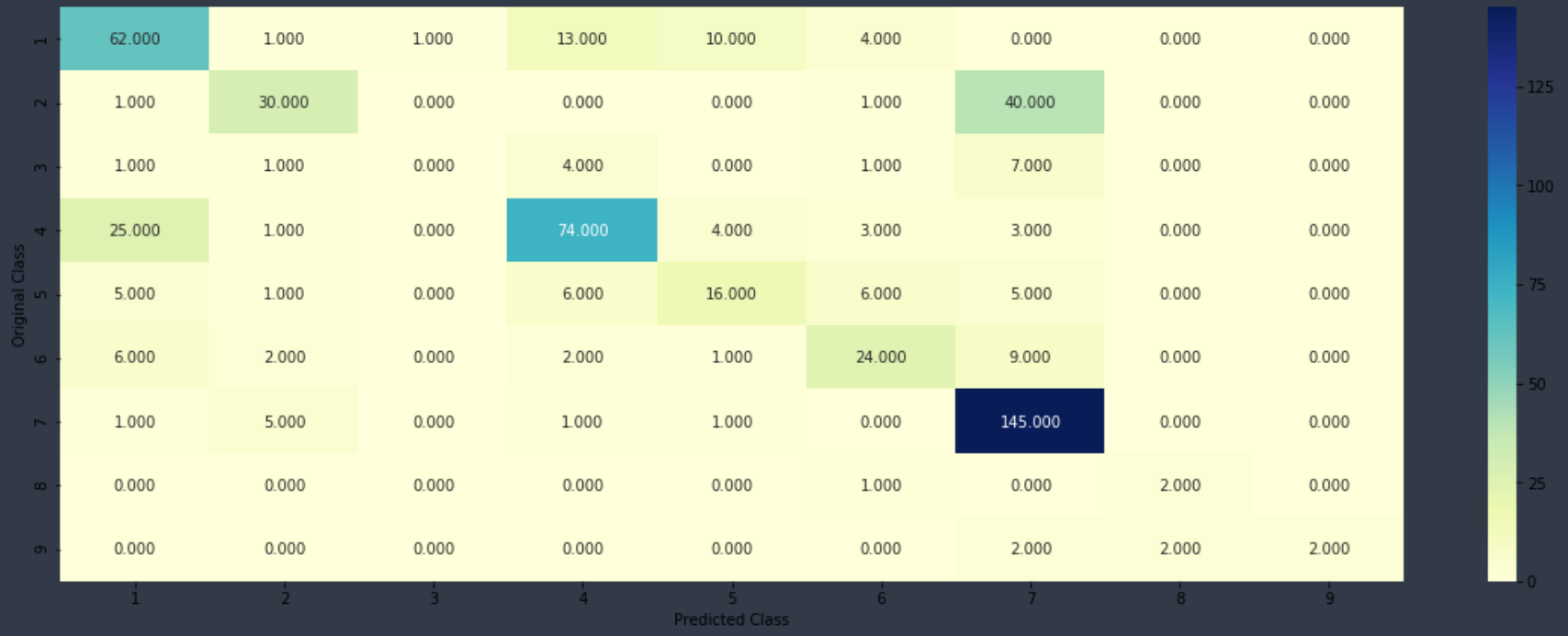
## Testing on best Hyperparameter

In [100]:

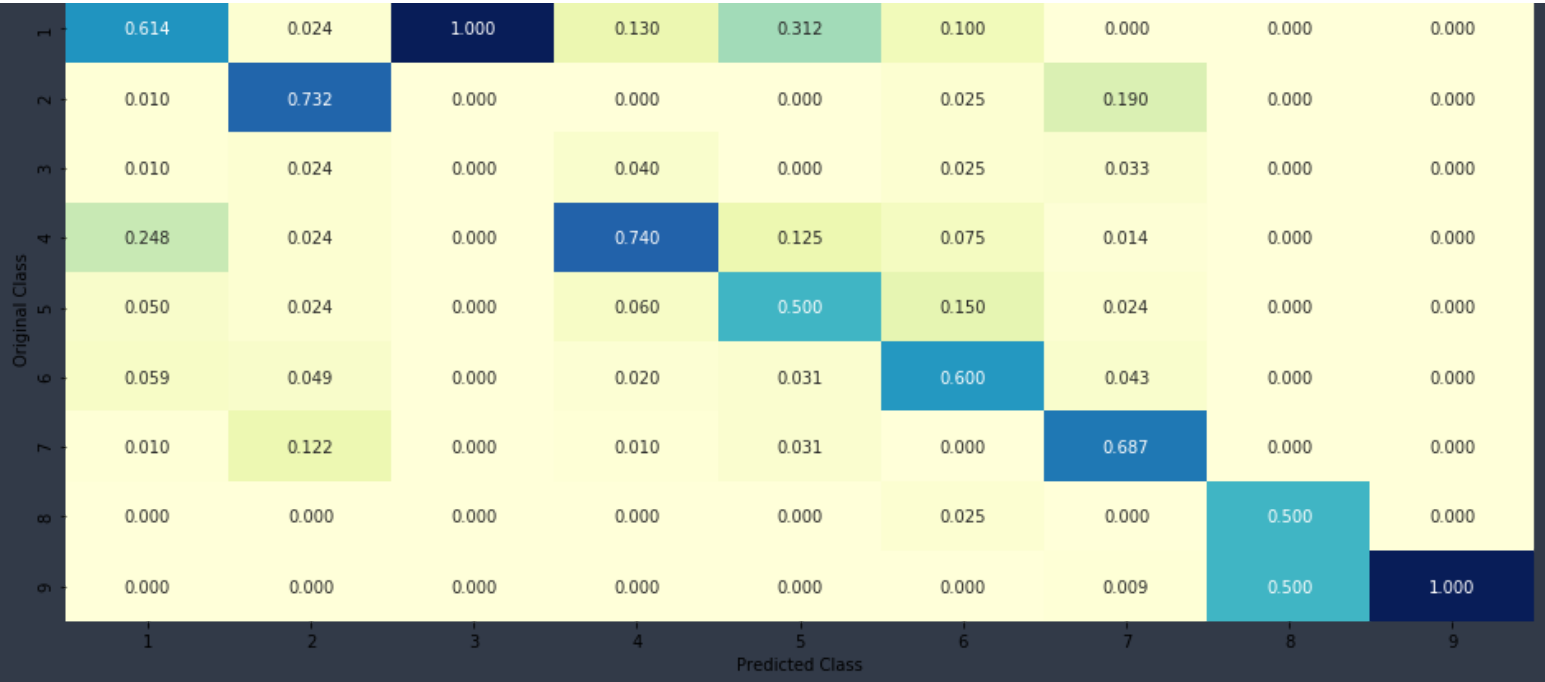
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
```

```
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

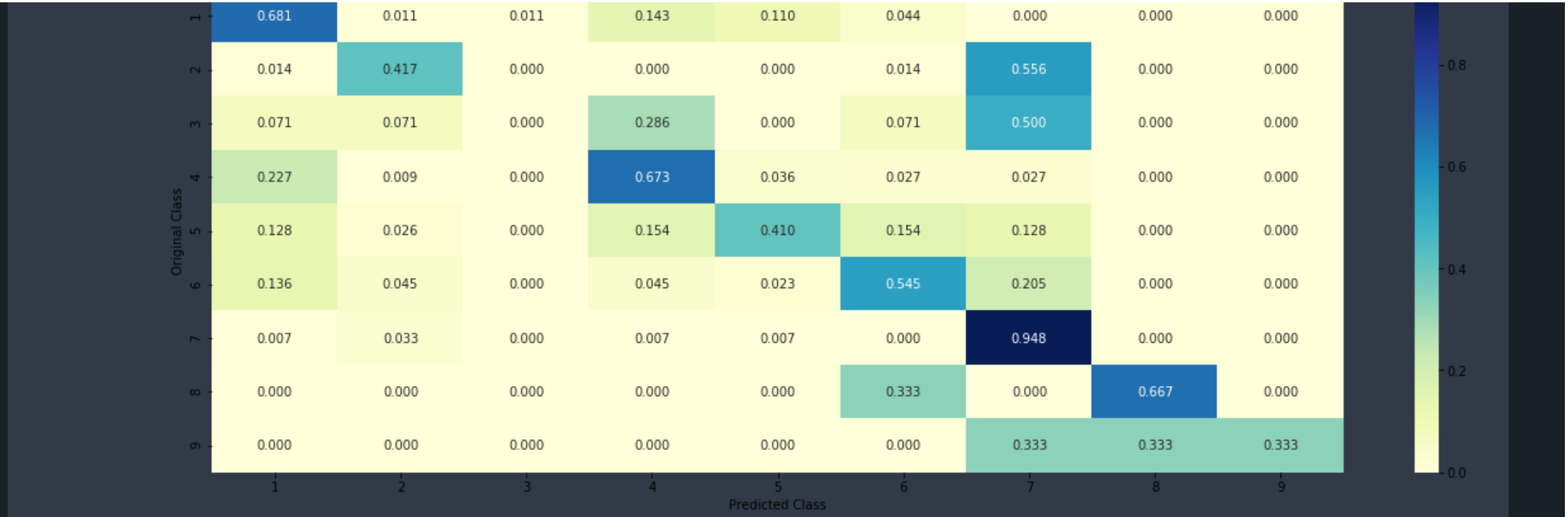
```
Log Loss : 1.024187932994131
Number of missclassified point : 0.33270676691729323
-------------------- Confusion matrix --------------------
```




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




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




## Important Features of Predicted point

In [101]:

```
test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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: [[0.3247 0.0061 0.0035 0.0864 0.5471 0.0177 0.0089 0.0047 0.0009]]
Actual Class : 5
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

# Logistic regression without class balancing

## Hyperparameter Tuning

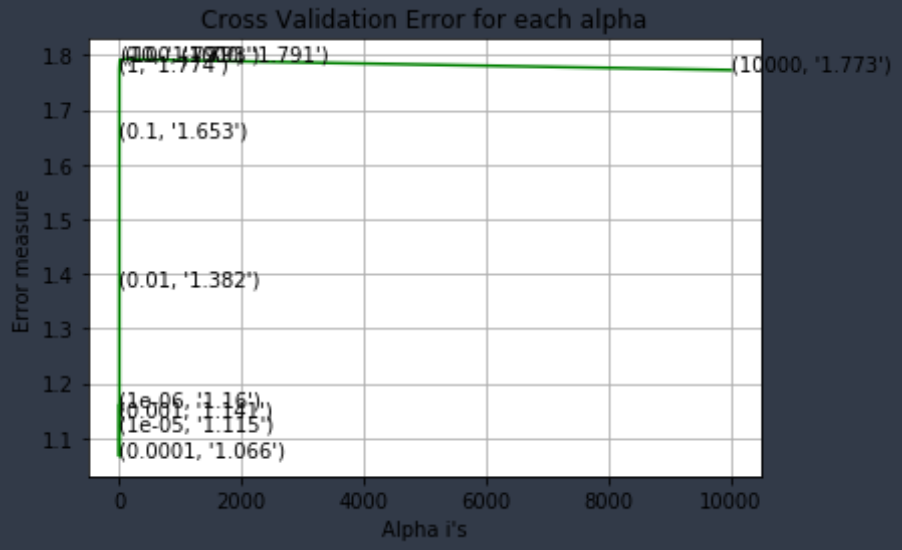
In [102]:

```
alpha = [10**i for i in range(-6,5) ]
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",cv=5)
    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 esti
mates
    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",cv=5)
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_los
s(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 i
s:",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 : 1.1595983053307697
for alpha = 1e-05
Log Loss : 1.1149181462024926
for alpha = 0.0001
Log Loss : 1.0664552571767256
for alpha = 0.001
Log Loss : 1.1413250951108032
for alpha = 0.01
Log Loss : 1.3816612980283973
for alpha = 0.1
```

```
Log Loss : 1.6533445062611152
for alpha = 1
Log Loss : 1.7744462997953399
for alpha = 10
Log Loss : 1.7912857862497737
for alpha = 100
Log Loss : 1.7932993661971908
for alpha = 1000
Log Loss : 1.790708652179575
for alpha = 10000
Log Loss : 1.7727425914531467
```

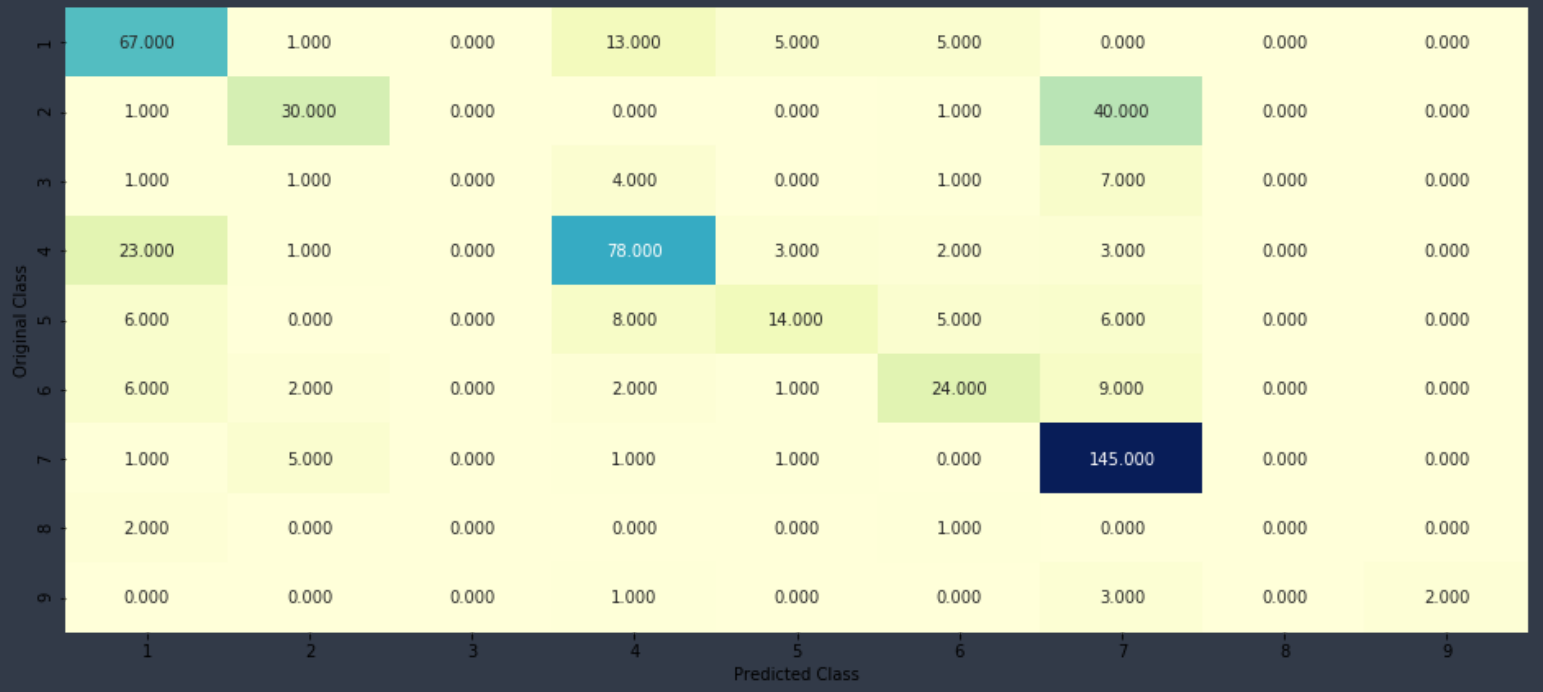



```
For values of best alpha =  0.0001 The train log loss is: 0.38661367108160877
For values of best alpha =  0.0001 The cross validation log loss is: 1.0664552571767256
For values of best alpha =  0.0001 The test log loss is: 0.9978097855044807
```

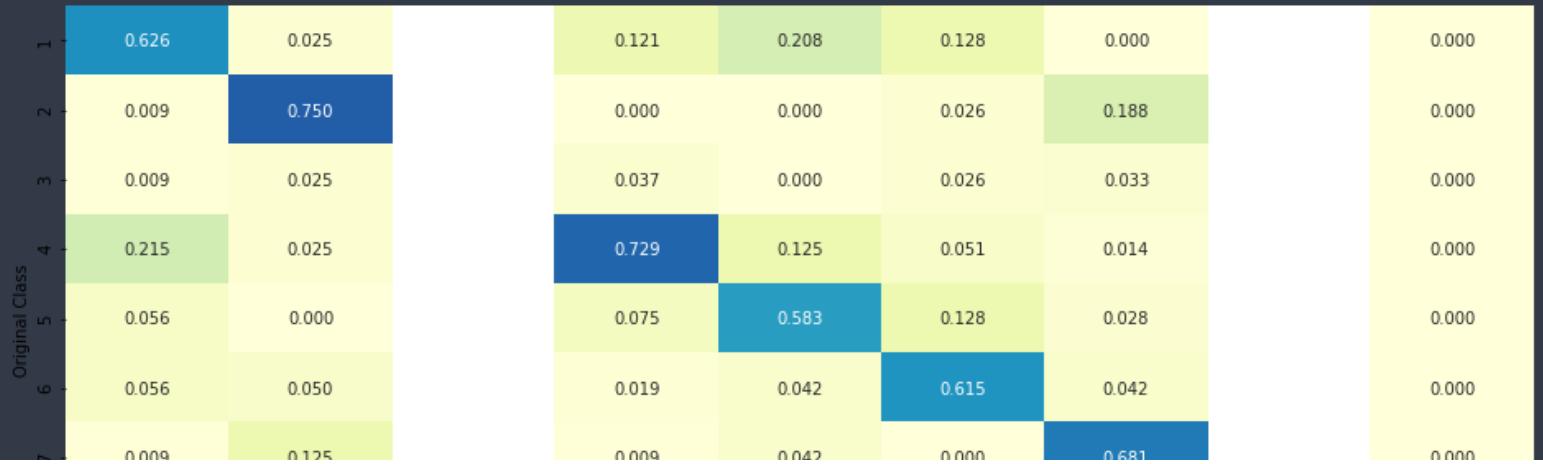
## Testing on Best Hyperparameter

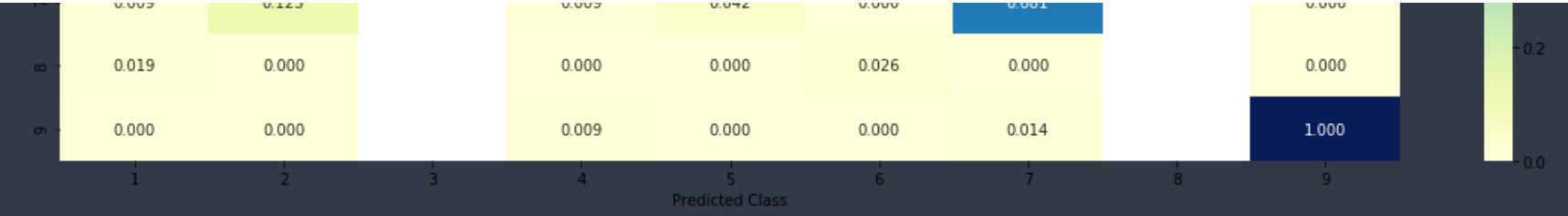
In [103]:
```
clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.0673379211842977
Number of missclassified point : 0.3233082706766917
-------------------- Confusion matrix --------------------
```

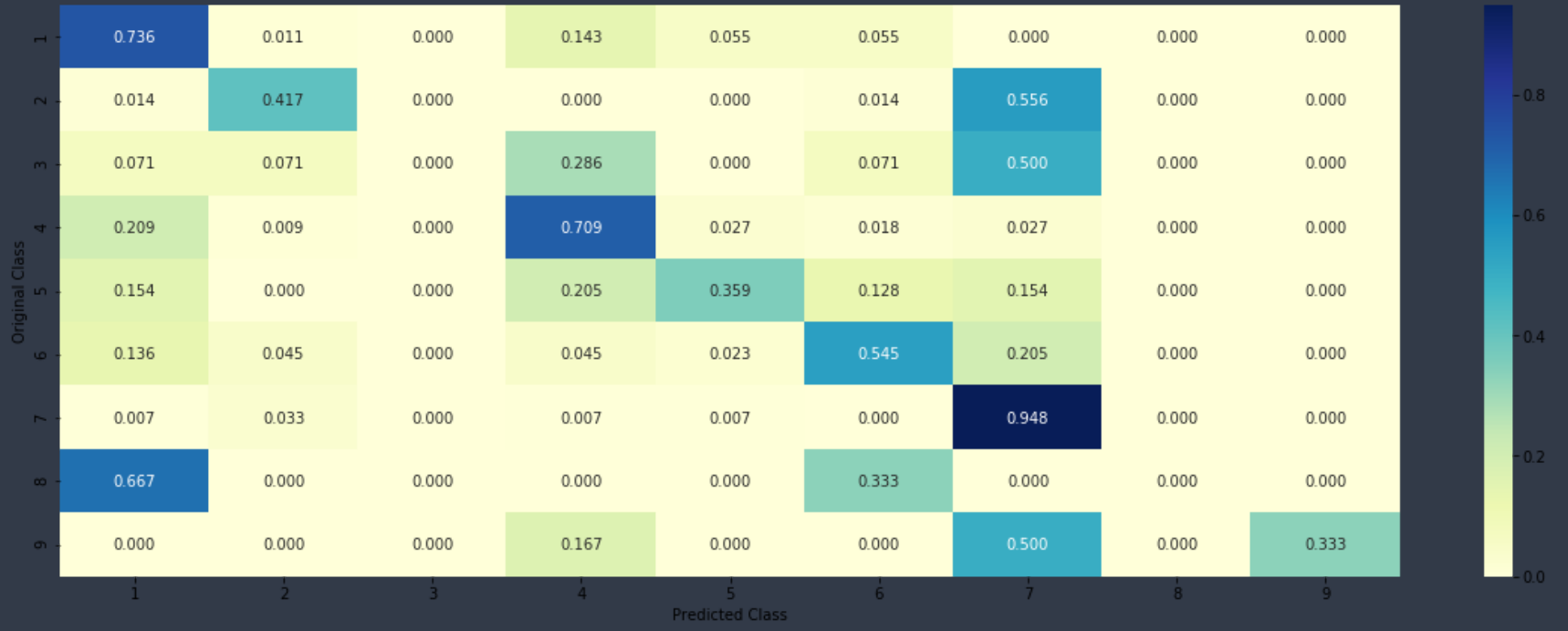



---------- Precision matrix (Columm Sum=1) ----------






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




In [104]:

