Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [ ]:

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

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 [ ]: # note the seprator in this file
        data_text =pd.read_csv("../input/training_text/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()
```

3.1.3. Preprocessing of text

In [ ]:

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

In [ ]:

```
# 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 [ ]:

```
#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")
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

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

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 [ ]:

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

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

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

In [ ]:

```python
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])
```

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

In [ ]:

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

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

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


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

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

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

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

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 [ ]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column
    
    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]
    
    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1
    
    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    
    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    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 [ ]:

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

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


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

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

3.3 Univariate Analysis

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.2
5, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.0
61224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818181818181
8177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782,
 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.
075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.0
66225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.
093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [ ]:
```
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))
```

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

In [ ]:
```
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 [ ]:
```
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 [ ]:

```
#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 [ ]:

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

In [ ]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(max_features =1000)
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 [ ]:

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

In [ ]:

```
gene_vectorizer.get_feature_names()
```

In [ ]:

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

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 [ ]:

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

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

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

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


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

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

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

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

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 [ ]:

```
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])*1
00)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*1
00)
```

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 [ ]:
```
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))
```

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

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

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 [ ]:

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

In [ ]:

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

In [ ]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(max_features=1000)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [ ]:

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

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

Let's build a model just like the earlier!

In [ ]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=’optimal’, eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

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


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

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

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 [ ]:

```
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_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)
```

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 [ ]:

```
# 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 [ ]:

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

Assignment section 1 and 2: Use TFIDF vectorizer and top 1000 features in TFIDF vectorizer

In [ ]:

```
# building a TFIDF vectorizer with all the words that occured minimum 3 times in train data and also with top
1000 features from it
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

In [ ]:

```
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 [ ]:

```
#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 [ ]:

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

In [ ]:

```
# 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 [ ]:

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

In [ ]:

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

In [ ]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

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

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

Ans. Yes, it seems like!

In [ ]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [ ]:

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

4. Machine Learning Models

In [ ]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [ ]:

```
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 [ ]:

```
# 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 = TfidfVectorizer(max_features=1000)
    var_count_vec = TfidfVectorizer(max_features=1000)
    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 query point")
```

Stacking the three types of features

In [ ]:

```
# merging gene, variance and text features

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

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

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).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)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_response
Coding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCod
ing))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

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

In [ ]:

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

In [ ]:

```
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)
```

4.1. Base Line Model

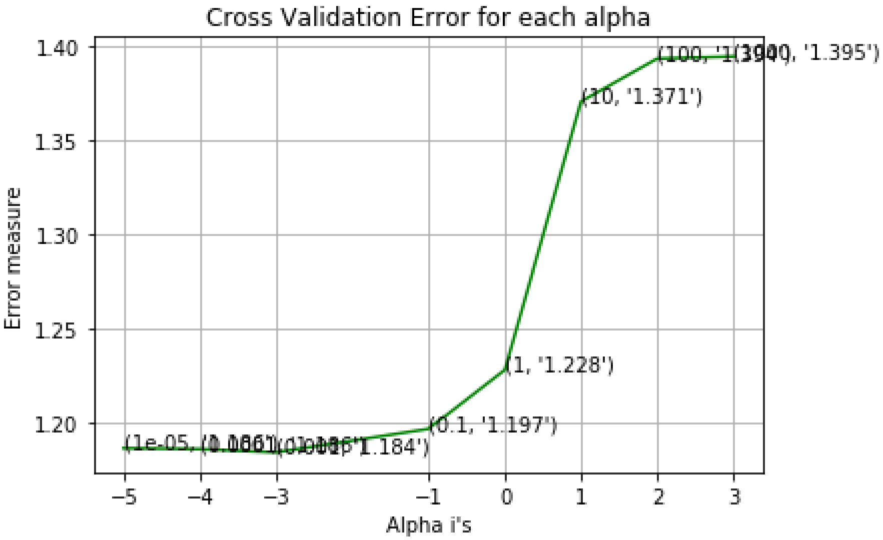
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [134]:

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


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.18629100910839
for alpha = 0.0001
Log Loss : 1.1858194268809341
for alpha = 0.001
Log Loss : 1.184061004326612
for alpha = 0.1
Log Loss : 1.1965477583047557
for alpha = 1
Log Loss : 1.2280413735490678
for alpha = 10
Log Loss : 1.370829337226954
for alpha = 100
Log Loss : 1.3936908279728344
for alpha = 1000
Log Loss : 1.394841278134568
```



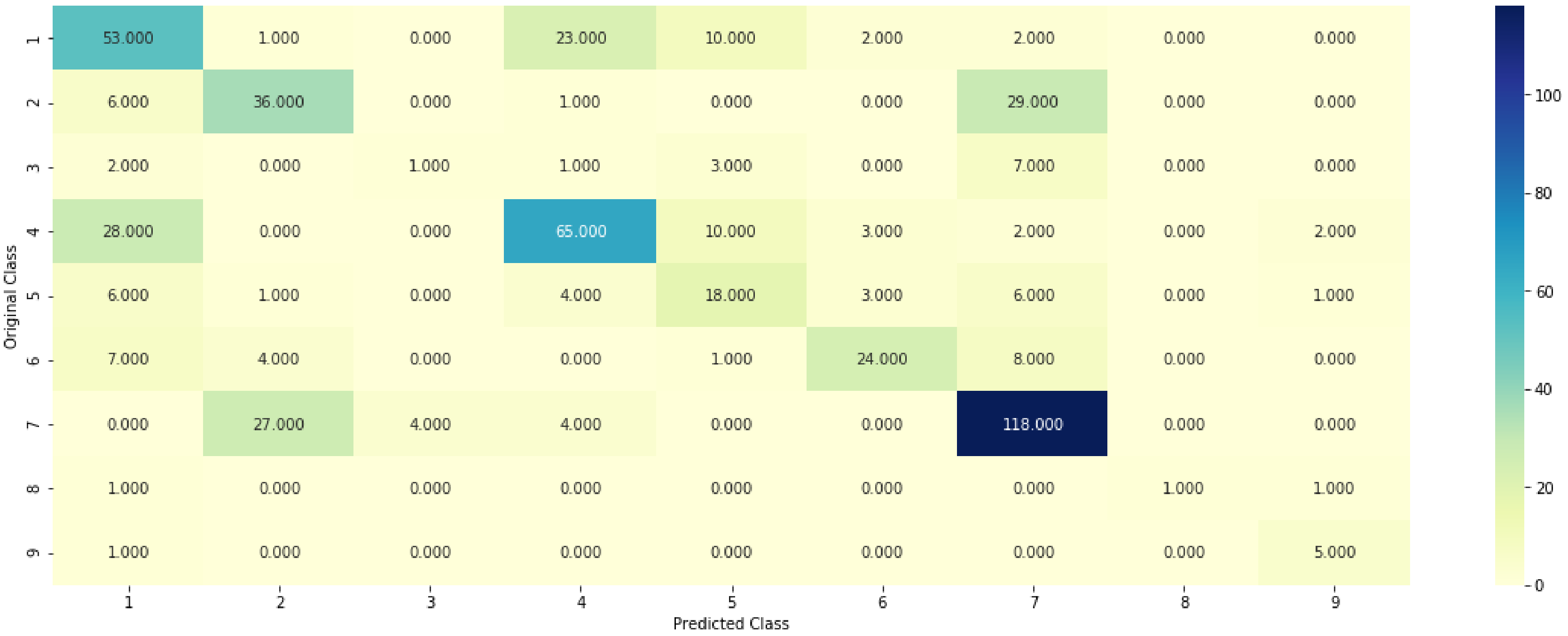
```
For values of best alpha =  0.001 The train log loss is: 0.7489017480903717
For values of best alpha =  0.001 The cross validation log loss is: 1.184061004326612
For values of best alpha =  0.001 The test log loss is: 1.181998257591528
```