```
test_point_index = 200
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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 : 6
Predicted Class Probabilities: [[7.000e-03 1.980e-02 1.100e-03 4.400e-03 2.300e-03 8.874e-01 7.440e-02
  3.400e-03 2.000e-04]]
Actual Class : 6
--------------------------------------------------
74 Text feature [resistance] present in test data point [True]
76 Text feature [blue] present in test data point [True]
84 Text feature [substrate] present in test data point [True]
85 Text feature [values] present in test data point [True]
96 Text feature [substitutions] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

## Linear Support Vector Machine

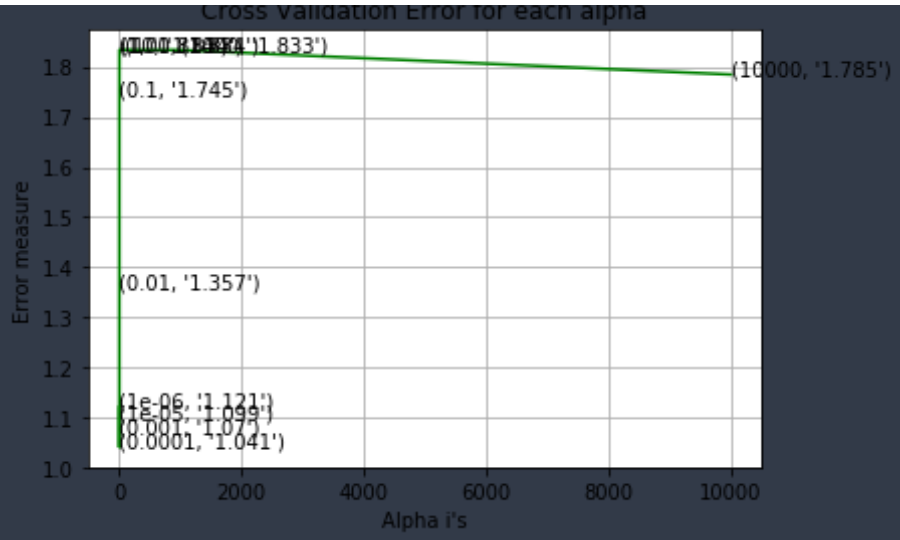
In [105]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hinge', ra
ndom_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",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 : 1.1208239417967307
for alpha = 1e-05
Log Loss : 1.0994136603052724
for alpha = 0.0001
Log Loss : 1.0406999499342668
for alpha = 0.001
Log Loss : 1.0695052100763967
for alpha = 0.01
Log Loss : 1.3570272562045098
for alpha = 0.1
Log Loss : 1.7454466615623678
for alpha = 1
Log Loss : 1.8336985155093224
for alpha = 10
Log Loss : 1.833698928812389
for alpha = 100
Log Loss : 1.8337615980319724
for alpha = 1000
Log Loss : 1.8334121716371192
for alpha = 10000
Log Loss : 1.7845278136454599
```




```
For values of best alpha =  0.0001 The train log loss is: 0.3135211764531792
For values of best alpha =  0.0001 The cross validation log loss is: 1.0406999499342668
For values of best alpha =  0.0001 The test log loss is: 1.009727395167734
```

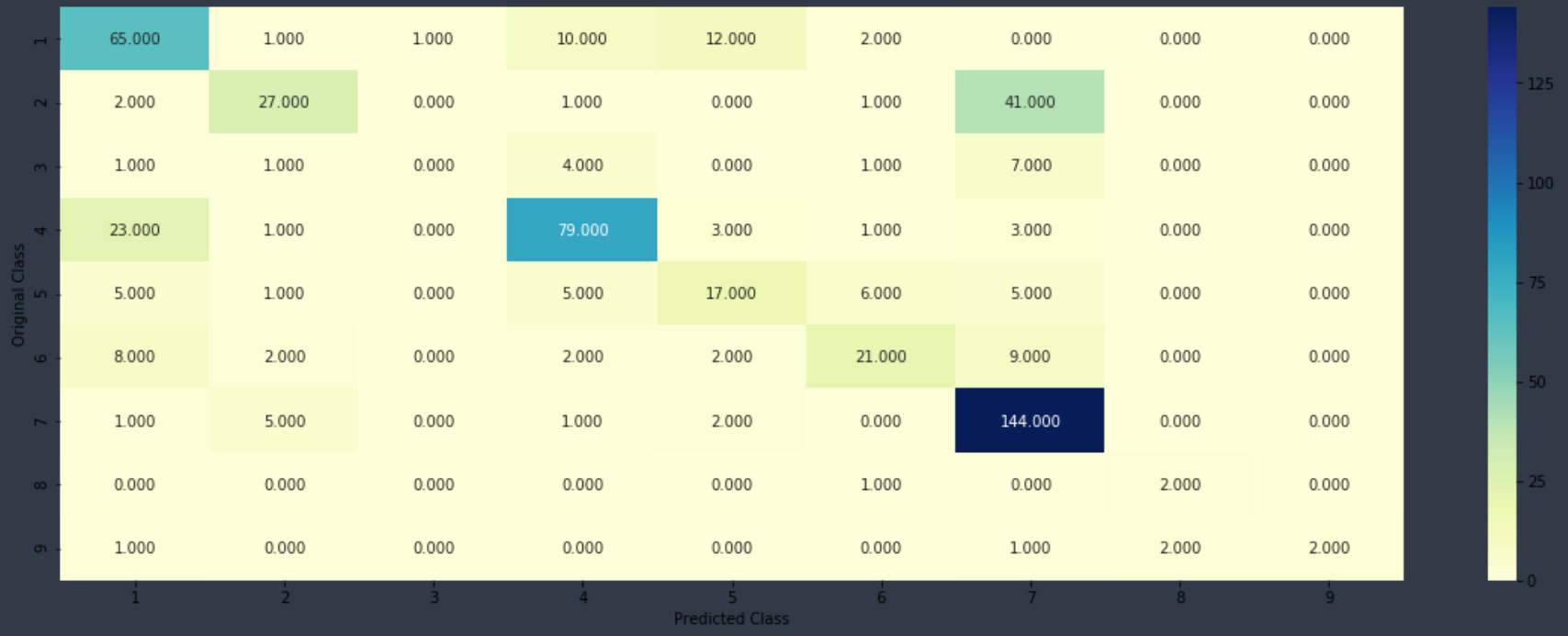
## Testing on best Hyperparameter

In [106]:

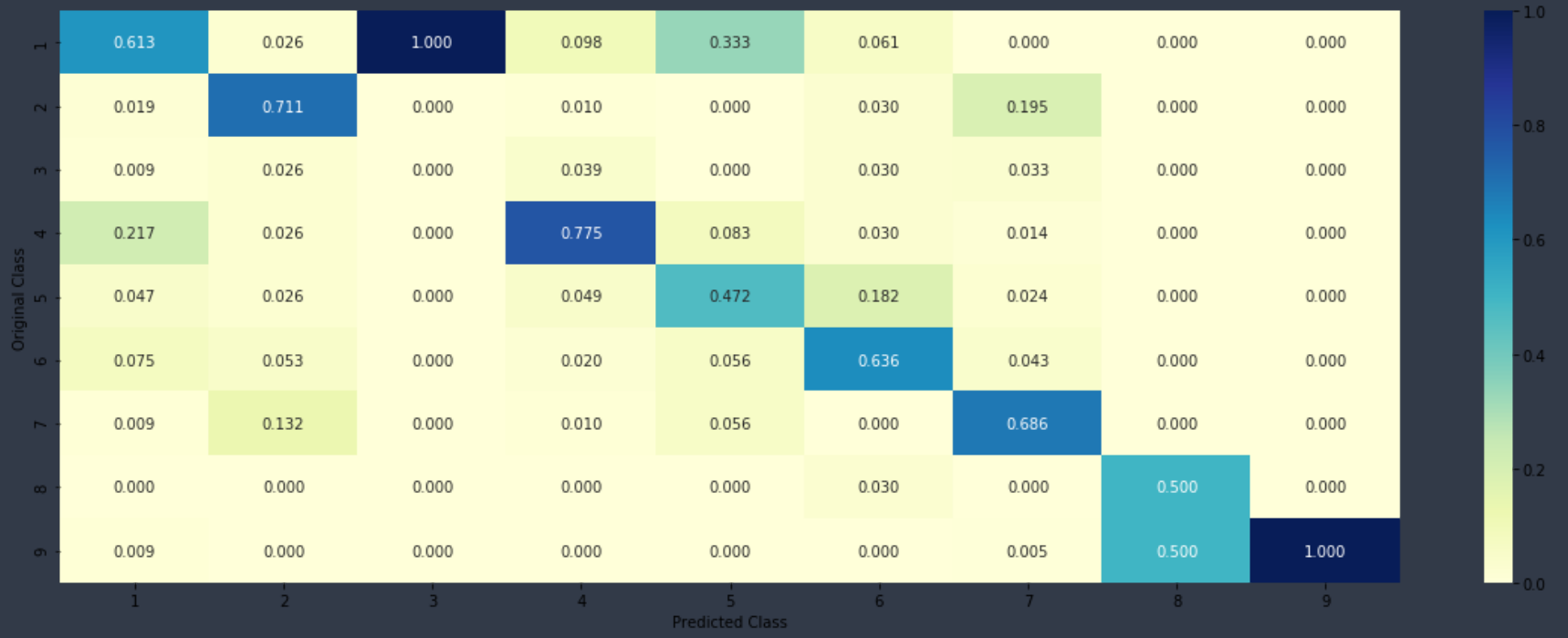
```
clf = SGDClassifier(class_weight='balanced' ,alpha=alpha[best_alpha], penalty='l2', loss
='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
```

```
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

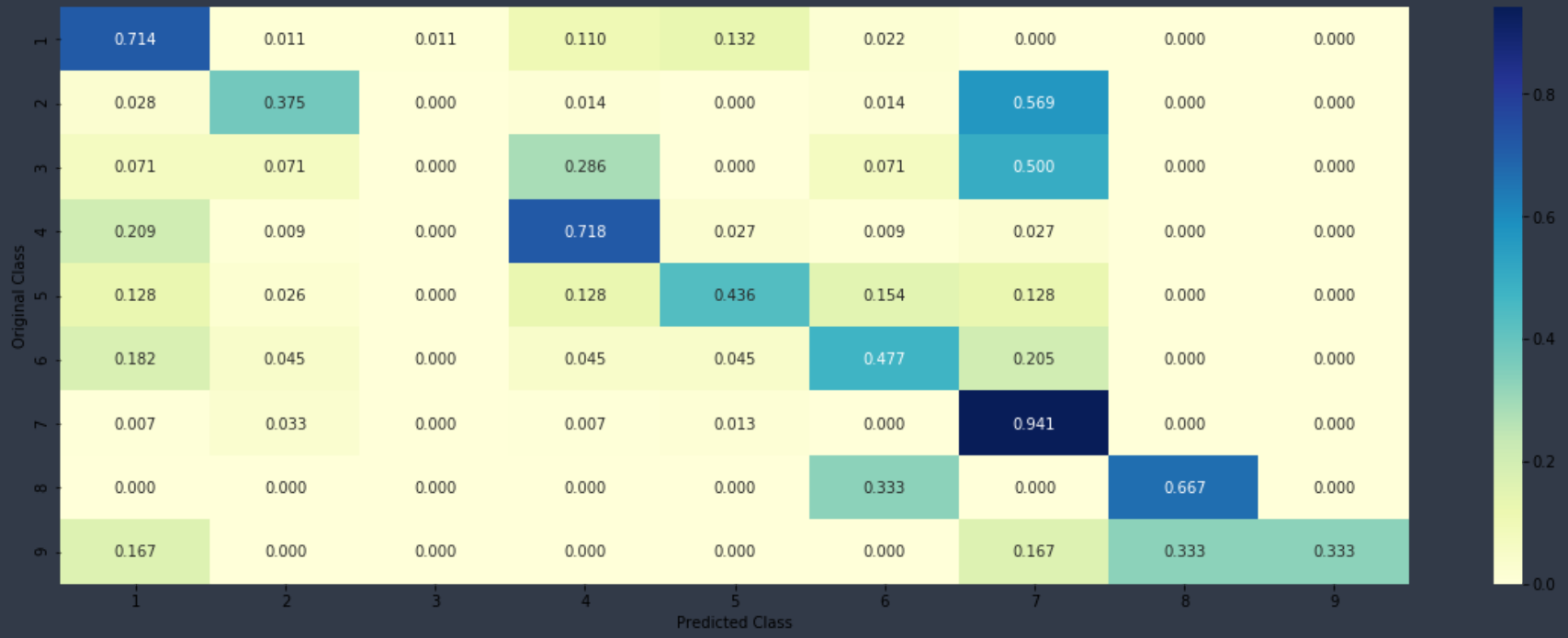
```
Log Loss : 1.0596762941785134
Number of missclassified point : 0.32894736842105265
-------------------- Confusion matrix --------------------
```




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




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




## Important features for predicted point

In [107]:

```
test_point_index = 500
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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: [[0.6812 0.0445 0.0054 0.2092 0.0221 0.0121 0.0228 0.0017 0.0009]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

## Random Forest Classifier

In [108]:

```
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, rand
om_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv =None)
        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))


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_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",cv =None)
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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.209407337503811
for n_estimators = 100 and max depth =  10
Log Loss : 1.2513748293950913
for n_estimators = 200 and max depth =  5
Log Loss : 1.1963101858583827
for n_estimators = 200 and max depth =  10
Log Loss : 1.2386892509905536
for n_estimators = 500 and max depth =  5
Log Loss : 1.1834490421994914
for n_estimators = 500 and max depth =  10
Log Loss : 1.2310880114255751
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1794425781115954
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2265477196682208
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1786100566582702
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2268340203419448
For values of best estimator =  2000 The train log loss is: 0.8448989344194195
```

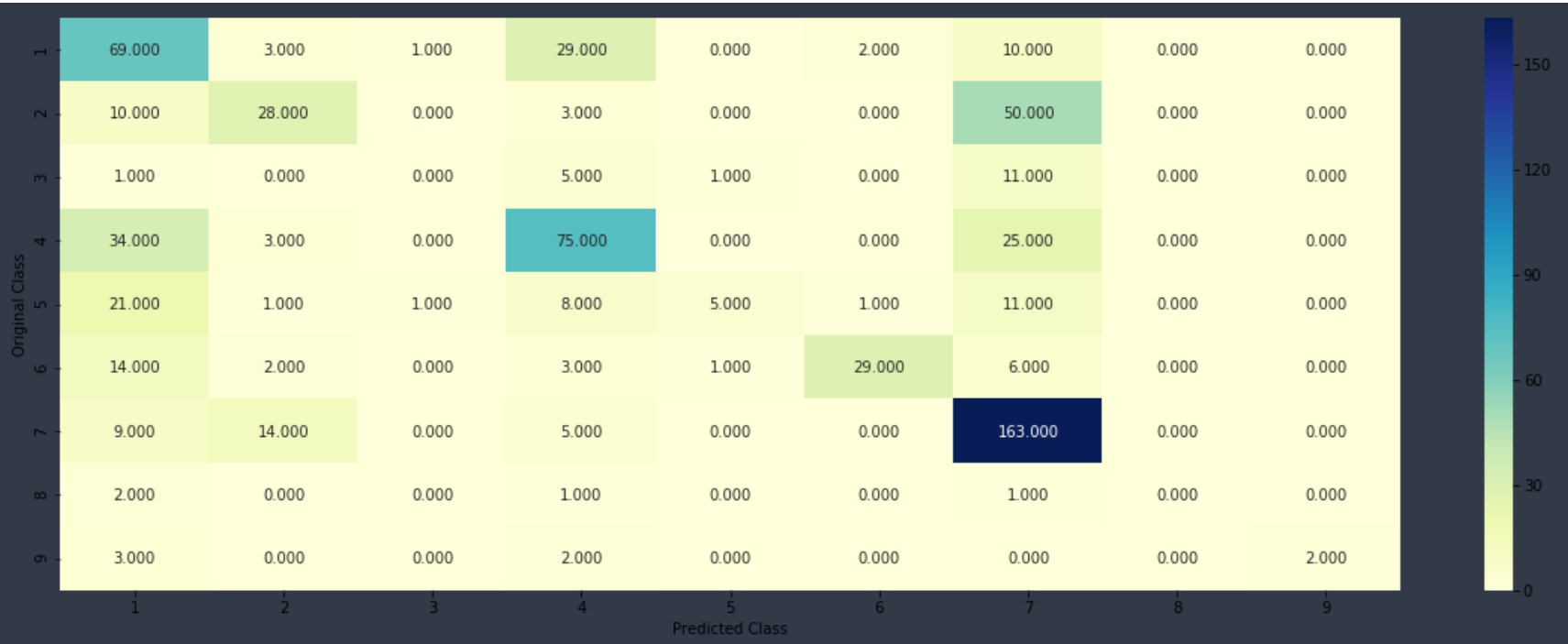
```
For values of best estimator =  2000 The cross validation log loss is: 1.1786100566582705
For values of best estimator =  2000 The test log loss is: 1.1912866311548105
```

## Testing on best Hyperparameter

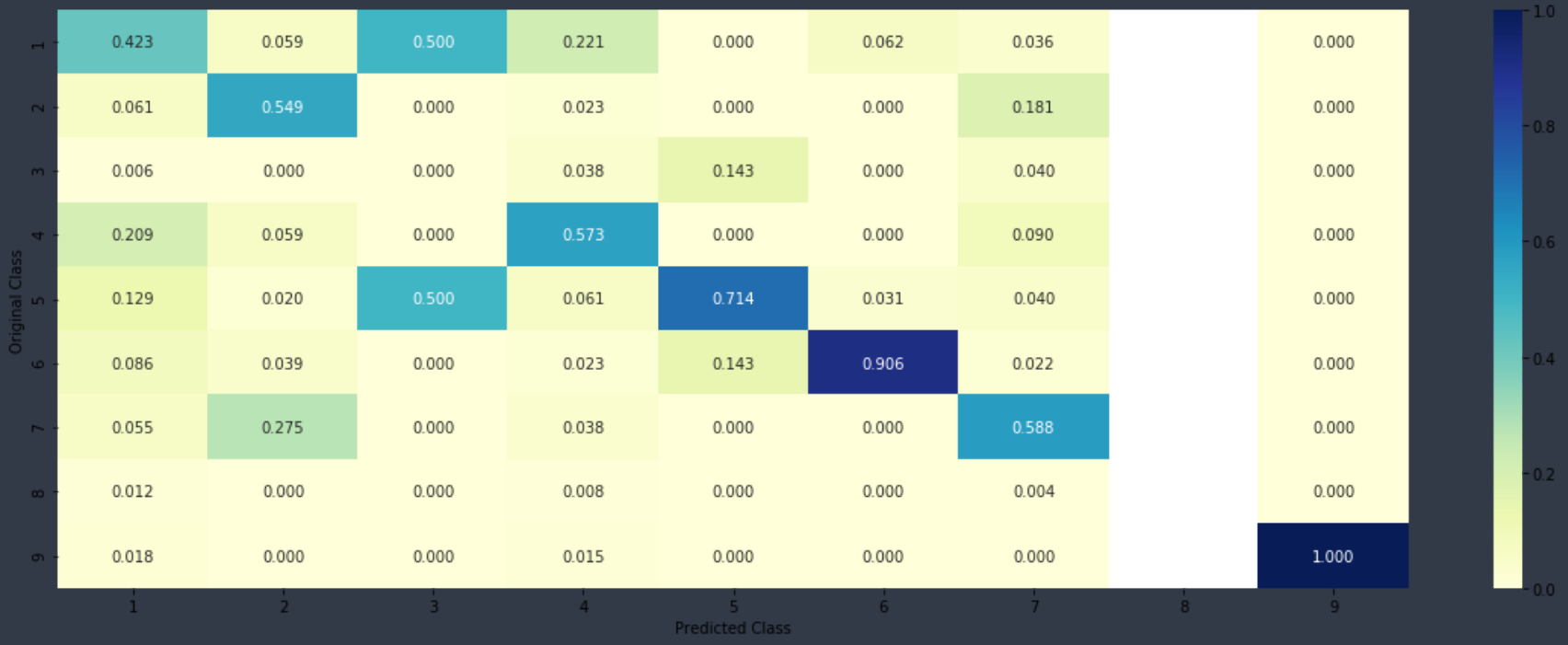
In [109]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_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",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

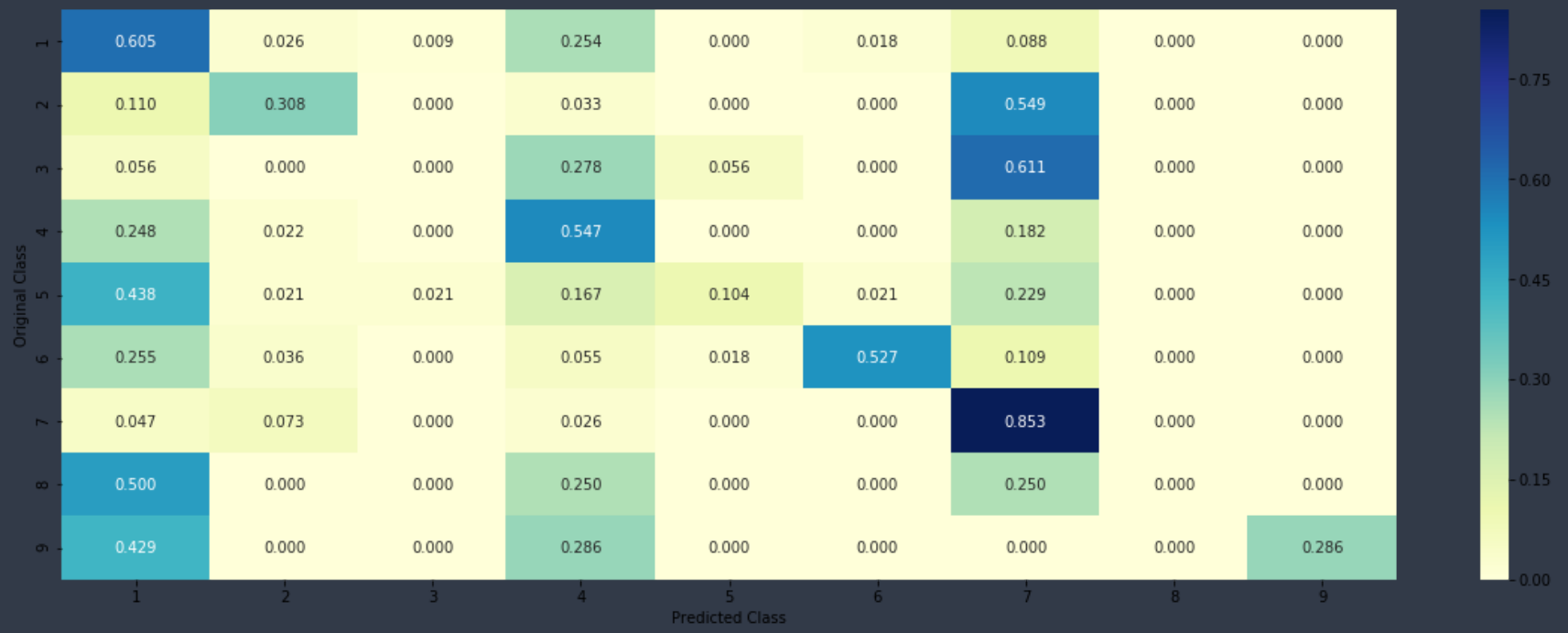
```
Log Loss : 1.1912866311548107
Number of missclassified point : 0.4421052631578947
-------------------- Confusion matrix --------------------
```




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




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




## Feature importance of predicted point

In [110]:

```
test_point_index = 120
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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: [[0.0514 0.2353 0.0204 0.0389 0.0469 0.0433 0.554  0.0068 0.0029]]
Actual Class : 5
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitors] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
6 Text feature [activation] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [constitutive] present in test data point [True]
10 Text feature [loss] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
15 Text feature [functional] present in test data point [True]
17 Text feature [protein] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
20 Text feature [inhibitor] present in test data point [True]
22 Text feature [drug] present in test data point [True]
23 Text feature [erk] present in test data point [True]
24 Text feature [trials] present in test data point [True]
25 Text feature [pten] present in test data point [True]
26 Text feature [variants] present in test data point [True]
28 Text feature [therapeutic] present in test data point [True]
31 Text feature [therapy] present in test data point [True]
33 Text feature [expression] present in test data point [True]
35 Text feature [months] present in test data point [True]
37 Text feature [resistance] present in test data point [True]
39 Text feature [activate] present in test data point [True]
42 Text feature [treated] present in test data point [True]
43 Text feature [transforming] present in test data point [True]
45 Text feature [akt] present in test data point [True]
```

```
47 Text feature [cells] present in test data point [True]
48 Text feature [57] present in test data point [True]
49 Text feature [cell] present in test data point [True]
50 Text feature [downstream] present in test data point [True]
51 Text feature [proteins] present in test data point [True]
55 Text feature [growth] present in test data point [True]
56 Text feature [advanced] present in test data point [True]
57 Text feature [patients] present in test data point [True]
58 Text feature [ic50] present in test data point [True]
59 Text feature [variant] present in test data point [True]
61 Text feature [inhibition] present in test data point [True]
63 Text feature [survival] present in test data point [True]
64 Text feature [oncogene] present in test data point [True]
66 Text feature [inhibited] present in test data point [True]
72 Text feature [sensitivity] present in test data point [True]
73 Text feature [response] present in test data point [True]
74 Text feature [ovarian] present in test data point [True]
75 Text feature [phospho] present in test data point [True]
76 Text feature [activity] present in test data point [True]
79 Text feature [predicted] present in test data point [True]
80 Text feature [inactivation] present in test data point [True]
82 Text feature [affected] present in test data point [True]
84 Text feature [egfr] present in test data point [True]
85 Text feature [clinical] present in test data point [True]
87 Text feature [expected] present in test data point [True]
88 Text feature [lines] present in test data point [True]
89 Text feature [ras] present in test data point [True]
90 Text feature [lung] present in test data point [True]
95 Text feature [based] present in test data point [True]
96 Text feature [dose] present in test data point [True]
97 Text feature [serum] present in test data point [True]
98 Text feature [values] present in test data point [True]
99 Text feature [dna] present in test data point [True]
Out of the top  100  features  61 are present in query point
```

## Stacking the models

In [111]:
```
clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', class_weight='balanced', ra
ndom_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid",cv=None)

clf2 = SGDClassifier(alpha=.0001, penalty='l2', loss='hinge', class_weight='balanced', r
andom_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid",cv=None)


clf3 = MultinomialNB(alpha=0.01)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(
cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pro
ba(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_oneh
otCoding))))
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_classifie
```

```
r=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_lo
ss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.03
Support vector machines : Log Loss: 1.07
Naive Bayes : Log Loss: 1.18
-------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.171
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.974
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.367
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.119
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.376
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.809
```

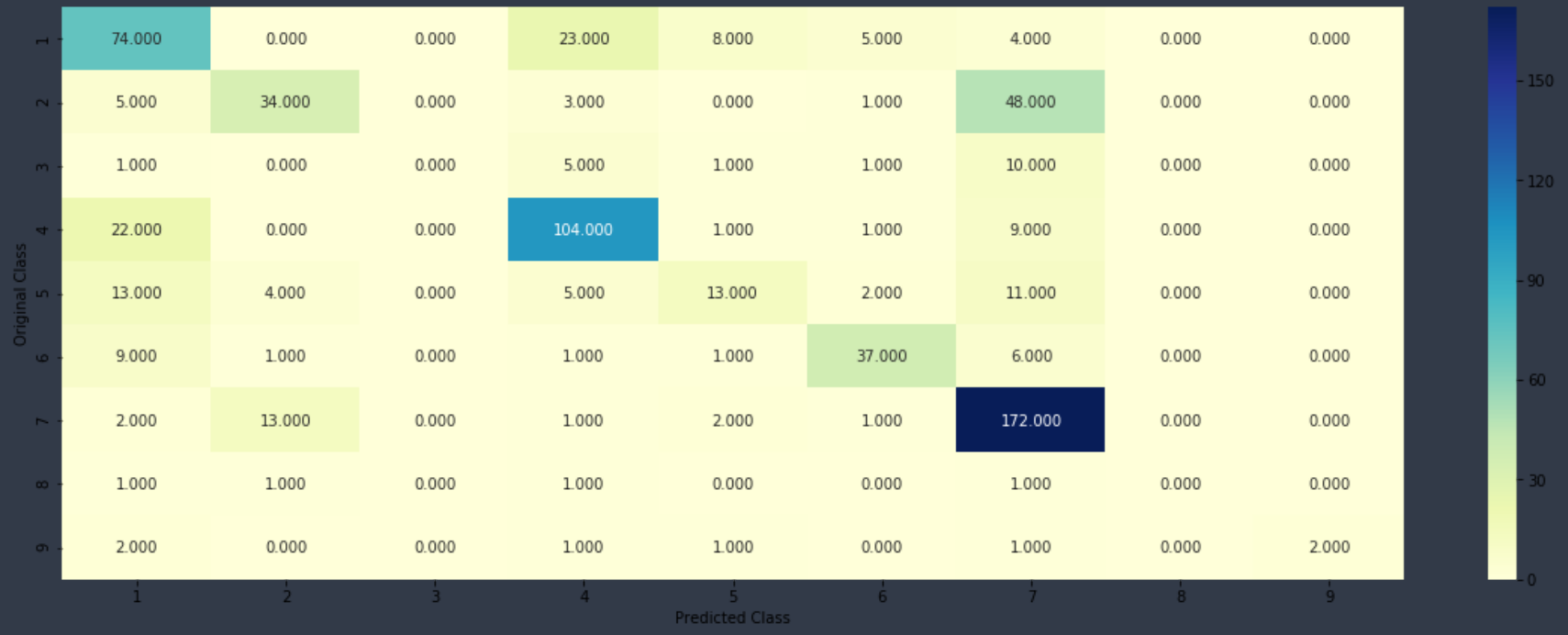
## Testing on the best Hyperparameter

In [112]:

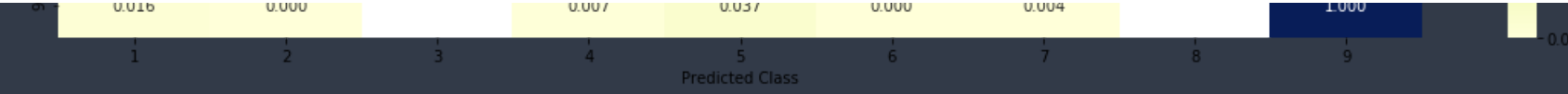
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr
, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error = log_loss(test_y, sclf.predict_proba(test_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_onehotCo
ding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

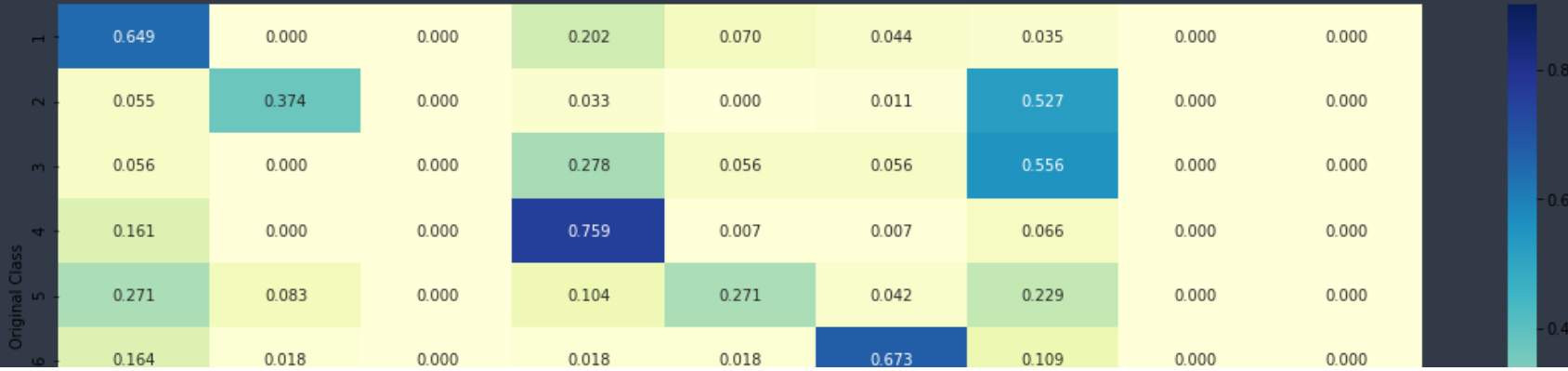
```
Log loss (train) on the stacking classifier : 1.084326892504591
Log loss (CV) on the stacking classifier : 1.084326892504591
Log loss (test) on the stacking classifier : 1.084326892504591
Number of missclassified point : 0.3443609022556391
-------------------- Confusion matrix --------------------
```

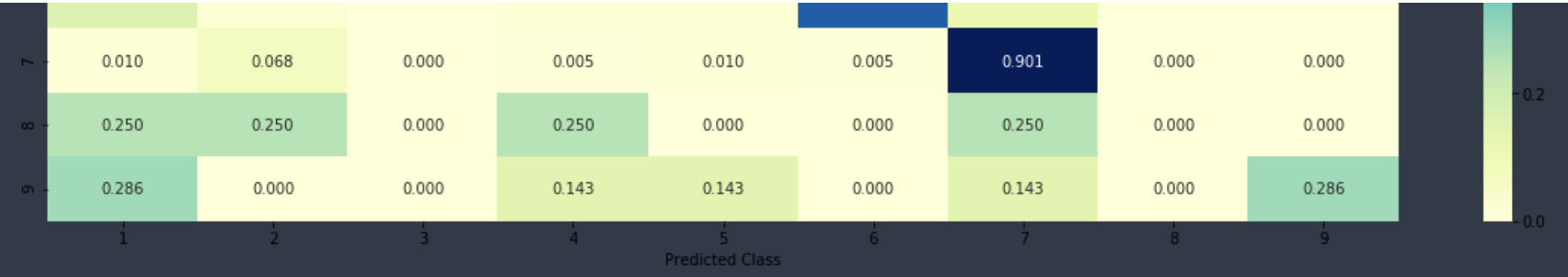



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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

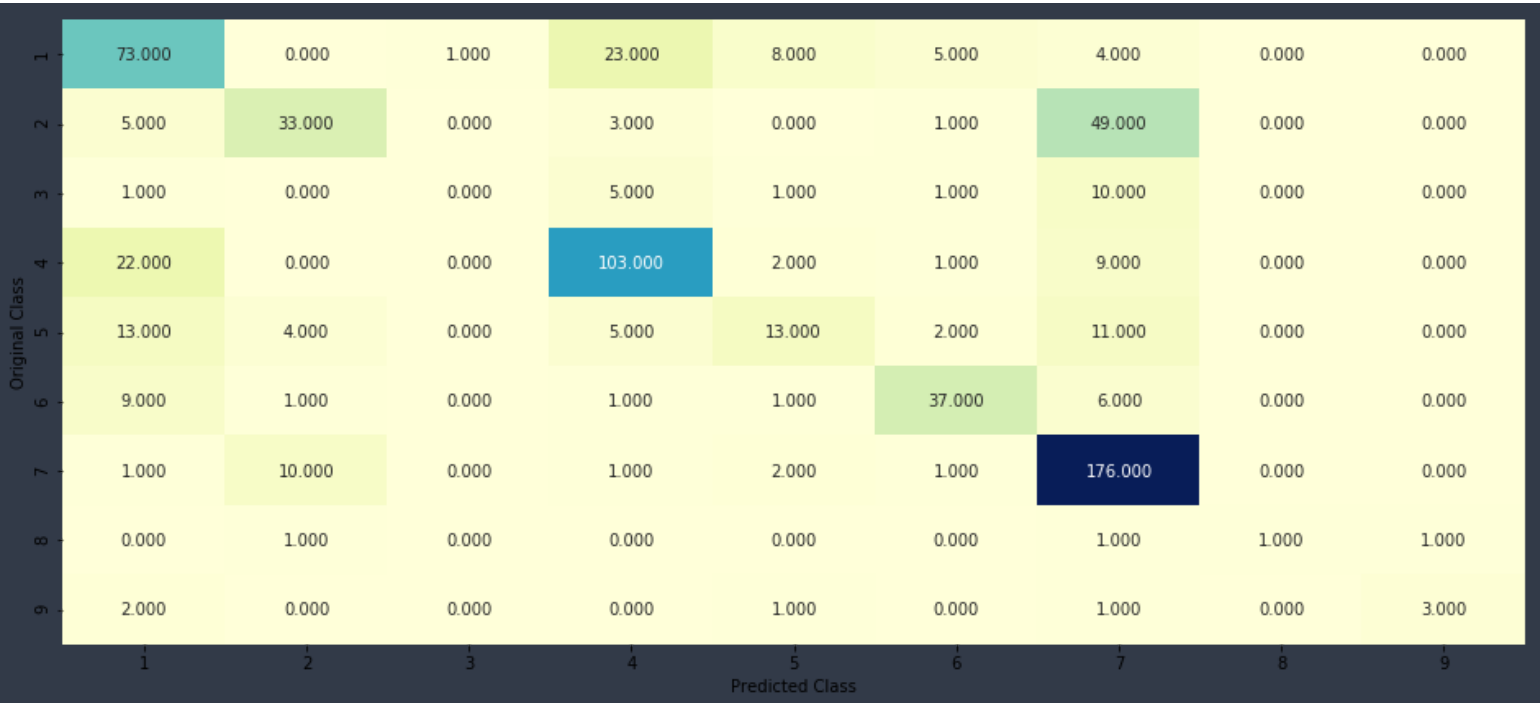




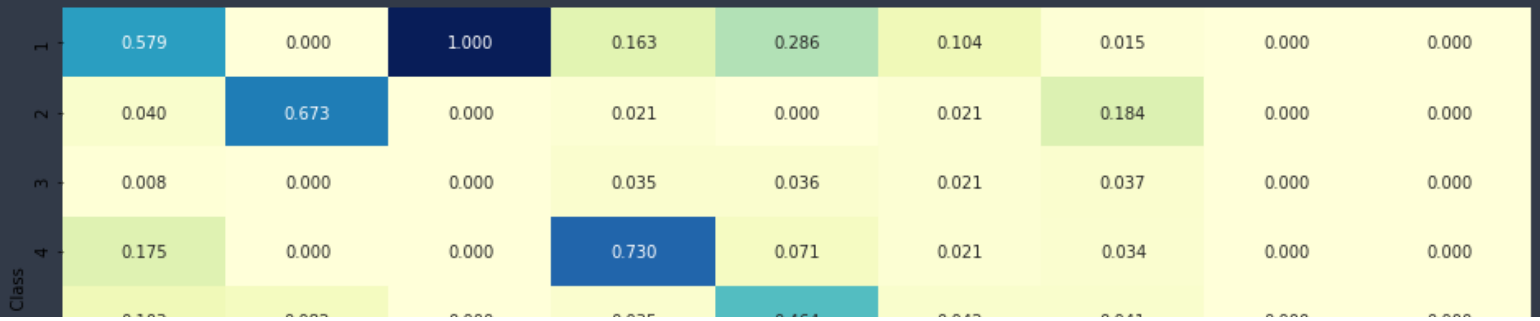

# Voting Classifier

In [113]:
```
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(t
est_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCo
ding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