4.1.1.2. Testing the model with best hyper paramters

In [135]:

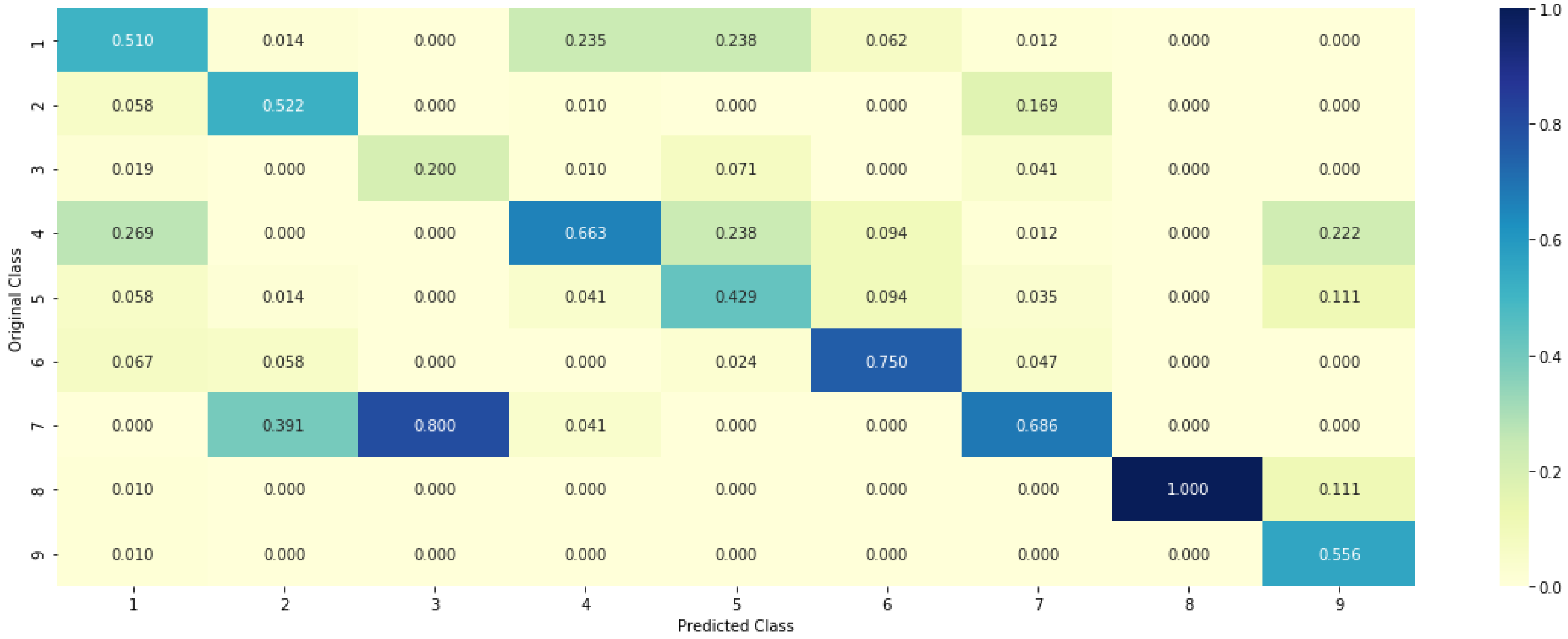
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.184061004326612
Number of missclassified point : 0.3966165413533835
-------------------- Confusion matrix --------------------
```

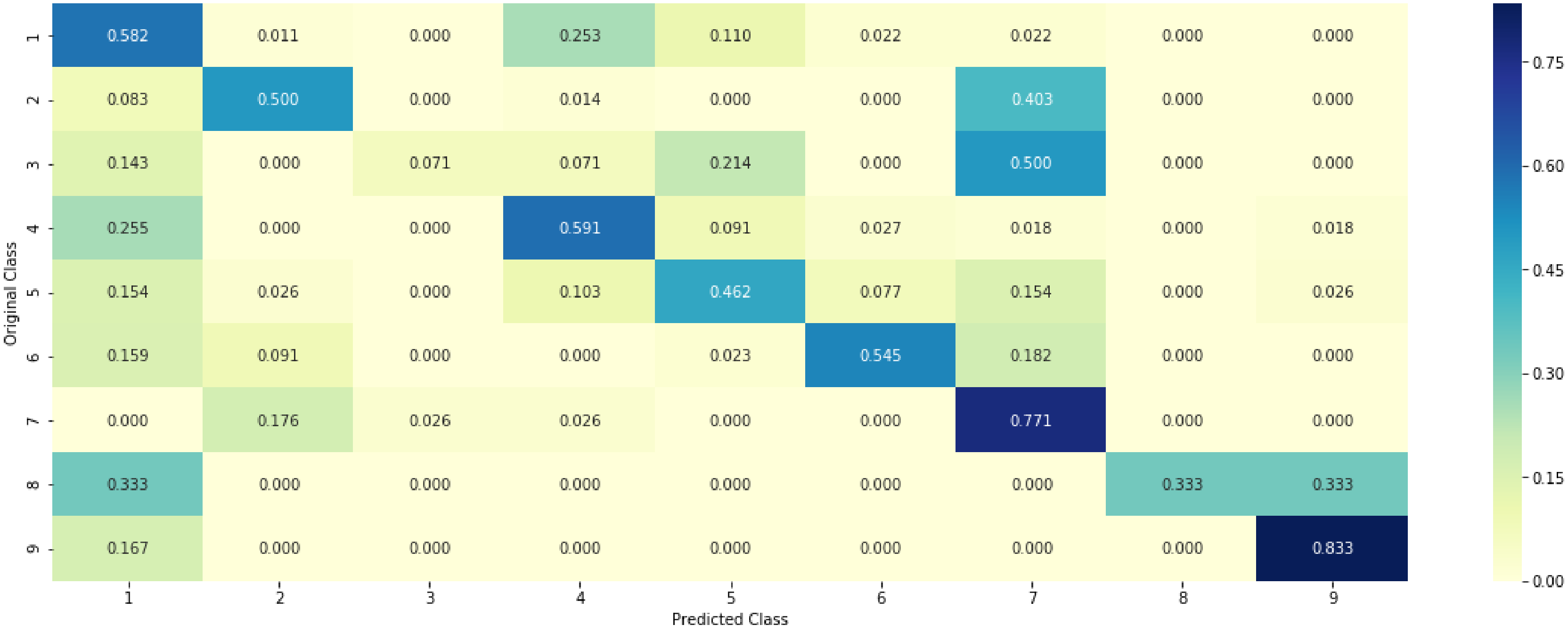


	1	2	3	4	5	6	7	8	9
1	53.000	1.000	0.000	23.000	10.000	2.000	2.000	0.000	0.000
2	6.000	36.000	0.000	1.000	0.000	0.000	29.000	0.000	0.000
3	2.000	0.000	1.000	1.000	3.000	0.000	7.000	0.000	0.000
4	28.000	0.000	0.000	65.000	10.000	3.000	2.000	0.000	2.000
5	6.000	1.000	0.000	4.000	18.000	3.000	6.000	0.000	1.000
6	7.000	4.000	0.000	0.000	1.000	24.000	8.000	0.000	0.000
7	0.000	27.000	4.000	4.000	0.000	0.000	118.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	5.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.582	0.011	0.000	0.253	0.110	0.022	0.022	0.000	0.000
2	0.083	0.500	0.000	0.014	0.000	0.000	0.403	0.000	0.000
3	0.143	0.000	0.071	0.071	0.214	0.000	0.500	0.000	0.000
4	0.255	0.000	0.000	0.591	0.091	0.027	0.018	0.000	0.018
5	0.154	0.026	0.000	0.103	0.462	0.077	0.154	0.000	0.026
6	0.159	0.091	0.000	0.000	0.023	0.545	0.182	0.000	0.000
7	0.000	0.176	0.026	0.026	0.000	0.000	0.771	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.333
9	0.167	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.833

4.1.1.3. Feature Importance, Correctly classified point

In [136]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5471 0.0455 0.0167 0.0574 0.0379 0.0367 0.2503 0.0046 0.0037]]
Actual Class : 1
--------------------------------------------------
8 Text feature [one] present in test data point [True]
9 Text feature [results] present in test data point [True]
11 Text feature [protein] present in test data point [True]
12 Text feature [dna] present in test data point [True]
13 Text feature [type] present in test data point [True]
14 Text feature [two] present in test data point [True]
15 Text feature [region] present in test data point [True]
16 Text feature [also] present in test data point [True]
18 Text feature [binding] present in test data point [True]
19 Text feature [loss] present in test data point [True]
21 Text feature [wild] present in test data point [True]
22 Text feature [however] present in test data point [True]
23 Text feature [table] present in test data point [True]
25 Text feature [either] present in test data point [True]
26 Text feature [role] present in test data point [True]
28 Text feature [may] present in test data point [True]
30 Text feature [well] present in test data point [True]
34 Text feature [analysis] present in test data point [True]
35 Text feature [used] present in test data point [True]
37 Text feature [result] present in test data point [True]
38 Text feature [three] present in test data point [True]
39 Text feature [large] present in test data point [True]
40 Text feature [using] present in test data point [True]
43 Text feature [specific] present in test data point [True]
46 Text feature [15] present in test data point [True]
47 Text feature [suggest] present in test data point [True]
49 Text feature [affect] present in test data point [True]
50 Text feature [shown] present in test data point [True]
52 Text feature [involved] present in test data point [True]
55 Text feature [data] present in test data point [True]
56 Text feature [present] present in test data point [True]
59 Text feature [several] present in test data point [True]
60 Text feature [proteins] present in test data point [True]
62 Text feature [gene] present in test data point [True]
65 Text feature [32] present in test data point [True]
66 Text feature [similar] present in test data point [True]
67 Text feature [based] present in test data point [True]
68 Text feature [observed] present in test data point [True]
69 Text feature [transcriptional] present in test data point [True]
```

```
74 Text feature [likely] present in test data point [True]
75 Text feature [transcription] present in test data point [True]
76 Text feature [although] present in test data point [True]
77 Text feature [additional] present in test data point [True]
78 Text feature [compared] present in test data point [True]
81 Text feature [single] present in test data point [True]
84 Text feature [including] present in test data point [True]
85 Text feature [previously] present in test data point [True]
87 Text feature [associated] present in test data point [True]
89 Text feature [different] present in test data point [True]
90 Text feature [addition] present in test data point [True]
92 Text feature [example] present in test data point [True]
93 Text feature [directly] present in test data point [True]
94 Text feature [deletion] present in test data point [True]
95 Text feature [cancer] present in test data point [True]
98 Text feature [previous] present in test data point [True]
Out of the top  100  features  55 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [137]:

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