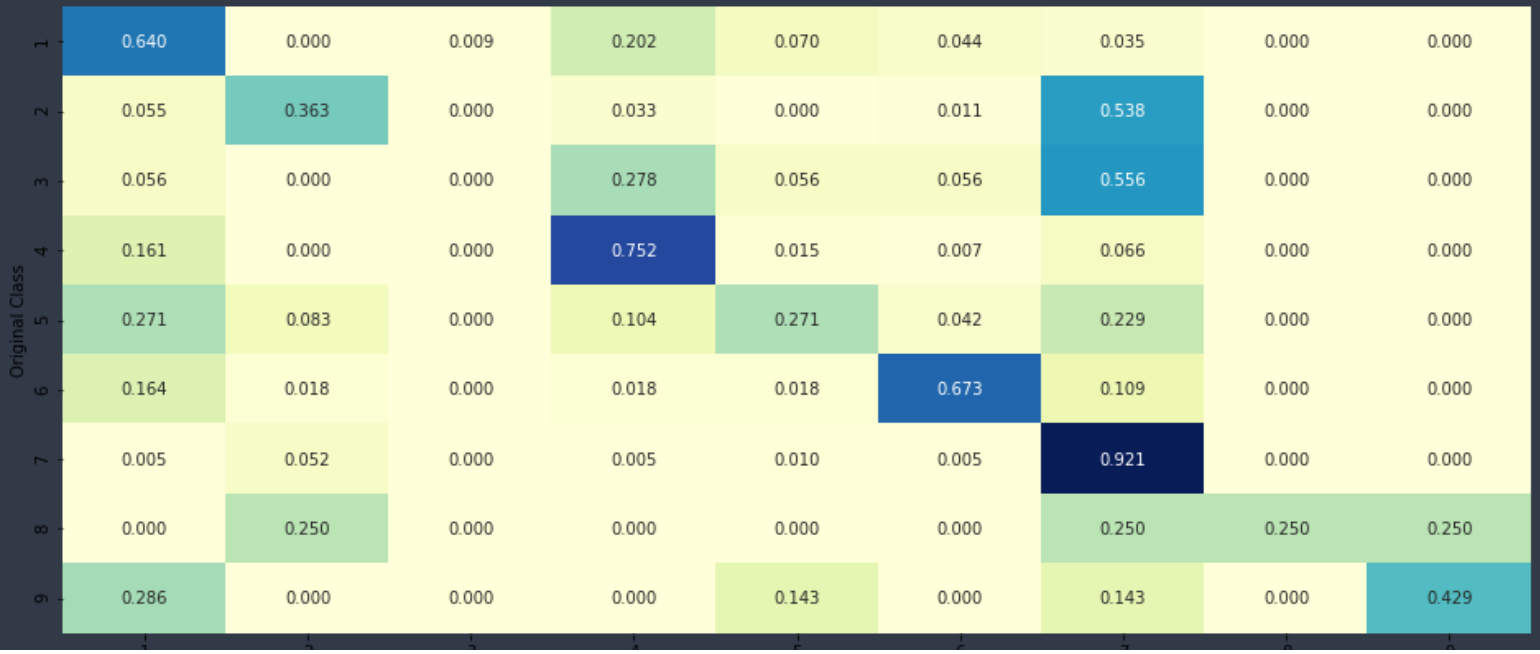
```
Log loss (train) on the VotingClassifier : 0.45591203089212384
Log loss (CV) on the VotingClassifier : 1.0380479197304984
Log loss (test) on the VotingClassifier : 1.010861041973004
Number of missclassified point : 0.3398496240601504
-------------------- Confusion matrix --------------------
```

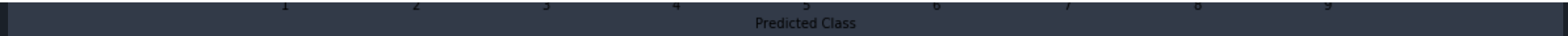



---------- Precision matrix (Columm Sum=1) ----------




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






## 2 Applying Logistic Regression to count Vectorizer inclusing bigrams and unigrams

In [114]:
```
text_vectorizer = CountVectorizer(ngram_range=(1,2),min_df=3,max_features=5000)
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*numbe
r of features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it o
ccured
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 : 5000
```

In [115]:
```
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 [116]:

```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_fe
ature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
re_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one
hotCoding))

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

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

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



print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
```

```
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 7198)
(number of data points * number of features) in test data =  (665, 7198)
(number of data points * number of features) in cross validation data = (532, 7198)
```

# Logistic Regression

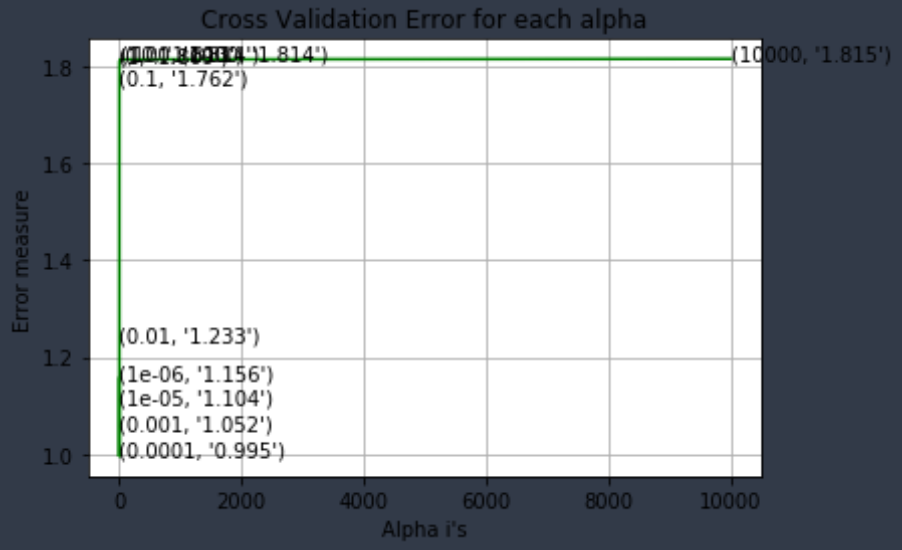
In [117]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",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 : 1.1559159898860694
for alpha = 1e-05
Log Loss : 1.1041450548664682
for alpha = 0.0001
Log Loss : 0.9952544440712352
```

```
for alpha = 0.001

Log Loss : 1.0517451600231427
for alpha = 0.01
Log Loss : 1.232507179896973
for alpha = 0.1
Log Loss : 1.7617611759229794
for alpha = 1
Log Loss : 1.8086425015811496
for alpha = 10
Log Loss : 1.8132750154433848
for alpha = 100
Log Loss : 1.813747049392956
for alpha = 1000
Log Loss : 1.813805360317339
for alpha = 10000
Log Loss : 1.8148637544348285
```



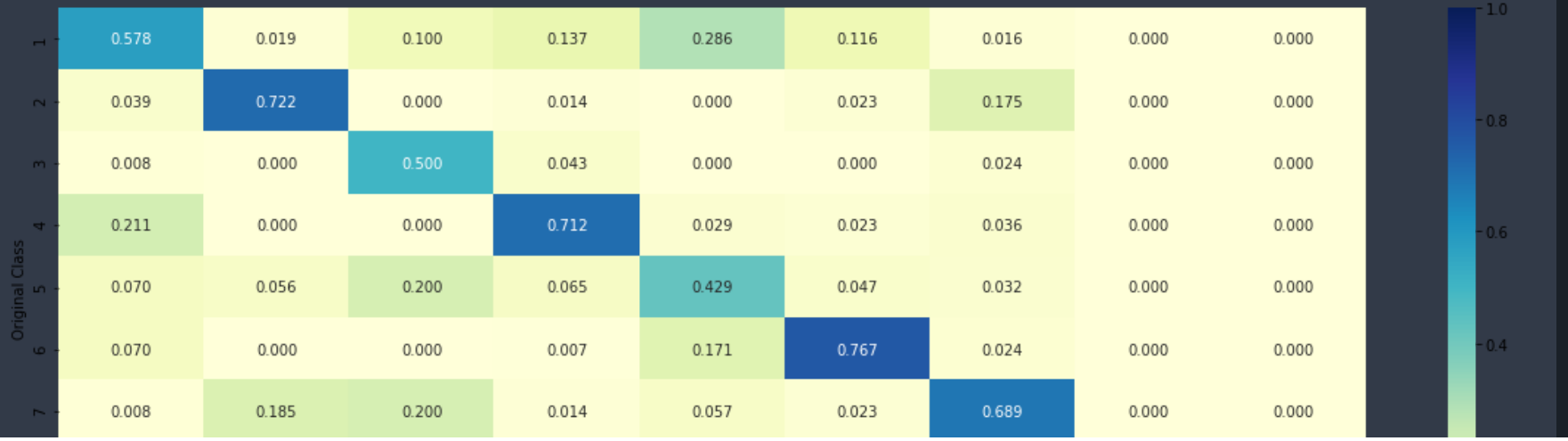

```
For values of best alpha =  0.0001 The train log loss is: 0.3819753740002309
For values of best alpha =  0.0001 The cross validation log loss is: 0.9952544440712352
For values of best alpha =  0.0001 The test log loss is: 0.9620562033780273
```

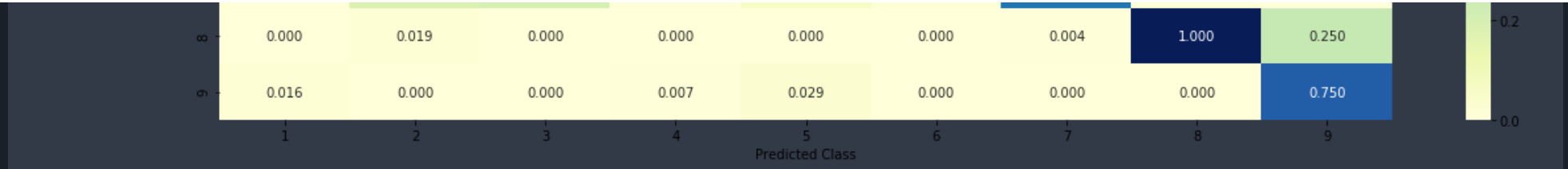
In [118]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

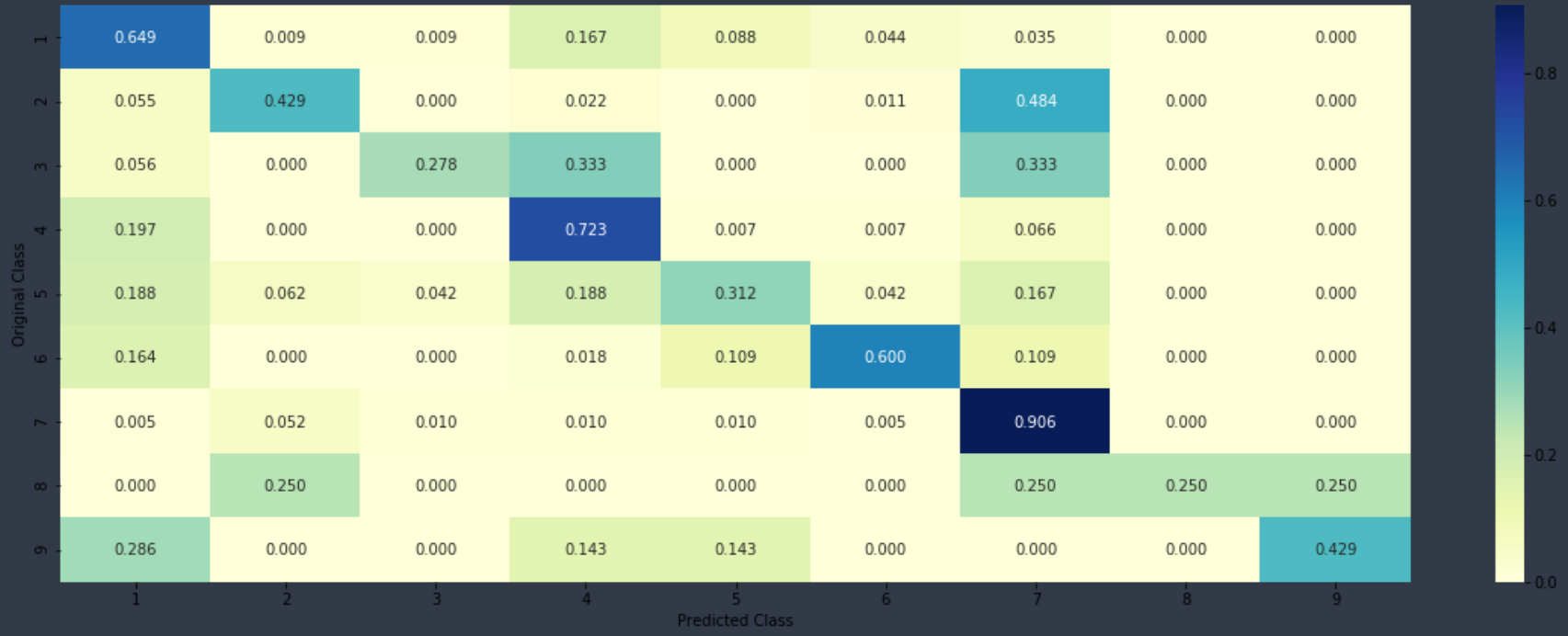
```
Log Loss : 0.96911618942236
Number of missclassified point : 0.33533834586466166
-------------------- Confusion matrix --------------------
```


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






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




## Important features for predicted points

In [119]:
```
def get_impfeature_names(indices, text, gene, var, no_features):
```

```
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(ngram_range=(1,2),min_df=3,max_features=50000)

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

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

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

```
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word
,yes_no))

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

In [120]:

```
test_point_index = 100
no_feature = 1000
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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: [[0.0225 0.2024 0.0072 0.0404 0.0309 0.026  0.6638 0.0049 0.002 ]]
Actual Class : 7
--------------------------------------------------
43 Text feature [20] present in test data point [True]
368 Text feature [10] present in test data point [True]
760 Text feature [activating] present in test data point [True]
875 Text feature [26] present in test data point [True]
897 Text feature [77] present in test data point [True]
Out of the top  1000  features  5 are present in query point
```

# Experimienting some feature engineering

# techniques

## Combining tfidf nad Bow vectorizer with unigram and bigrams and applying balanced Logistic Regression

In [121]:
```
tfidf_text_vectorizer = TfidfVectorizer(min_df=3,max_features=5000)
tfidf_train_text_feature_onehotCoding = tfidf_text_vectorizer.fit_transform(train_df['TE
XT'])
# getting all the feature names (words)
tfidf_train_text_features= tfidf_text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numbe
r of features) vector
tfidf_train_text_fea_counts = tfidf_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 o
ccured
tfidf_text_fea_dict = dict(zip(list(tfidf_train_text_features),tfidf_train_text_fea_coun
ts))


print("Total number of unique words in train data :", len(tfidf_train_text_features))
tfidf_train_text_feature_onehotCoding = normalize(tfidf_train_text_feature_onehotCoding,
 axis=0)

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

```
xis=0)

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

```
Total number of unique words in train data : 5000
```

In [122]:

```
train_x_onehotCoding = hstack((train_x_onehotCoding,tfidf_train_text_feature_onehotCodin
g)).tocsr()
test_x_onehotCoding = hstack((test_x_onehotCoding, tfidf_test_text_feature_onehotCoding
)).tocsr()
cv_x_onehotCoding = hstack((cv_x_onehotCoding, tfidf_cv_text_feature_onehotCoding)).tocs
r()



print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 12198)
(number of data points * number of features) in test data =  (665, 12198)
(number of data points * number of features) in cross validation data = (532, 12198)
```

## Balanced weighted logistic regression

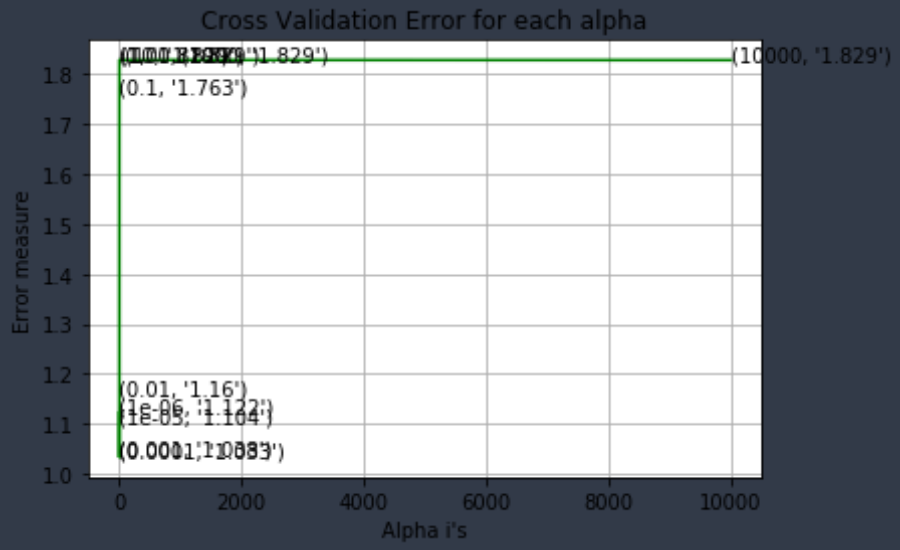
In [124]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=3)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=3)
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_los
s(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 i
s:",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 : 1.121839562289465
for alpha = 1e-05
Log Loss : 1.1042033774785402
for alpha = 0.0001
Log Loss : 1.0327945940252163
for alpha = 0.001
Log Loss : 1.037653683595981
for alpha = 0.01
Log Loss : 1.1601345508262806
for alpha = 0.1
Log Loss : 1.7631068670475298
for alpha = 1
Log Loss : 1.8290688106757766
for alpha = 10
Log Loss : 1.8287634570702795
for alpha = 100
Log Loss : 1.8287813378438855
for alpha = 1000
```

```
Log Loss : 1.8287755223988609
for alpha = 10000
Log Loss : 1.828589511926389
```



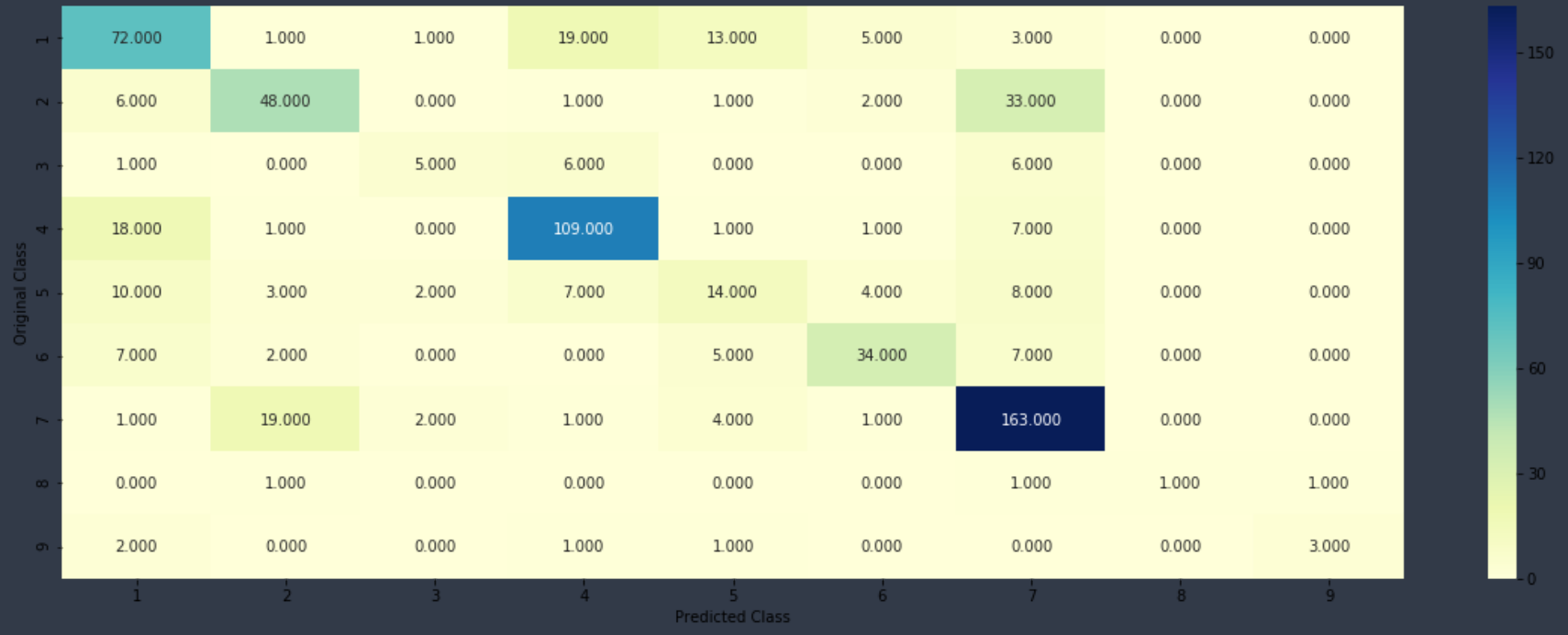

```
For values of best alpha =  0.0001 The train log loss is: 0.42391396443753687
For values of best alpha =  0.0001 The cross validation log loss is: 1.0327945940252163
For values of best alpha =  0.0001 The test log loss is: 0.9761455787348786
```

Here we can see that we are able to reduce the test log_loss to 0.97060 less than 1

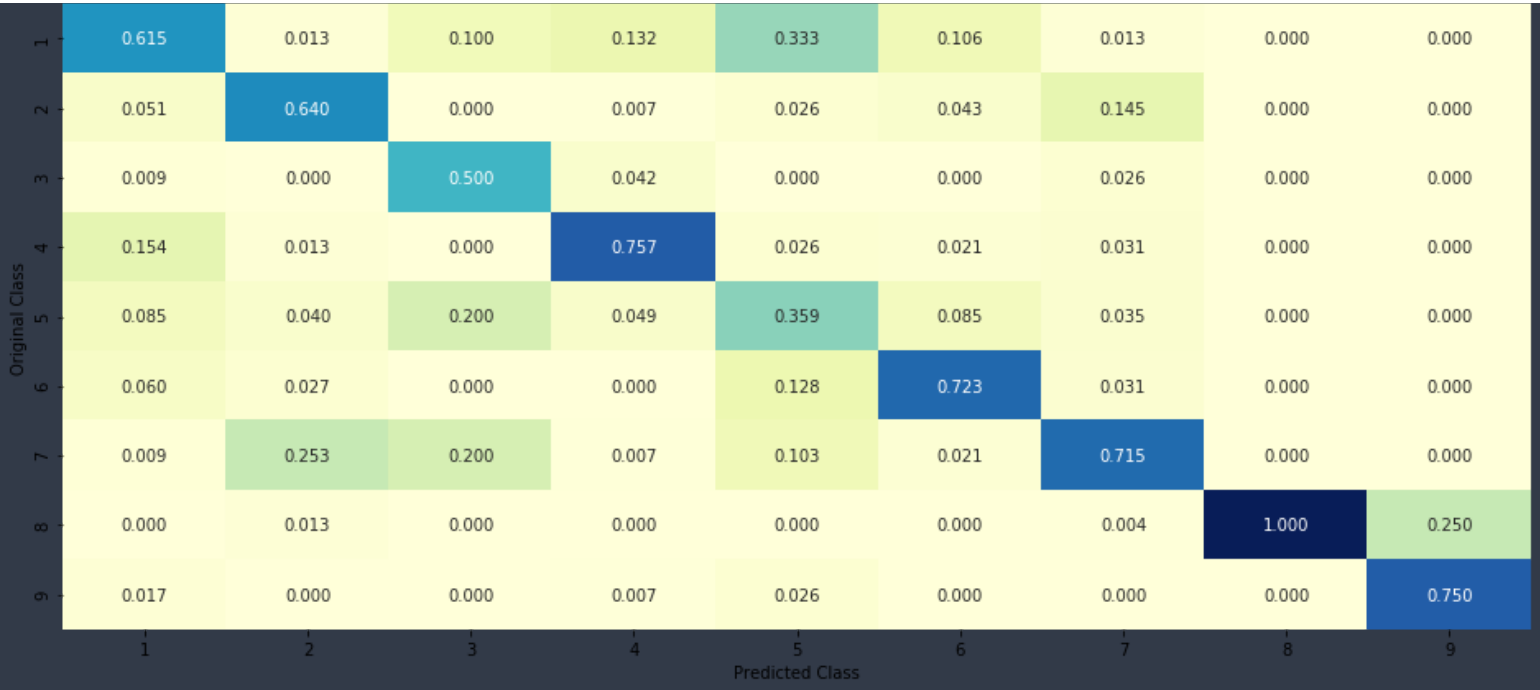
In [126]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
```

```
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

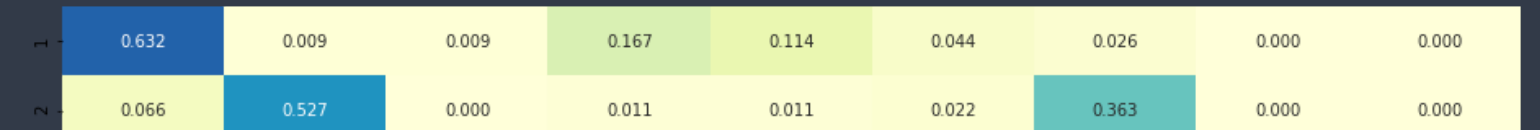
```
Log Loss : 0.9761455787348786
Number of missclassified point : 0.324812030075188
-------------------- Confusion matrix --------------------
```




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




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




## Logistic regression

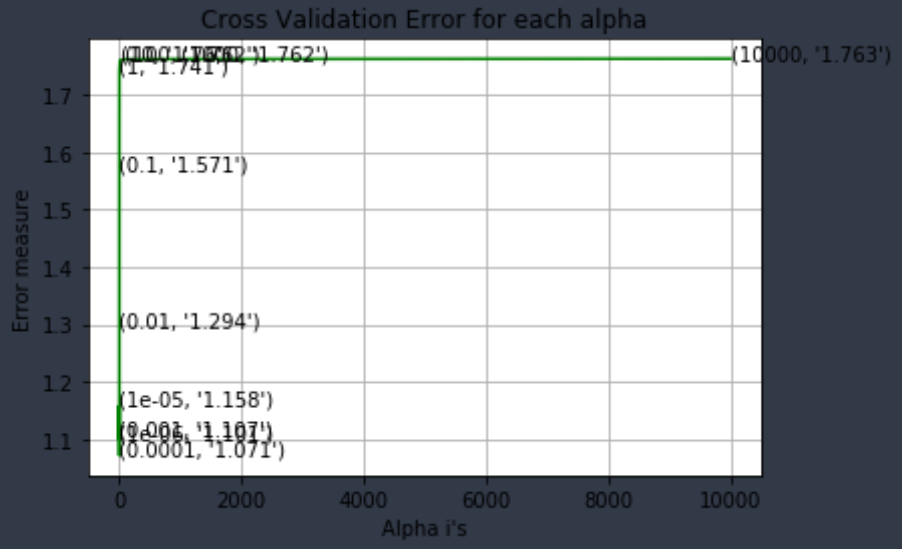
In [127]:

```
alpha = [10**i for i in range(-6,5) ]
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",cv=3)
    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 esti
mates
    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",cv=3)
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_los
s(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 i
s:",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 : 1.1008164335707604
for alpha = 1e-05
```

```
Log Loss : 1.1580867498167335
for alpha = 0.0001
Log Loss : 1.0712442629652899
for alpha = 0.001
Log Loss : 1.1071006287324454
for alpha = 0.01
Log Loss : 1.2944172441033663
for alpha = 0.1
Log Loss : 1.5706471387970384
for alpha = 1
Log Loss : 1.7406387592017136
for alpha = 10
Log Loss : 1.7600471024600775
for alpha = 100
Log Loss : 1.7620601695178861
for alpha = 1000
Log Loss : 1.7623267365459612
for alpha = 10000
Log Loss : 1.7628525868167413
```



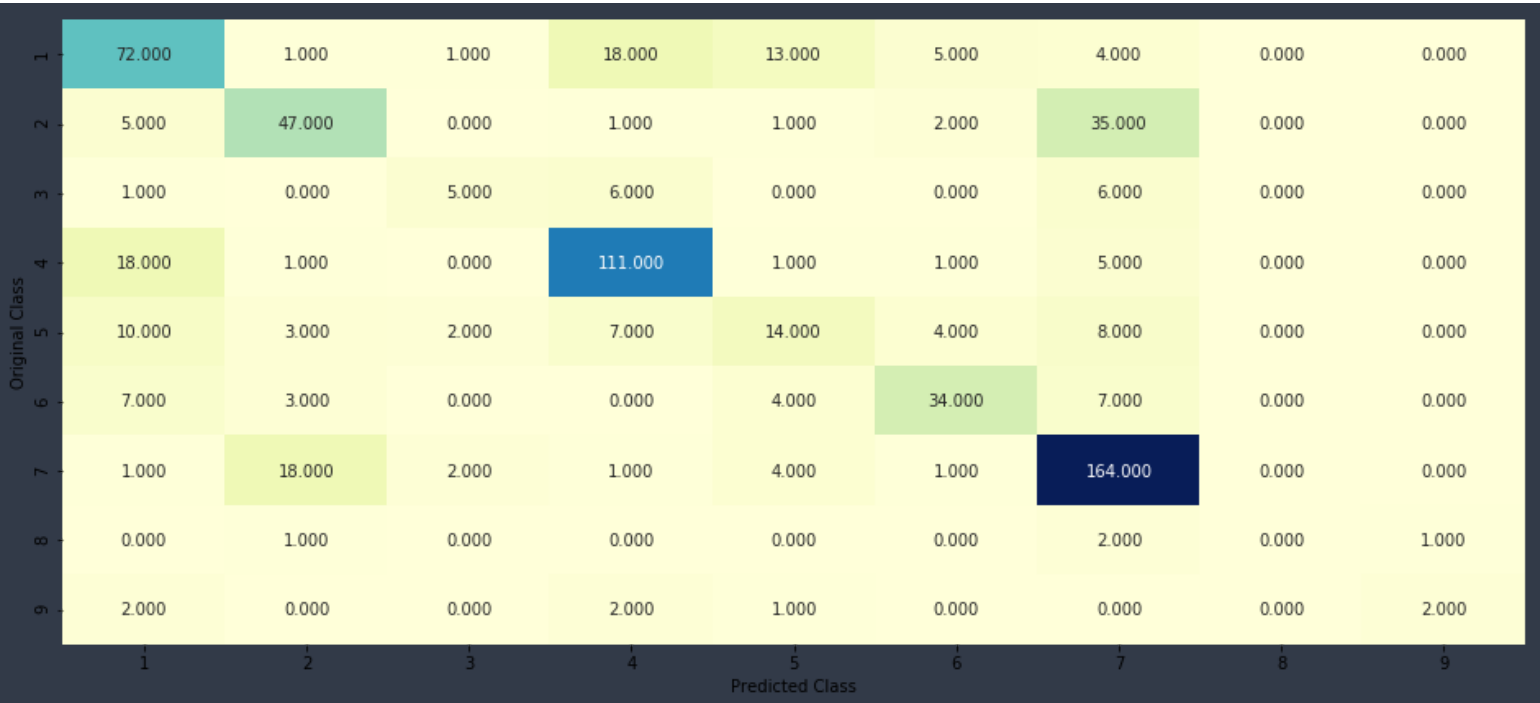

```
For values of best alpha =  0.0001 The train log loss is: 0.4206888722913239
```