```
Predicted Class : 9
Predicted Class Probabilities: [[0.1463 0.0776 0.0214 0.1093 0.0462 0.0448 0.1014 0.006  0.447 ]]
Actual Class : 7
--------------------------------------------------
6 Text feature [rna] present in test data point [True]
10 Text feature [mutant] present in test data point [True]
12 Text feature [substrate] present in test data point [True]
14 Text feature [alternative] present in test data point [True]
15 Text feature [levels] present in test data point [True]
32 Text feature [genes] present in test data point [True]
35 Text feature [wt] present in test data point [True]
36 Text feature [expressed] present in test data point [True]
37 Text feature [many] present in test data point [True]
39 Text feature [increase] present in test data point [True]
41 Text feature [observed] present in test data point [True]
43 Text feature [tagged] present in test data point [True]
45 Text feature [wild] present in test data point [True]
46 Text feature [using] present in test data point [True]
48 Text feature [relative] present in test data point [True]
53 Text feature [lysates] present in test data point [True]
55 Text feature [significant] present in test data point [True]
57 Text feature [together] present in test data point [True]
58 Text feature [figure] present in test data point [True]
59 Text feature [peptide] present in test data point [True]
60 Text feature [forms] present in test data point [True]
61 Text feature [samples] present in test data point [True]
62 Text feature [44] present in test data point [True]
63 Text feature [type] present in test data point [True]
64 Text feature [pattern] present in test data point [True]
65 Text feature [heterozygous] present in test data point [True]
66 Text feature [cells] present in test data point [True]
67 Text feature [example] present in test data point [True]
68 Text feature [cell] present in test data point [True]
69 Text feature [confirmed] present in test data point [True]
70 Text feature [significantly] present in test data point [True]
74 Text feature [suggest] present in test data point [True]
75 Text feature [complex] present in test data point [True]
76 Text feature [linked] present in test data point [True]
77 Text feature [specific] present in test data point [True]
78 Text feature [performed] present in test data point [True]
79 Text feature [state] present in test data point [True]
80 Text feature [found] present in test data point [True]
81 Text feature [series] present in test data point [True]
```