```
For values of best alpha =  0.0001 The cross validation log loss is: 1.0712442629652899
For values of best alpha =  0.0001 The test log loss is: 0.9914091228698361
```

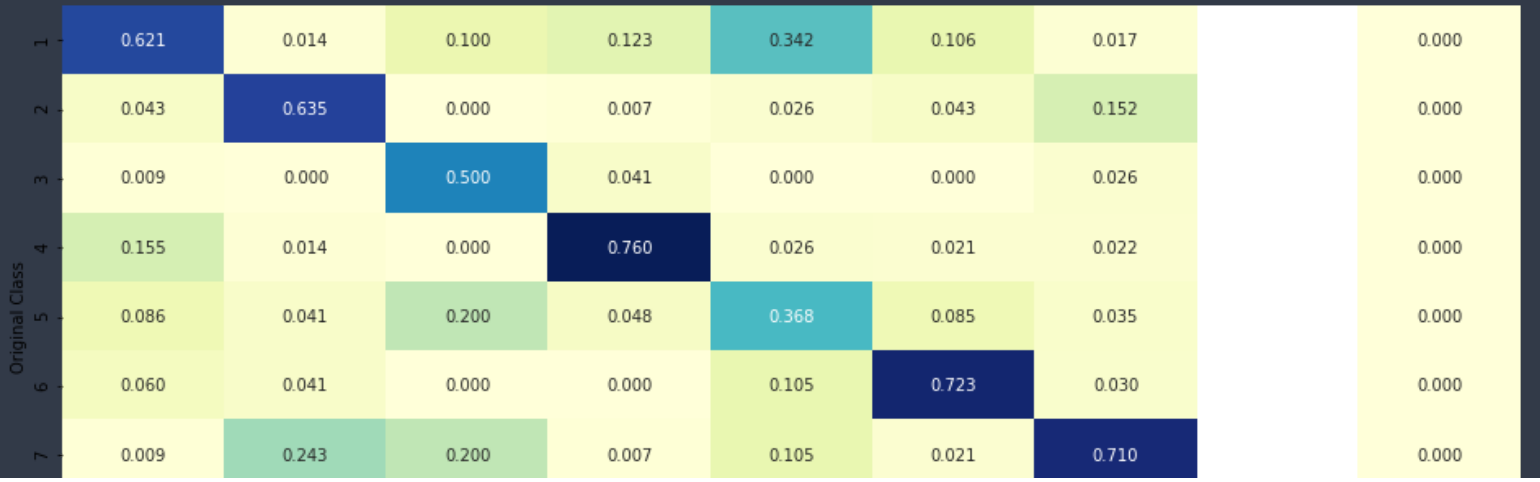
In [128]:

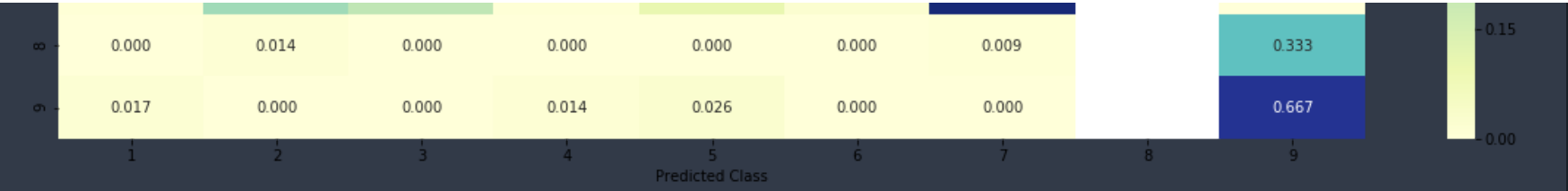
```
clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

```
Log Loss : 0.9914091228698361
Number of missclassified point : 0.324812030075188
-------------------- Confusion matrix --------------------
```

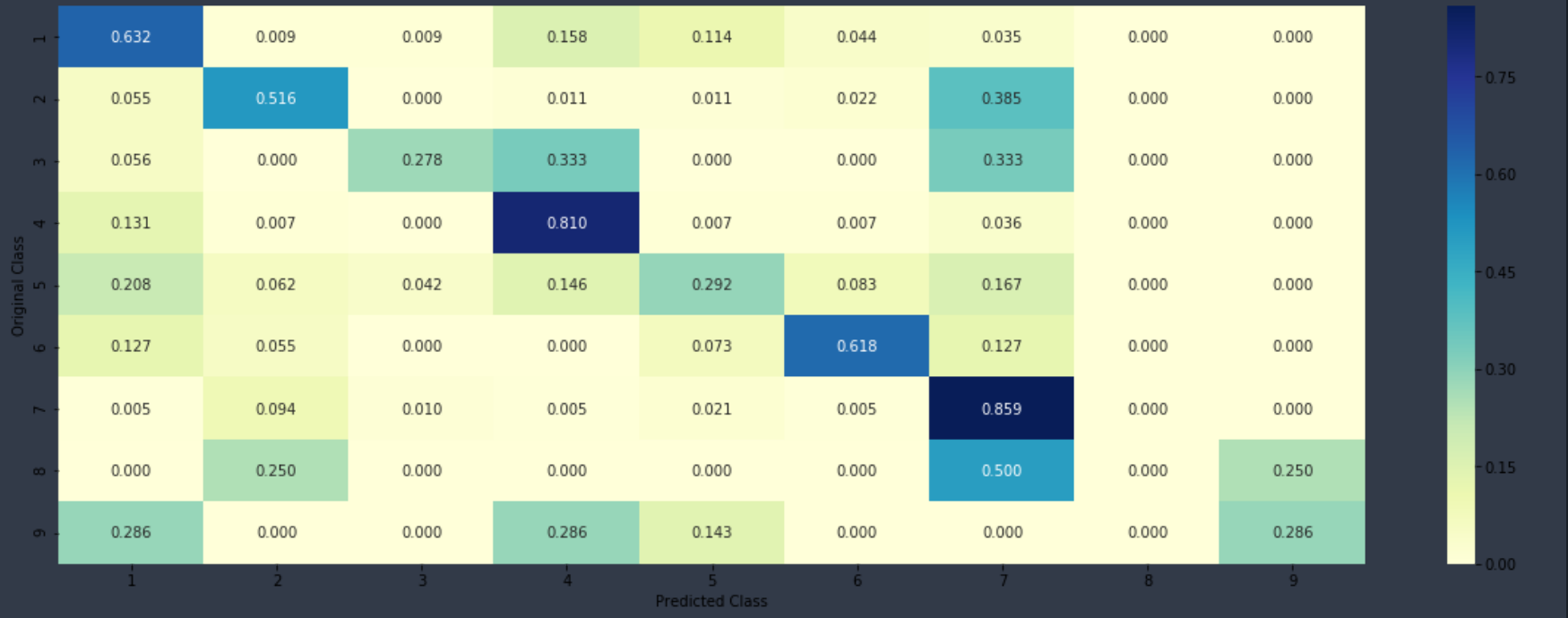



---------- Precision matrix (Columm Sum=1) ----------






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




## Stacking classifier

In [129]:
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', ran
dom_state=42)
```

```
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid",cv=None)

clf2 = SGDClassifier(alpha=0.001, penalty='l2', loss='hinge', class_weight='balanced', r
andom_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid",cv=None)


clf3 = SGDClassifier(alpha=0.001,penalty='l2',loss='log',random_state=42)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(
cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pro
ba(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_oneh
otCoding))))
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_classifie
r=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
```

```
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_lo
ss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

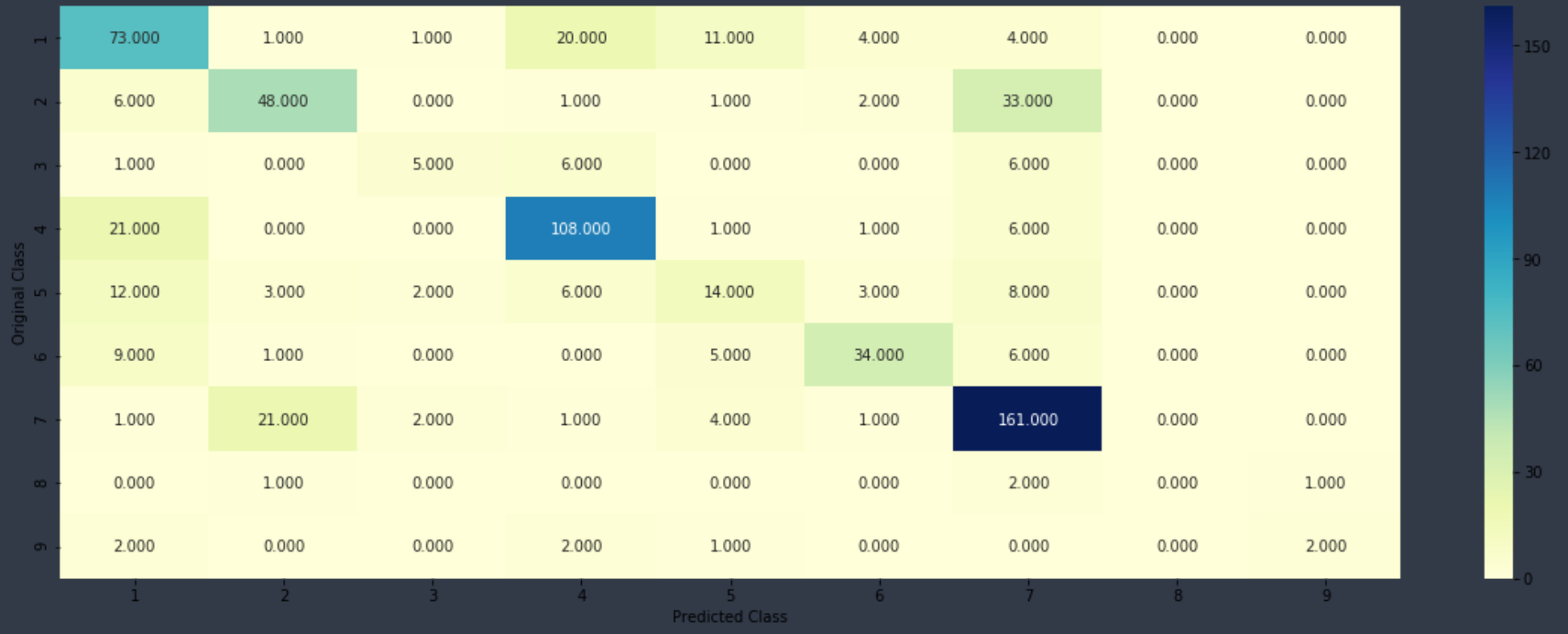
```
Logistic Regression :  Log Loss: 1.04
Support vector machines : Log Loss: 1.03
Naive Bayes : Log Loss: 1.11
-----------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.169
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.956
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.321
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.061
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.212
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.490
```

In [130]:

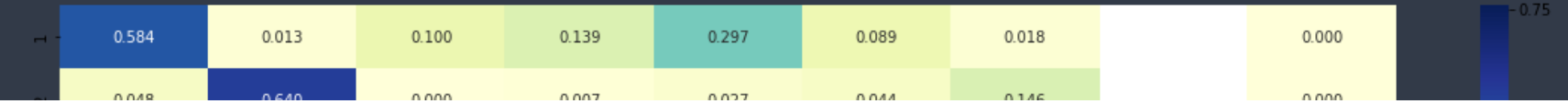
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr
, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error = log_loss(test_y, sclf.predict_proba(test_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_onehotCo
```

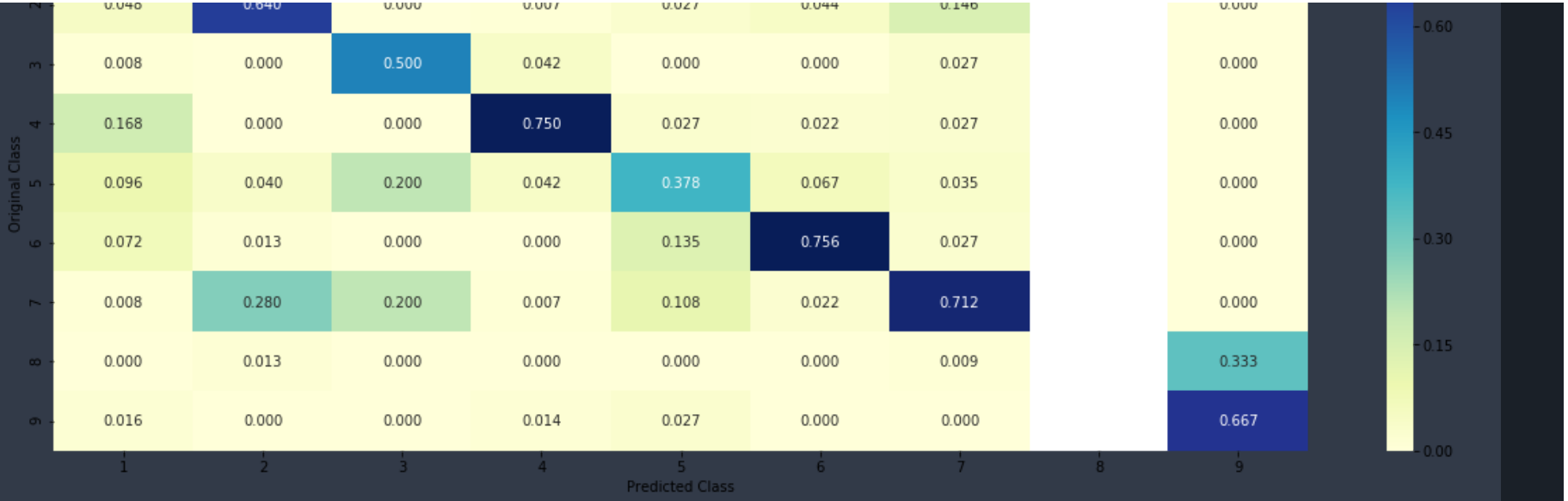
```
ding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 1.0405261486824826
Log loss (CV) on the stacking classifier : 1.0405261486824826
Log loss (test) on the stacking classifier : 1.0405261486824826
Number of missclassified point : 0.3308270676691729
-------------------- Confusion matrix --------------------
```

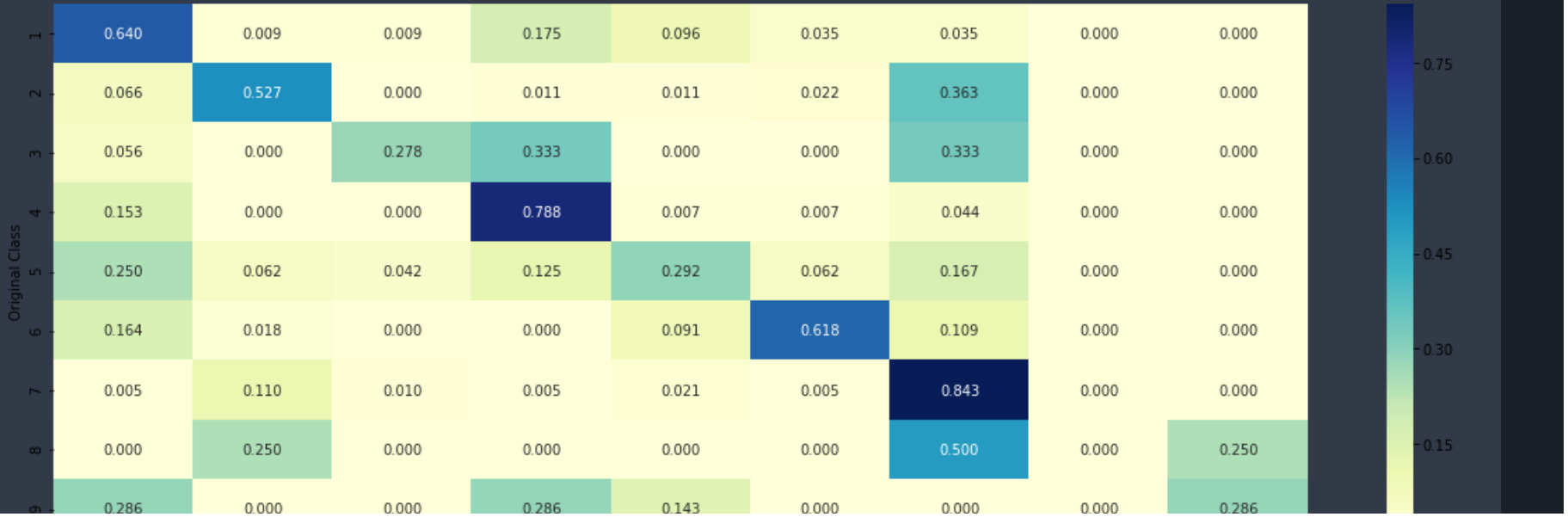



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





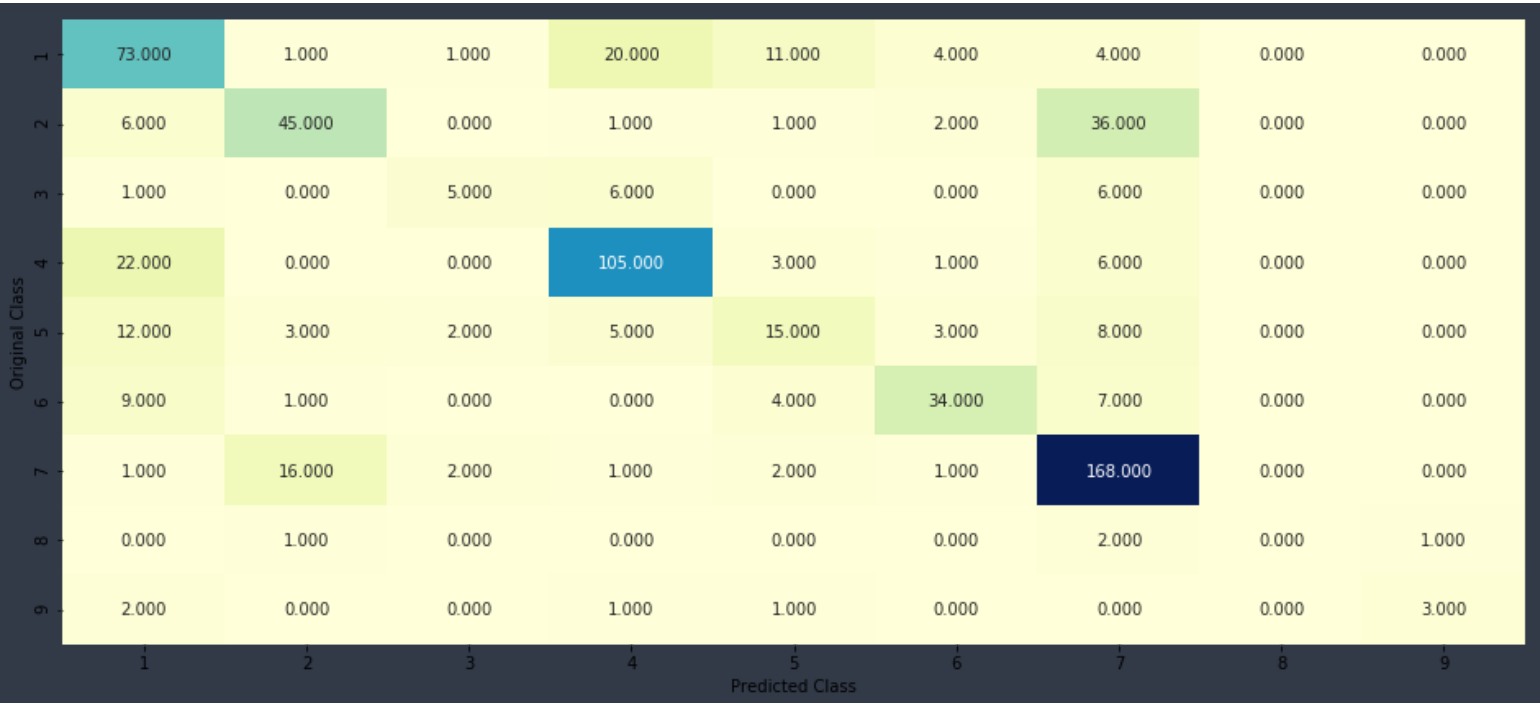

---------- Recall matrix (Row sum=1) ----------



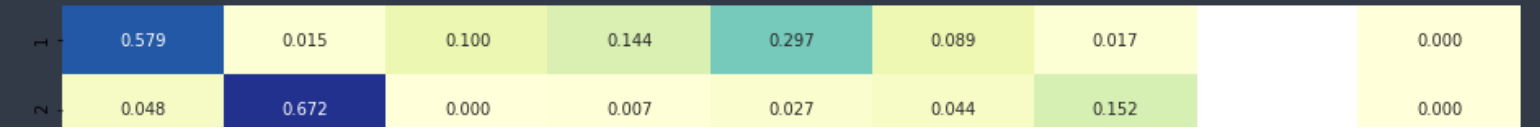

## Voting Classifier

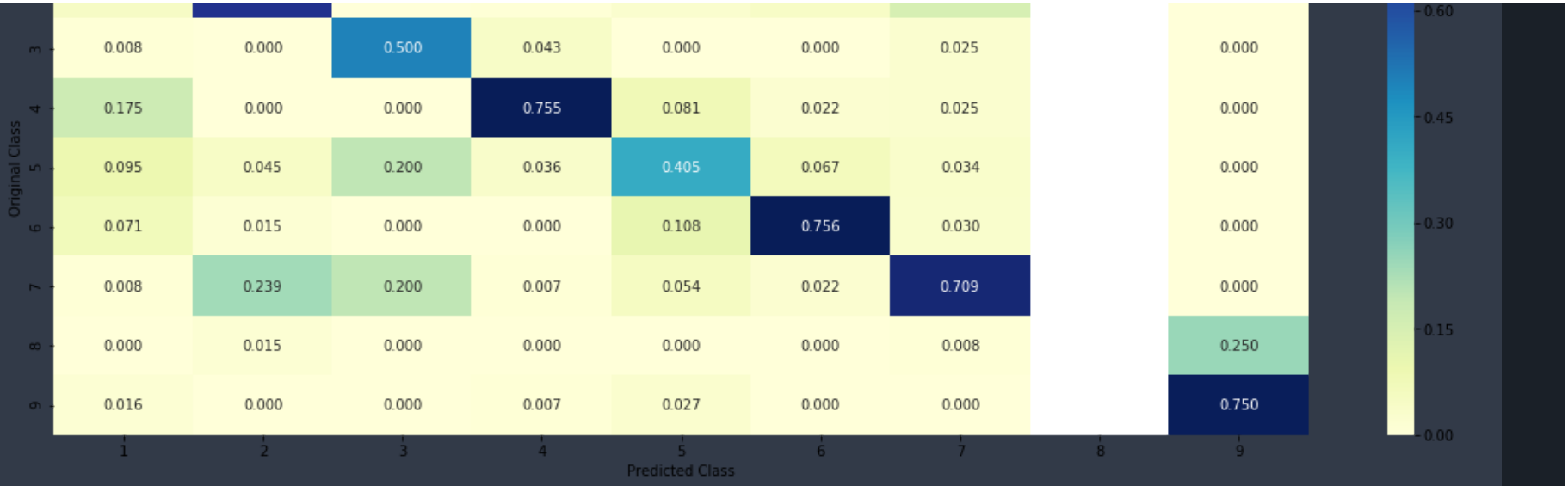
In [131]:
```
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3
)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba
(train_x_onehotCoding)))
print("Log loss (CV) on the 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(t
est_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCo
ding)- 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 : 0.5157624808203557
Log loss (CV) on the VotingClassifier : 1.0001106427460265
Log loss (test) on the VotingClassifier : 0.9768831130105323
Number of missclassified point : 0.3263157894736842
-------------------- Confusion matrix --------------------
```

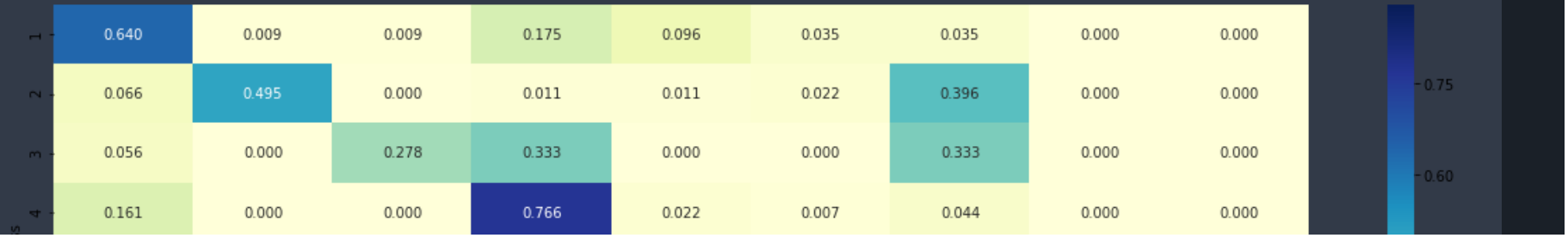


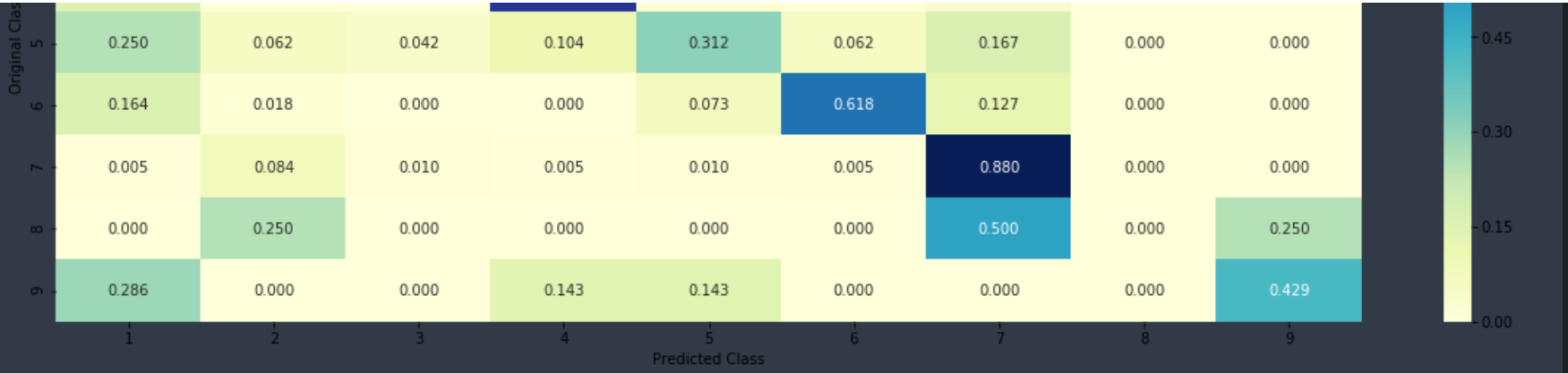

---------- Precision matrix (Columm Sum=1) ----------






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






In [132]:

```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', ran
dom_state=42)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid",cv=None)

clf2 = SGDClassifier(alpha=0.001, penalty='l2', loss='hinge', class_weight='balanced', r
andom_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid",cv=None)


clf3 = SGDClassifier(alpha=0.001,penalty='l2',loss='log',random_state=42)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)

clf4 = KNeighborsClassifier(n_neighbors=5)
clf4.fit(train_x_onehotCoding, train_y)
sig_clf4 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)
```

```
sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(
cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pro
ba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("logistic unbalanced : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(c
v_x_onehotCoding))))
sig_clf4.fit(train_x_onehotCoding, train_y)
print("KNN : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf4.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,sig_clf4], meta_
classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_lo
ss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

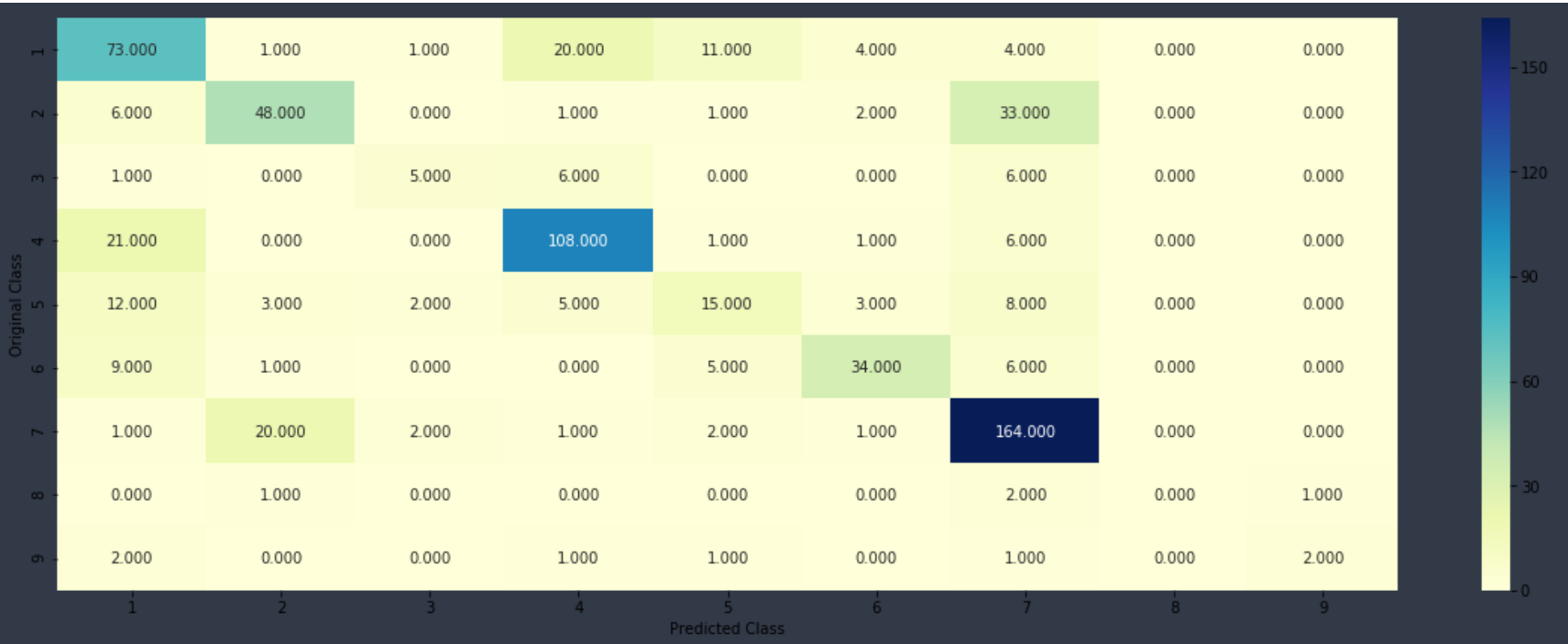
```
Logistic Regression :  Log Loss: 1.04
Support vector machines : Log Loss: 1.03
logistic unbalanced : Log Loss: 1.11
KNN : Log Loss: 1.11
```

```
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.162
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.908
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.267
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.070
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.223
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.494
```

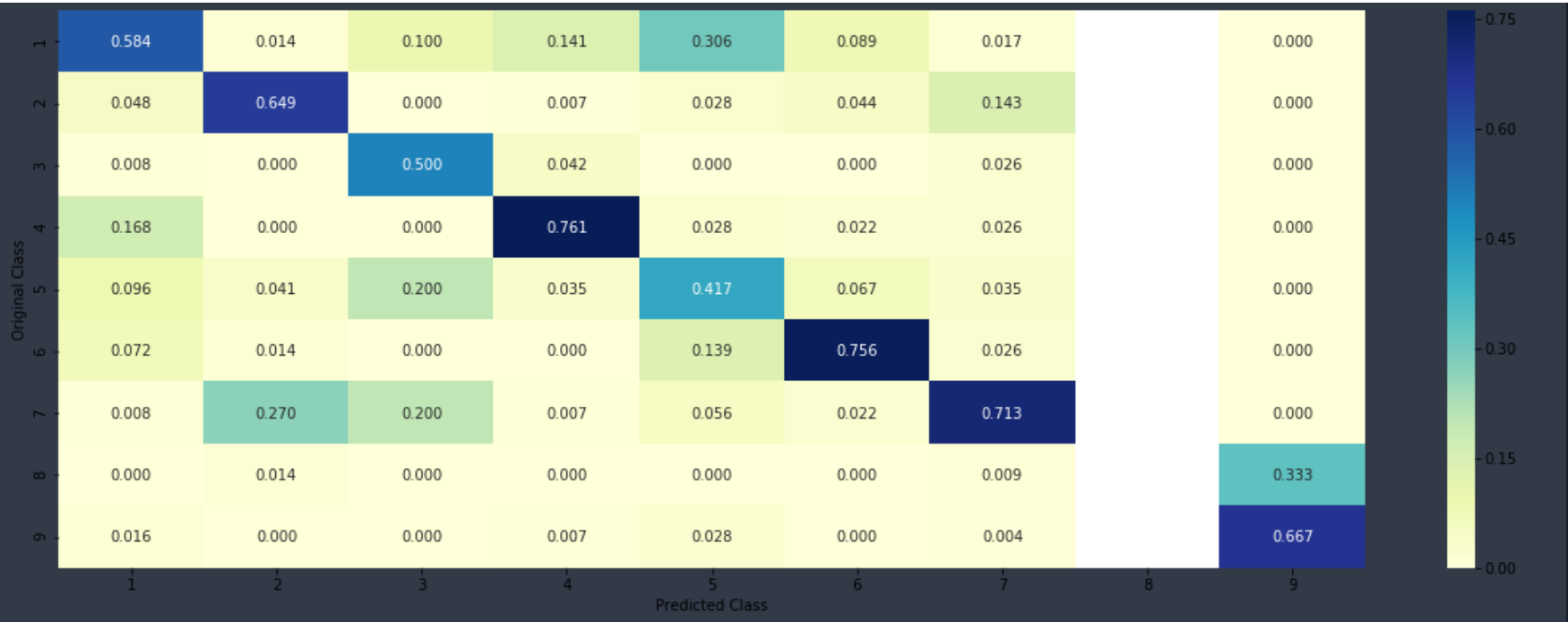
In [133]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3, sig_clf4], meta_cla
ssifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error = log_loss(test_y, sclf.predict_proba(test_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_onehotCo
ding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

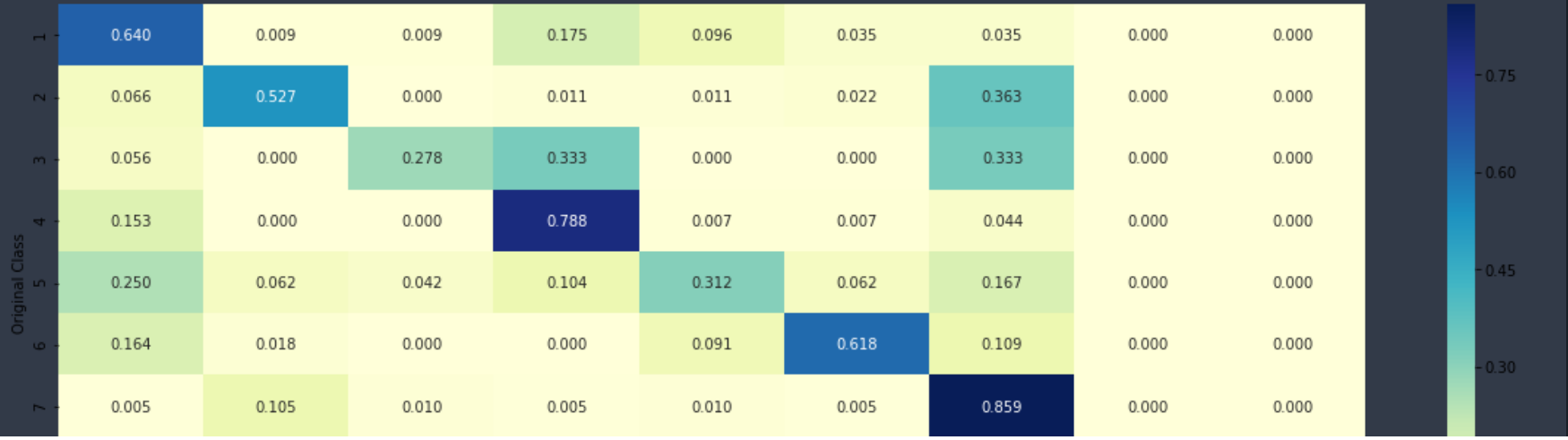
```
Log loss (train) on the stacking classifier : 1.0439243683627082
Log loss (CV) on the stacking classifier : 1.0439243683627082
Log loss (test) on the stacking classifier : 1.0439243683627082
Number of missclassified point : 0.324812030075188
------------------- Confusion matrix -------------------
```