```
82 Text feature [2010] present in test data point [True]
84 Text feature [10] present in test data point [True]
85 Text feature [2009] present in test data point [True]
86 Text feature [2012] present in test data point [True]
88 Text feature [recently] present in test data point [True]
89 Text feature [set] present in test data point [True]
90 Text feature [expression] present in test data point [True]
92 Text feature [determined] present in test data point [True]
96 Text feature [total] present in test data point [True]
97 Text feature [university] present in test data point [True]
Out of the top  100  features  49 are present in query point
```

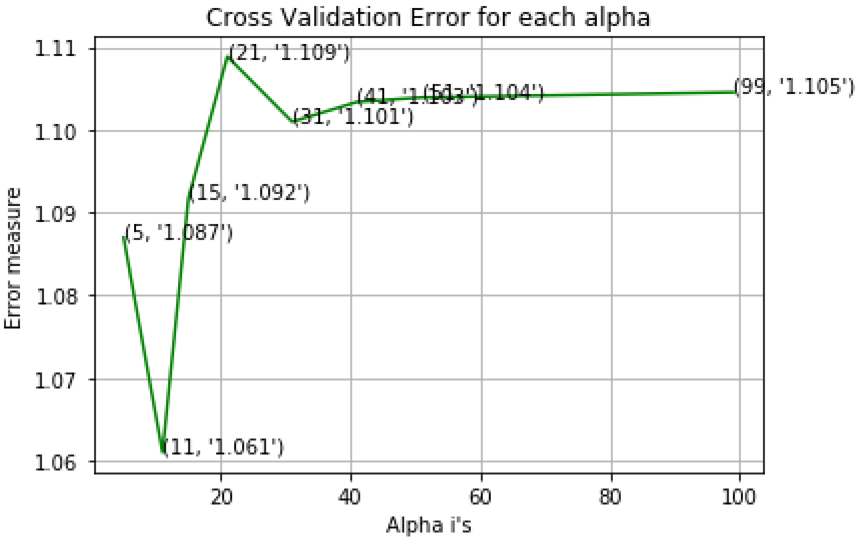
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [138]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighb
ors.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-geo
metric-intuition-with-a-toy-example-1/
#-------------------------------------


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


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

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


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

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

```
for alpha = 5
Log Loss : 1.0869499001165954
for alpha = 11
Log Loss : 1.0610181981509876
for alpha = 15
Log Loss : 1.0917005785707772
for alpha = 21
Log Loss : 1.108853001187891
for alpha = 31
Log Loss : 1.1009553794121465
for alpha = 41
Log Loss : 1.1033502037252916
for alpha = 51
Log Loss : 1.1038973531848624
for alpha = 99
Log Loss : 1.1045181352151239
```



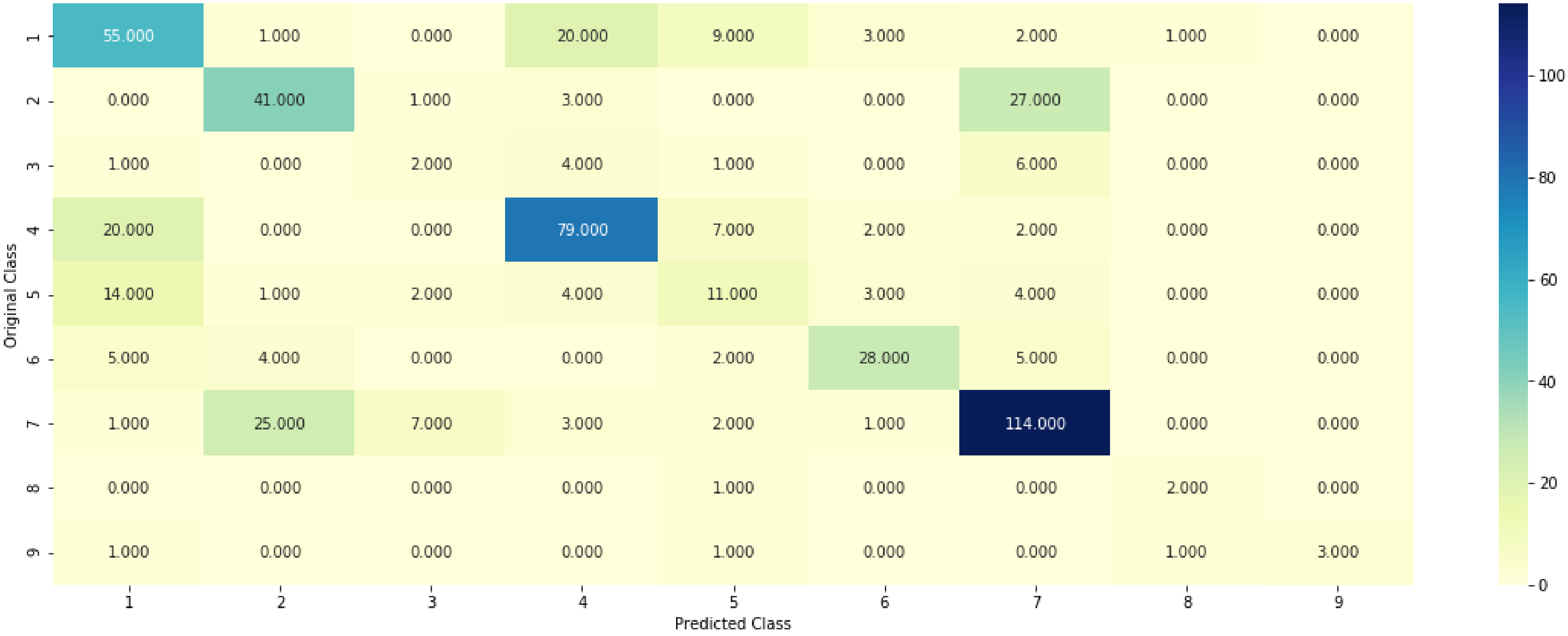
```
For values of best alpha =  11 The train log loss is: 0.6465937272340705
For values of best alpha =  11 The cross validation log loss is: 1.0610181981509876
For values of best alpha =  11 The test log loss is: 1.072694241678819
```

4.2.2. Testing the model with best hyper paramters

In [139]:

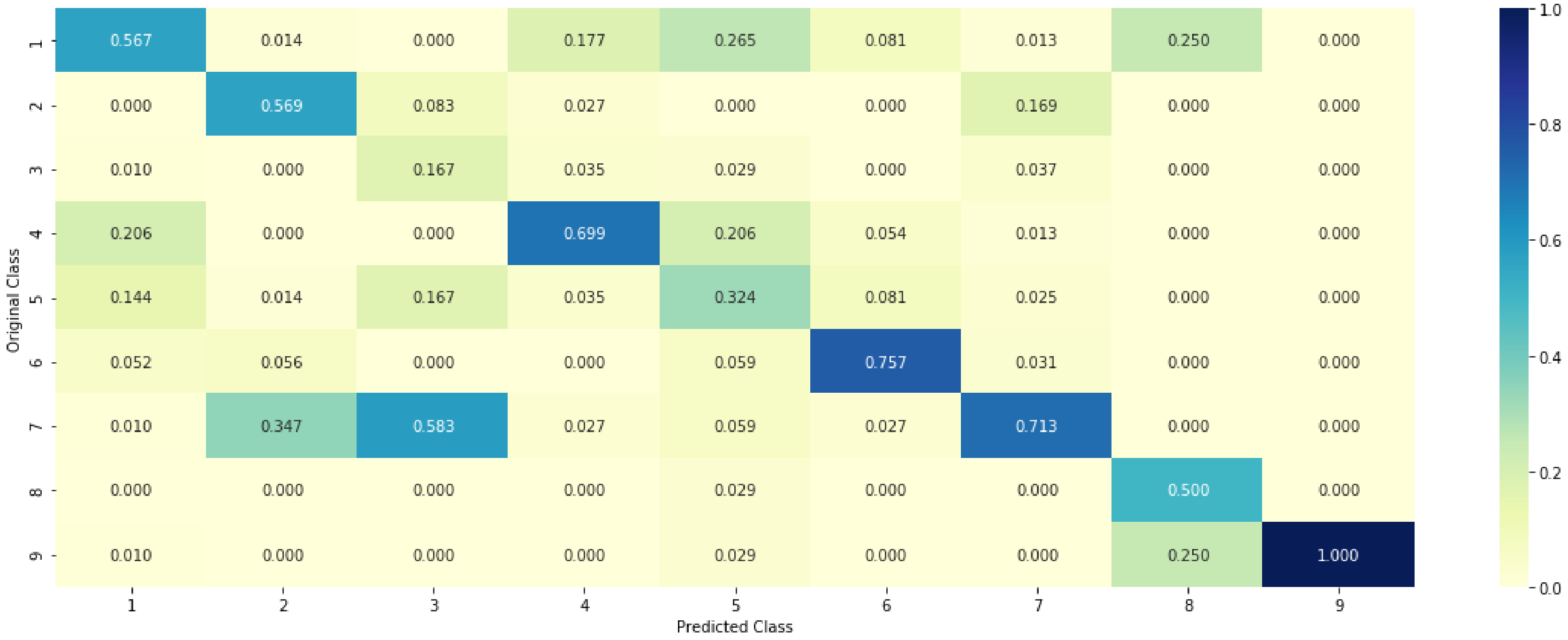
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=’uniform’, algorithm=’auto’, leaf_size=30, p=2,
# metric=’minkowski’, 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 : 1.0610181981509876
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```

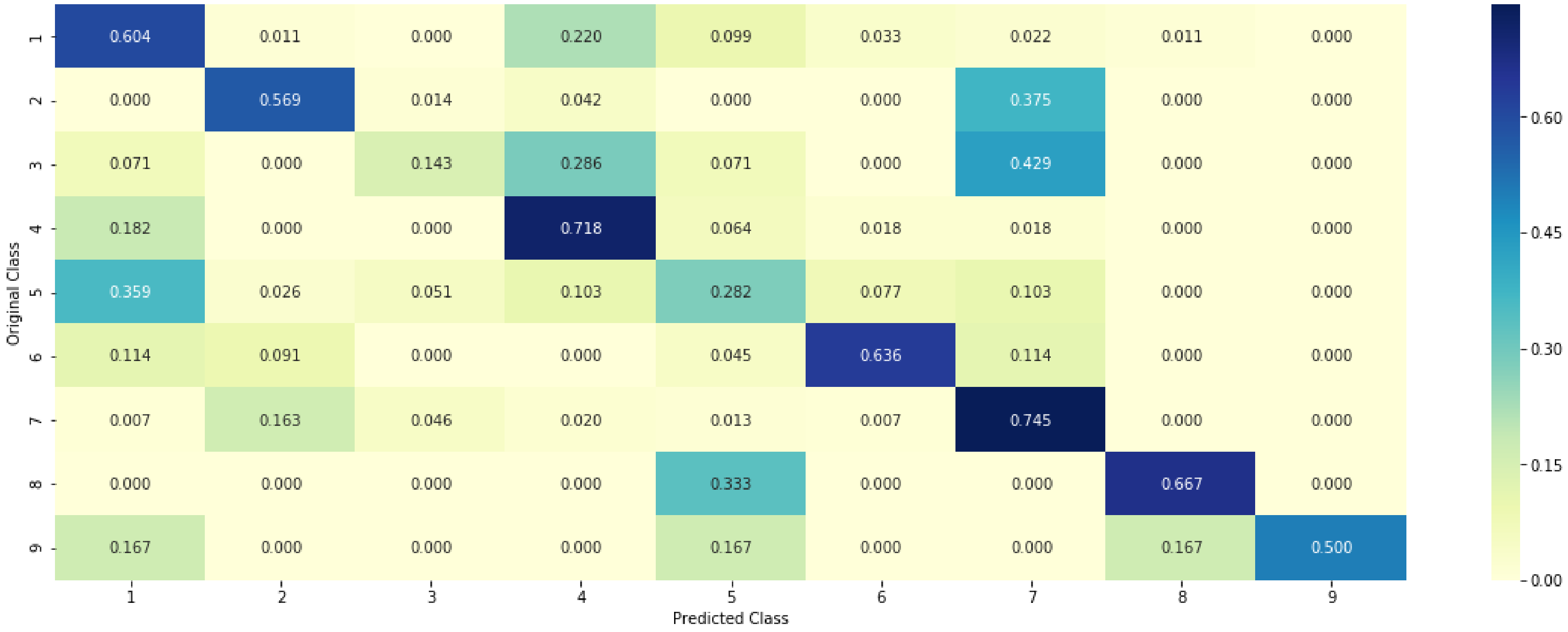


	1	2	3	4	5	6	7	8	9
1	55.000	1.000	0.000	20.000	9.000	3.000	2.000	1.000	0.000
2	0.000	41.000	1.000	3.000	0.000	0.000	27.000	0.000	0.000
3	1.000	0.000	2.000	4.000	1.000	0.000	6.000	0.000	0.000
4	20.000	0.000	0.000	79.000	7.000	2.000	2.000	0.000	0.000
5	14.000	1.000	2.000	4.000	11.000	3.000	4.000	0.000	0.000
6	5.000	4.000	0.000	0.000	2.000	28.000	5.000	0.000	0.000
7	1.000	25.000	7.000	3.000	2.000	1.000	114.000	0.000	0.000
8	0.000	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000
9	1.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000	3.000

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



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



4.2.3.Sample Query point -1

In [140]:

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

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