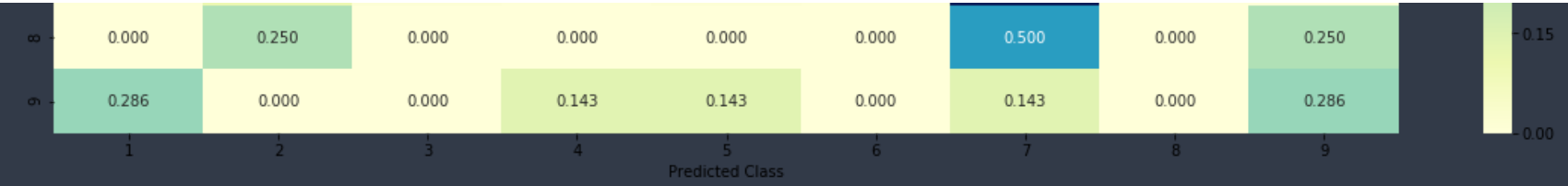



---------- Precision matrix (Columm Sum=1) ----------




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

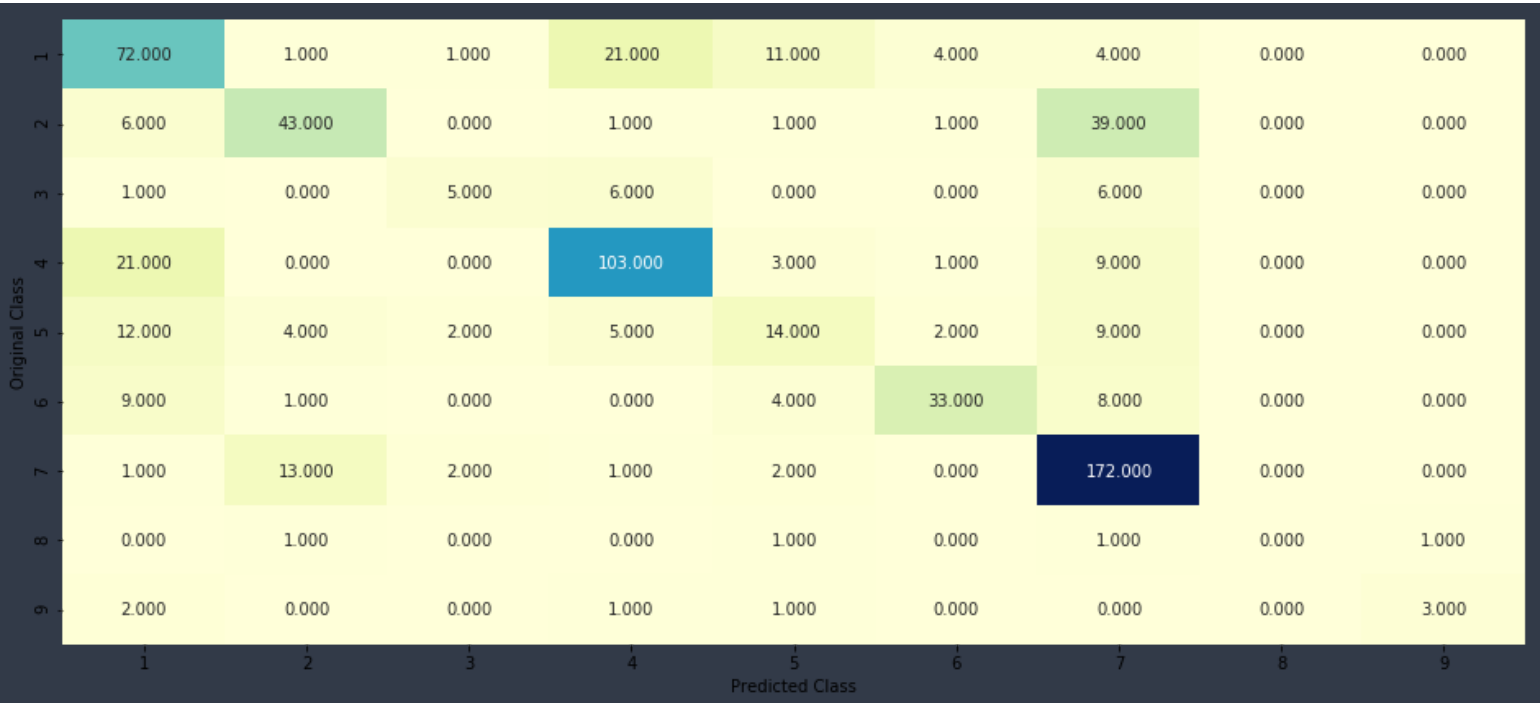




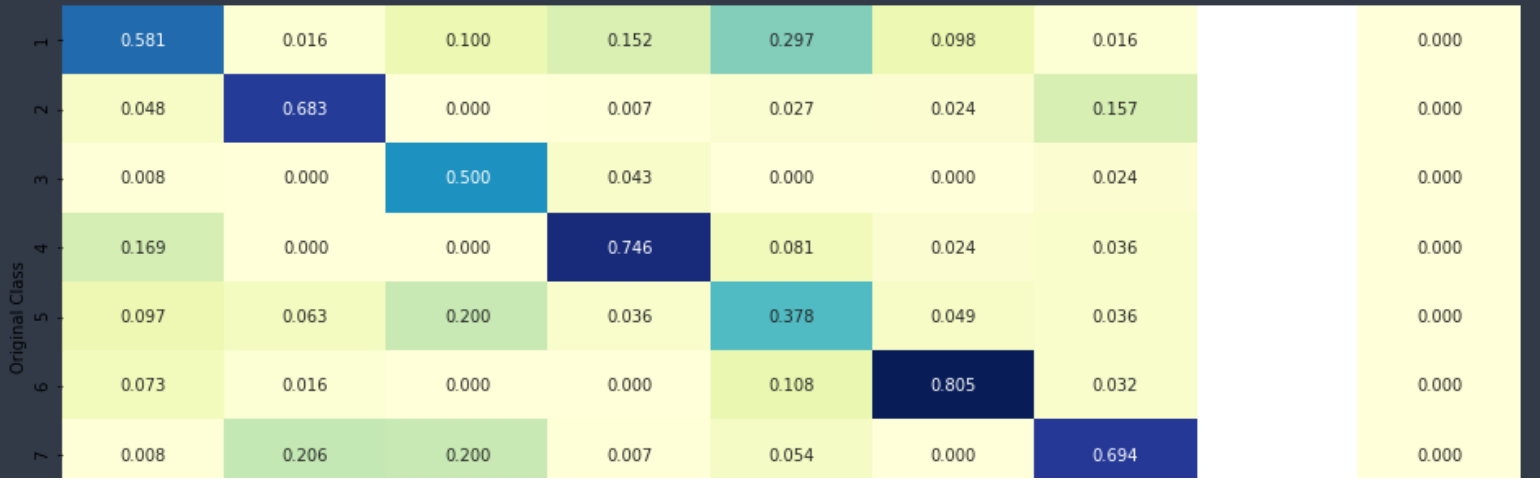

In [134]:

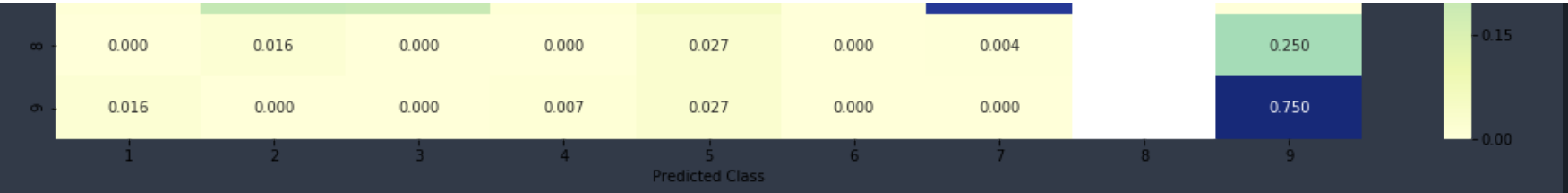
```
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('lru', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba
(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_
onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(t
est_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCo
ding)- 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 : 0.5586887563748579
Log loss (CV) on the VotingClassifier : 1.0511960327464183
Log loss (test) on the VotingClassifier : 0.9924148692808032
Number of missclassified point : 0.3308270676691729
------------------- Confusion matrix --------------------
```

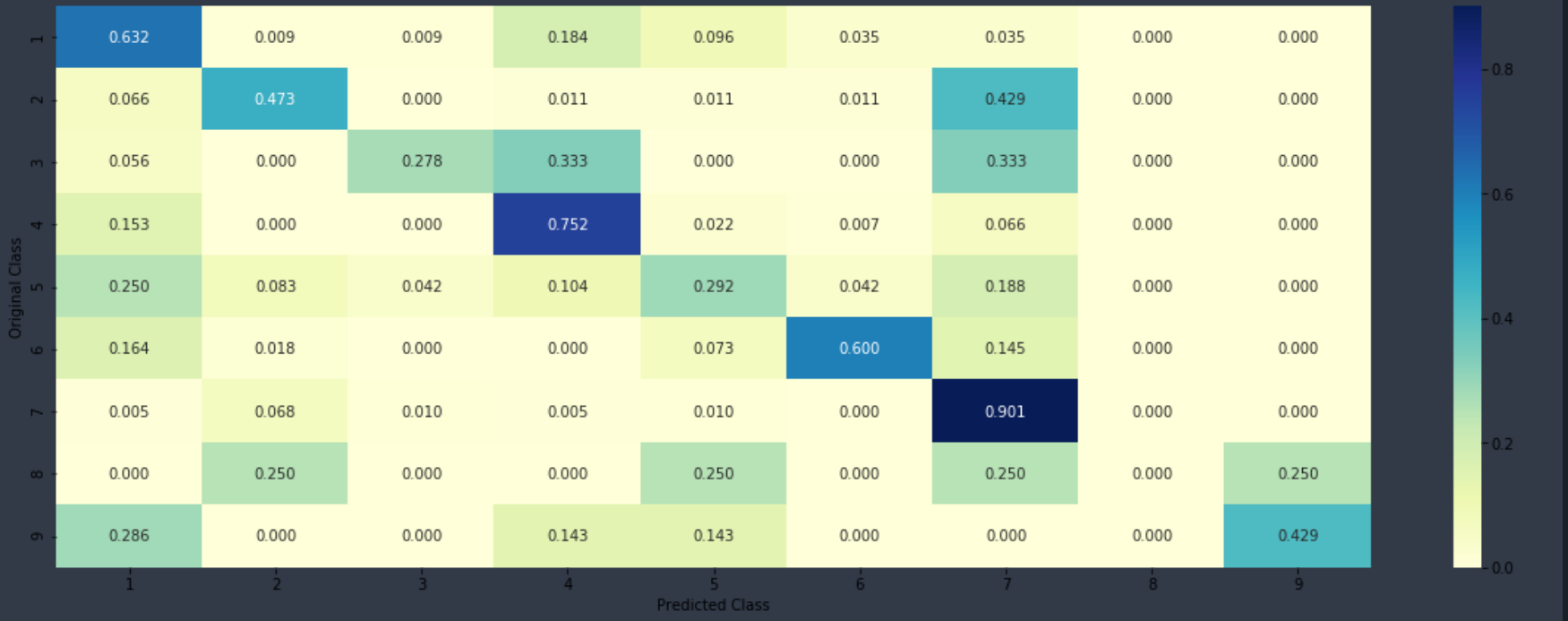



---------- Precision matrix (Columm Sum=1) ----------






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



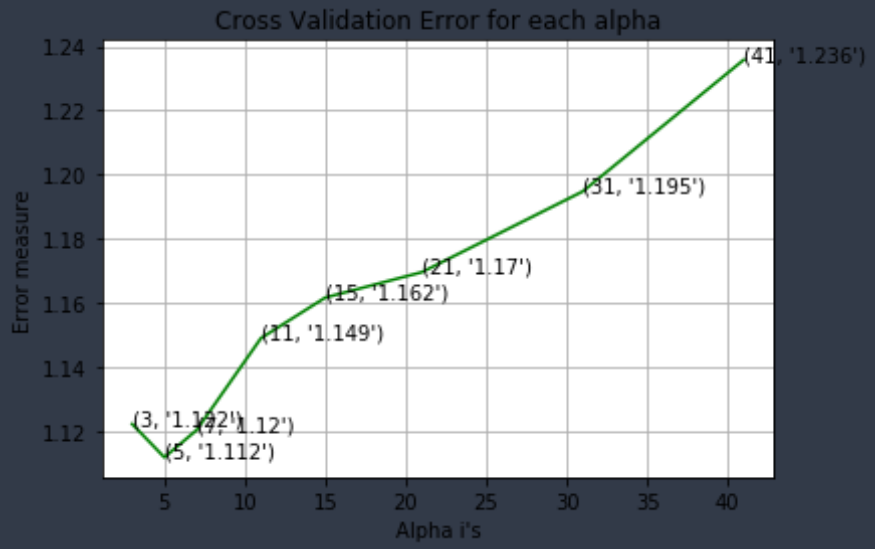

# KNN

In [135]:
```
alpha = [3,5, 7,11, 15, 21, 31, 41]
cv_log_error_array = []
```

```
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",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 = 3
Log Loss : 1.1222121037252772
for alpha = 5
Log Loss : 1.1118392249284594
for alpha = 7
Log Loss : 1.1202972728734302
for alpha = 11
Log Loss : 1.1489749989773221
for alpha = 15
Log Loss : 1.1616518832422182
for alpha = 21
Log Loss : 1.1695728032692823
for alpha = 31
Log Loss : 1.1947610562590767
for alpha = 41
Log Loss : 1.235798570140746
```




```
For values of best alpha =  5 The train log loss is: 0.9012839988292586
For values of best alpha =  5 The cross validation log loss is: 1.1118392249284594
For values of best alpha =  5 The test log loss is: 1.0959923558537161
```

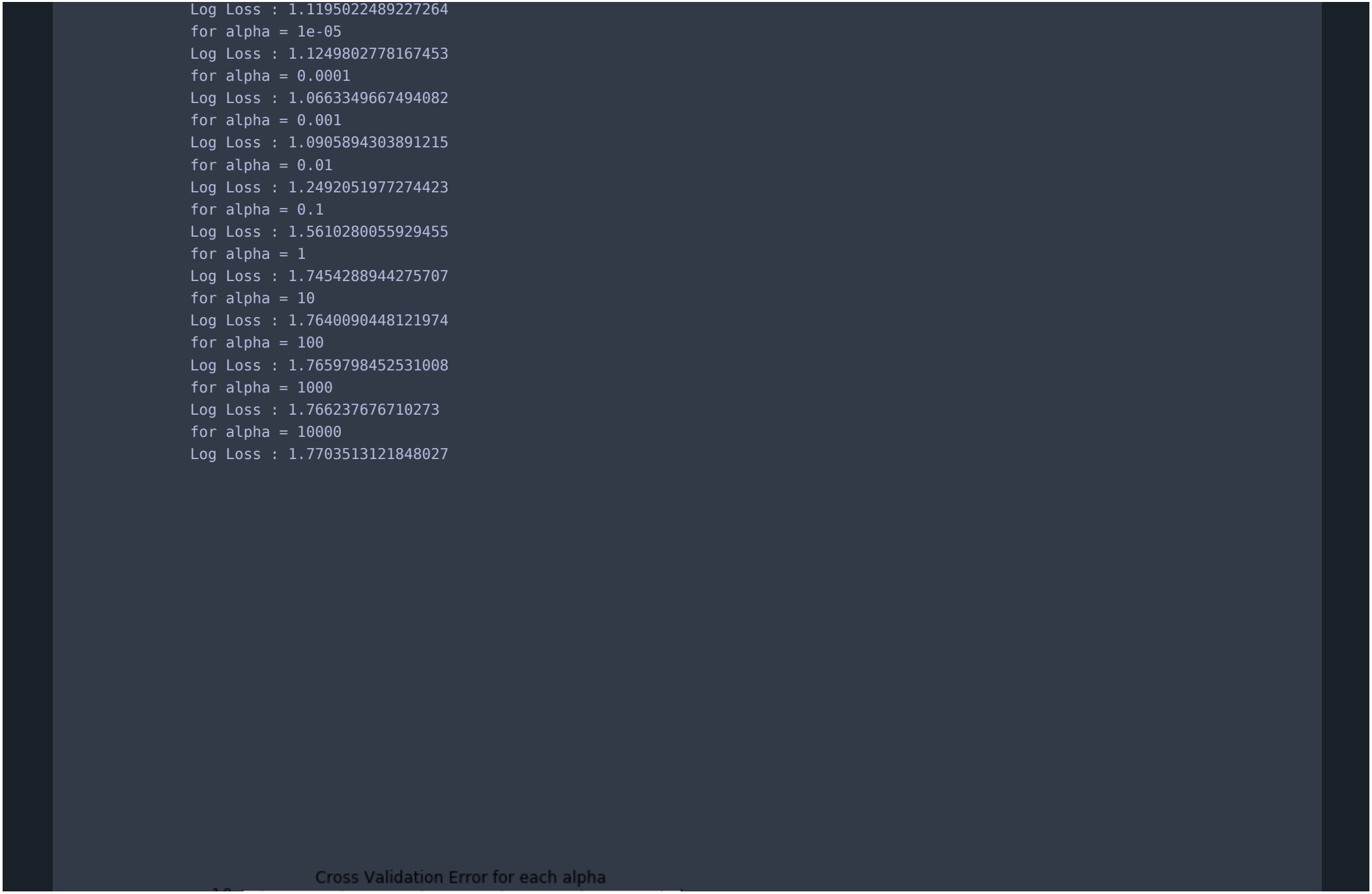
In [136]:

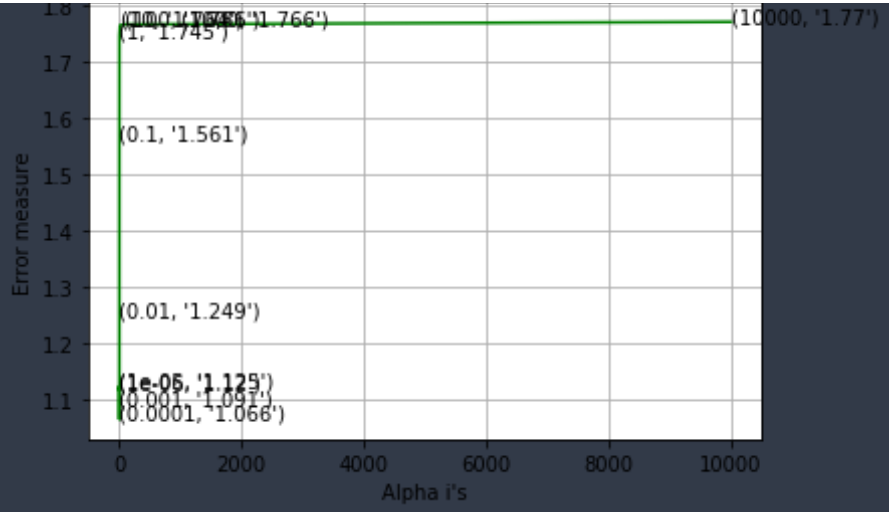
```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier( alpha=i, penalty='l2', loss='log', random_state=0)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    sig_clf.fit(train_x_onehotCoding, 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 esti
```

```
mates
    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=0)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",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 : 1.1195022489227264
for alpha = 1e-05
Log Loss : 1.1249802778167453
for alpha = 0.0001
Log Loss : 1.0663349667494082
for alpha = 0.001
Log Loss : 1.0905894303891215
for alpha = 0.01
Log Loss : 1.2492051977274423
for alpha = 0.1
Log Loss : 1.5610280055929455
for alpha = 1
Log Loss : 1.7454288944275707
for alpha = 10
Log Loss : 1.7640090448121974
for alpha = 100
Log Loss : 1.7659798452531008
for alpha = 1000
Log Loss : 1.766237676710273
for alpha = 10000
Log Loss : 1.7703513121848027
```






```
For values of best alpha =  0.0001 The train log loss is: 0.36207140171733054
For values of best alpha =  0.0001 The cross validation log loss is: 1.0663349667494082
For values of best alpha =  0.0001 The test log loss is: 0.9954403958998986
```

# [Conclusion]

In [6]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","Train log_loss","Test log_loss","CV log_loss" ]
x.add_row(["Logistic regression on count vectorizer unigrams and bigrams",0.381,0.995,0.
962])
x.add_row(["Balanced Logistic Regression on FE points",0.423,1.032,0.976])
x.add_row(["Logistic Regression on FE points",0.420,1.071,0.991])
x.add_row(["Stacking Classifier 1 on FE points",1.04,1.04,1.04])
x.add_row(["Stacking Classifier 2 on FE points",1.04,1.04,1.04])
x.add_row(["Voting Classifier on FE points",0.51,1.000,0.97])
```

```
x.add_row(["KNN on FE points",0.362,1.066,0.995])
print(x)
```

```
+-------------------------------------------------------------+---------------+--------------+-------------+
|                            Model                            | Train log_loss | Test log_loss | CV log_loss |
+-------------------------------------------------------------+---------------+--------------+-------------+
| Logistic regression on count vectorizer unigrams and bigrams |     0.381     |     0.995    |    0.962    |
|          Balanced Logistic Regression on FE points          |     0.423     |     1.032    |    0.976    |
|               Logistic Regression on FE points              |      0.42     |     1.071    |    0.991    |
|              Stacking Classifier 1 on FE points             |      1.04     |      1.04    |     1.04    |
|              Stacking Classifier 2 on FE points             |      1.04     |      1.04    |     1.04    |
|                Voting Classifier on FE points               |      0.51     |      1.0     |     0.97    |
|                       KNN on FE points                      |     0.362     |     1.066    |    0.995    |
+-------------------------------------------------------------+---------------+--------------+-------------+
```

**Here we can observe from above table that logistic regression with count vectorizer inclusing unigrams and bigrams both have test los_loss as 0.995 and CV log_loss as 0.962 which is below 1**