```
Predicted Class : 1
Actual Class : 1
The  11  nearest neighbours of the test points belongs to classes [1 7 1 1 7 1 1 7 1 7 7]
Fequency of nearest points : Counter({1: 6, 7: 5})
```

4.2.4. Sample Query Point-2

In [141]:

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

test_point_index = 100

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

```
Predicted Class : 2
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [2 1 1 9 1 8 9 2 4 2 4]
Fequency of nearest points : Counter({2: 3, 1: 3, 9: 2, 4: 2, 8: 1})
```

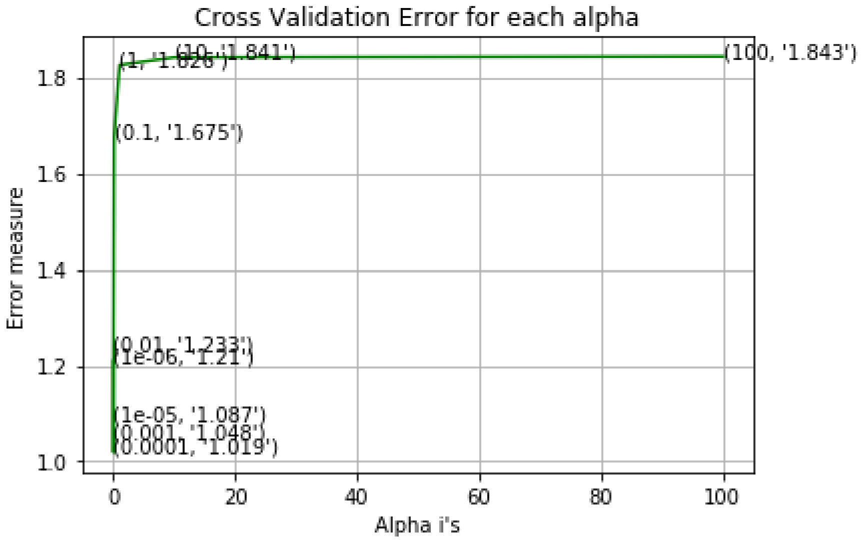
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [142]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=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.cal
ibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=’sigmoid’, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.209724895399181
for alpha = 1e-05
Log Loss : 1.0868874016286891
for alpha = 0.0001
Log Loss : 1.0190278980924616
for alpha = 0.001
Log Loss : 1.0478605496709168
for alpha = 0.01
Log Loss : 1.2330134763501672
for alpha = 0.1
Log Loss : 1.6749503234049399
for alpha = 1
Log Loss : 1.8255165879368478
for alpha = 10
Log Loss : 1.8413983380256034
for alpha = 100
Log Loss : 1.8431929945137084
```



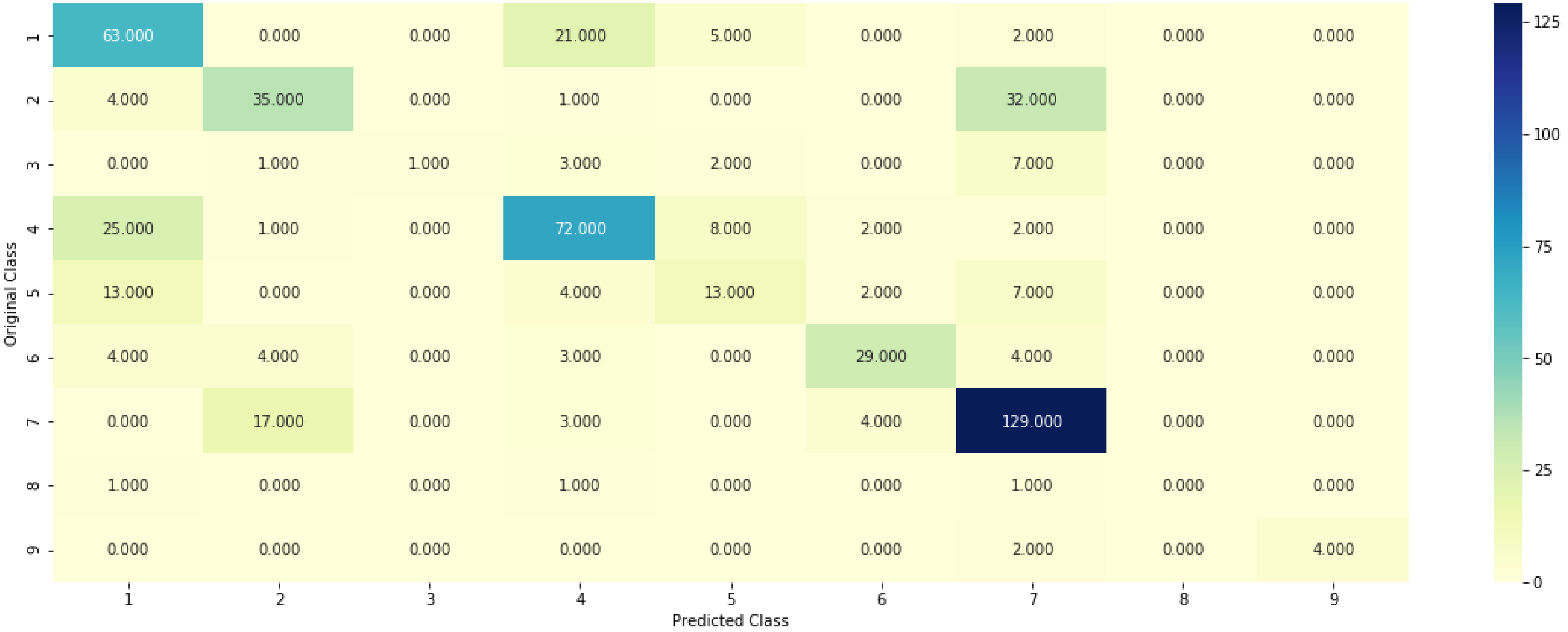
```
For values of best alpha =  0.0001 The train log loss is: 0.5759027098592892
For values of best alpha =  0.0001 The cross validation log loss is: 1.0190278980924616
For values of best alpha =  0.0001 The test log loss is: 0.9957701584696416
```

4.3.1.2. Testing the model with best hyper paramters

In [143]:

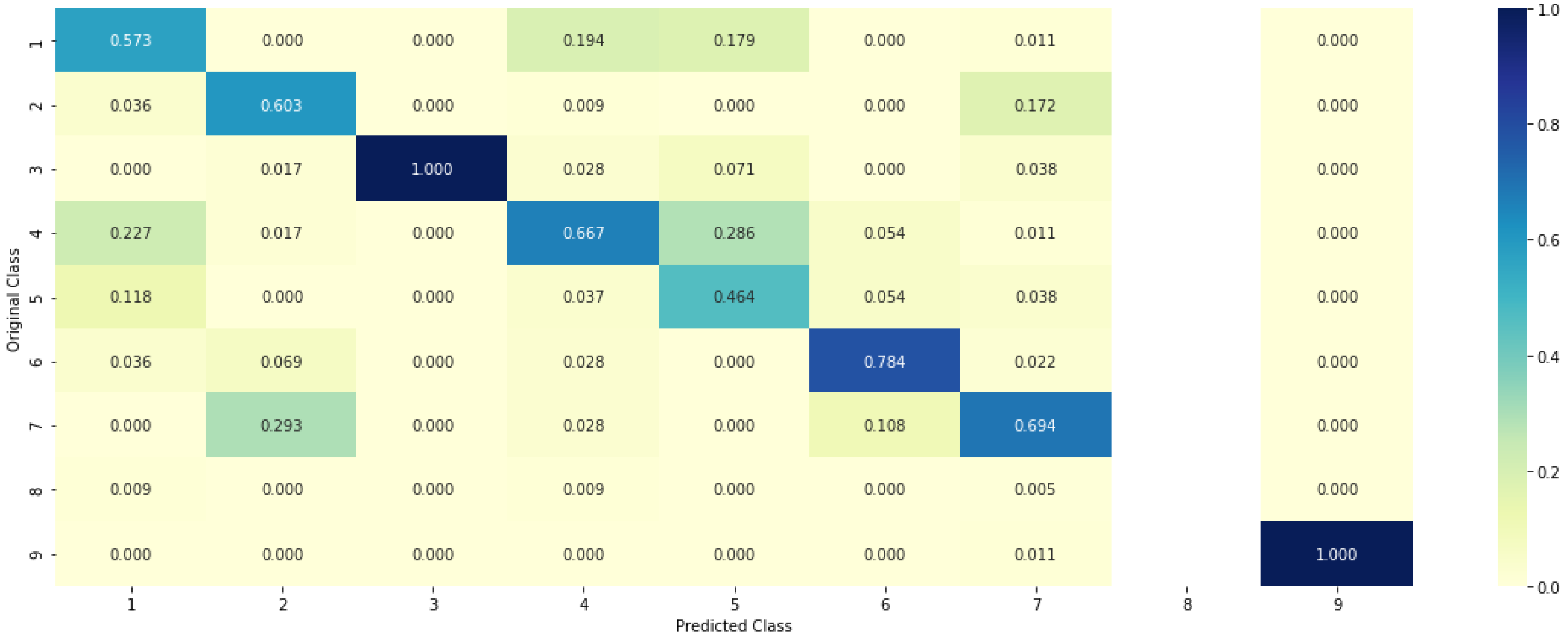
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

```
Log loss : 1.0190278980924616
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



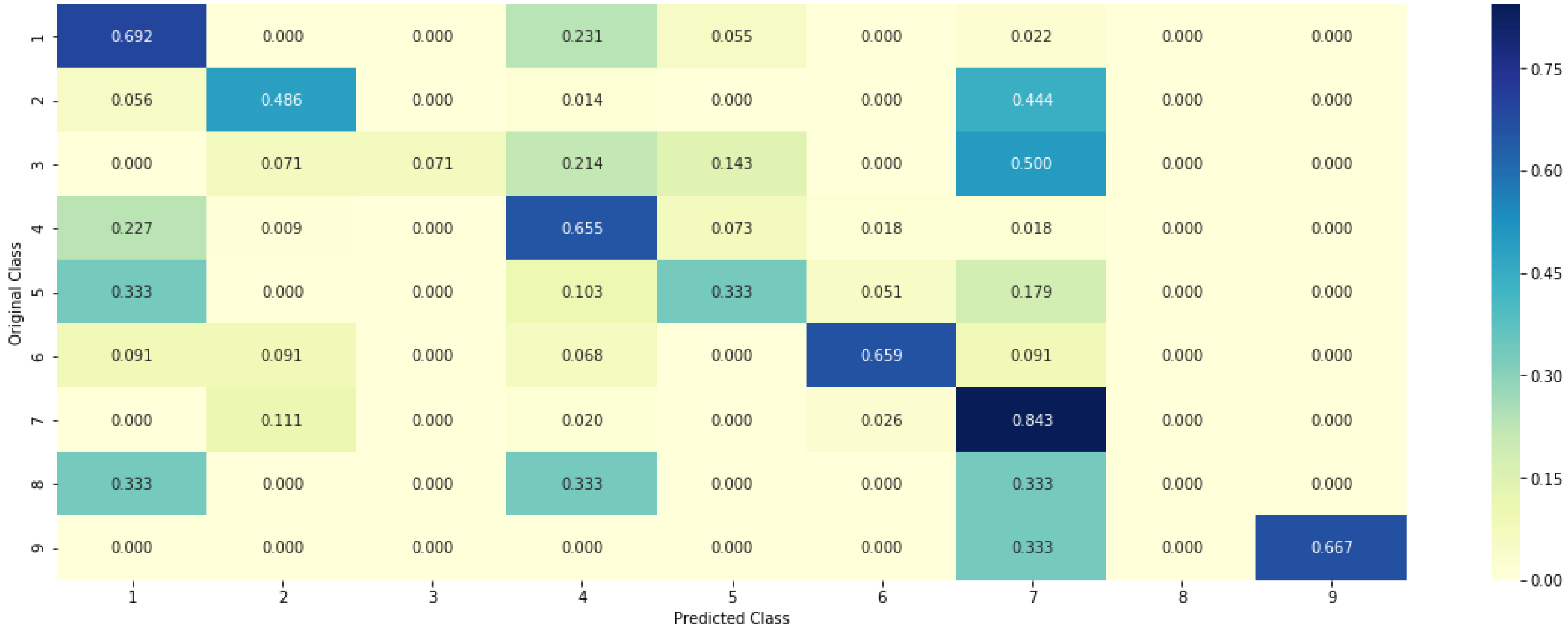
	1	2	3	4	5	6	7	8	9
1	63.000	0.000	0.000	21.000	5.000	0.000	2.000	0.000	0.000
2	4.000	35.000	0.000	1.000	0.000	0.000	32.000	0.000	0.000
3	0.000	1.000	1.000	3.000	2.000	0.000	7.000	0.000	0.000
4	25.000	1.000	0.000	72.000	8.000	2.000	2.000	0.000	0.000
5	13.000	0.000	0.000	4.000	13.000	2.000	7.000	0.000	0.000
6	4.000	4.000	0.000	3.000	0.000	29.000	4.000	0.000	0.000
7	0.000	17.000	0.000	3.000	0.000	4.000	129.000	0.000	0.000
8	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.573	0.000	0.000	0.194	0.179	0.000	0.011		0.000
2	0.036	0.603	0.000	0.009	0.000	0.000	0.172		0.000
3	0.000	0.017	1.000	0.028	0.071	0.000	0.038		0.000
4	0.227	0.017	0.000	0.667	0.286	0.054	0.011		0.000
5	0.118	0.000	0.000	0.037	0.464	0.054	0.038		0.000
6	0.036	0.069	0.000	0.028	0.000	0.784	0.022		0.000
7	0.000	0.293	0.000	0.028	0.000	0.108	0.694		0.000
8	0.009	0.000	0.000	0.009	0.000	0.000	0.005		0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.011		1.000

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



	1	2	3	4	5	6	7	8	9
1	0.692	0.000	0.000	0.231	0.055	0.000	0.022	0.000	0.000
2	0.056	0.486	0.000	0.014	0.000	0.000	0.444	0.000	0.000
3	0.000	0.071	0.071	0.214	0.143	0.000	0.500	0.000	0.000
4	0.227	0.009	0.000	0.655	0.073	0.018	0.018	0.000	0.000
5	0.333	0.000	0.000	0.103	0.333	0.051	0.179	0.000	0.000
6	0.091	0.091	0.000	0.068	0.000	0.659	0.091	0.000	0.000
7	0.000	0.111	0.000	0.020	0.000	0.026	0.843	0.000	0.000
8	0.333	0.000	0.000	0.333	0.000	0.000	0.333	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.667

4.3.1.3. Feature Importance

In [144]:

```
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 [145]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[0.4925 0.0747 0.0092 0.0437 0.0244 0.0177 0.3301 0.0051 0.0027]]
Actual Class : 1
--------------------------------------------------
54 Text feature [region] present in test data point [True]
97 Text feature [mutational] present in test data point [True]
98 Text feature [identify] present in test data point [True]
105 Text feature [59] present in test data point [True]
175 Text feature [transcriptional] present in test data point [True]
184 Text feature [binding] present in test data point [True]
188 Text feature [loss] present in test data point [True]
191 Text feature [sequenced] present in test data point [True]
196 Text feature [deletion] present in test data point [True]
200 Text feature [strand] present in test data point [True]
207 Text feature [insertion] present in test data point [True]
215 Text feature [sample] present in test data point [True]
219 Text feature [fold] present in test data point [True]
220 Text feature [wild] present in test data point [True]
246 Text feature [reverse] present in test data point [True]
249 Text feature [type] present in test data point [True]
251 Text feature [driven] present in test data point [True]
256 Text feature [dna] present in test data point [True]
257 Text feature [affect] present in test data point [True]
259 Text feature [transcription] present in test data point [True]
262 Text feature [across] present in test data point [True]
270 Text feature [one] present in test data point [True]
291 Text feature [subunit] present in test data point [True]
312 Text feature [present] present in test data point [True]
314 Text feature [large] present in test data point [True]
315 Text feature [ovarian] present in test data point [True]
319 Text feature [genome] present in test data point [True]
321 Text feature [obtained] present in test data point [True]
329 Text feature [interactions] present in test data point [True]
333 Text feature [protein] present in test data point [True]
337 Text feature [conserved] present in test data point [True]
340 Text feature [somatic] present in test data point [True]
341 Text feature [deletions] present in test data point [True]
344 Text feature [proteins] present in test data point [True]
345 Text feature [harboring] present in test data point [True]
349 Text feature [located] present in test data point [True]
353 Text feature [role] present in test data point [True]
361 Text feature [often] present in test data point [True]
363 Text feature [analyzed] present in test data point [True]
```

```
378 Text feature [dependent] present in test data point [True]
382 Text feature [22] present in test data point [True]
394 Text feature [development] present in test data point [True]
399 Text feature [previous] present in test data point [True]
402 Text feature [status] present in test data point [True]
404 Text feature [reported] present in test data point [True]
411 Text feature [damage] present in test data point [True]
412 Text feature [recently] present in test data point [True]
414 Text feature [terminal] present in test data point [True]
420 Text feature [tp53] present in test data point [True]
422 Text feature [progression] present in test data point [True]
425 Text feature [colony] present in test data point [True]
426 Text feature [shows] present in test data point [True]
429 Text feature [46] present in test data point [True]
431 Text feature [available] present in test data point [True]
434 Text feature [splice] present in test data point [True]
435 Text feature [next] present in test data point [True]
436 Text feature [www] present in test data point [True]
439 Text feature [example] present in test data point [True]
440 Text feature [cell] present in test data point [True]
449 Text feature [single] present in test data point [True]
450 Text feature [supplementary] present in test data point [True]
452 Text feature [common] present in test data point [True]
459 Text feature [sequencing] present in test data point [True]
462 Text feature [total] present in test data point [True]
464 Text feature [analyses] present in test data point [True]
469 Text feature [17] present in test data point [True]
470 Text feature [rna] present in test data point [True]
473 Text feature [involved] present in test data point [True]
476 Text feature [32] present in test data point [True]
480 Text feature [selection] present in test data point [True]
482 Text feature [cohort] present in test data point [True]
489 Text feature [based] present in test data point [True]
490 Text feature [negative] present in test data point [True]
494 Text feature [observed] present in test data point [True]
497 Text feature [six] present in test data point [True]
499 Text feature [grade] present in test data point [True]
Out of the top  500  features  76 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [146]:

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

```
Predicted Class : 9
Predicted Class Probabilities: [[0.2442 0.1443 0.0127 0.1619 0.018  0.0055 0.1178 0.0051 0.2904]]
Actual Class : 7
--------------------------------------------------
11 Text feature [mutant] present in test data point [True]
14 Text feature [wt] present in test data point [True]
15 Text feature [50] present in test data point [True]
19 Text feature [epithelial] present in test data point [True]
21 Text feature [expression] present in test data point [True]
25 Text feature [2012] present in test data point [True]
28 Text feature [figure] present in test data point [True]
33 Text feature [vector] present in test data point [True]
34 Text feature [linked] present in test data point [True]
44 Text feature [substrate] present in test data point [True]
45 Text feature [rna] present in test data point [True]
46 Text feature [levels] present in test data point [True]
47 Text feature [mediated] present in test data point [True]
48 Text feature [2013] present in test data point [True]
49 Text feature [reduced] present in test data point [True]
51 Text feature [suggest] present in test data point [True]
53 Text feature [significantly] present in test data point [True]
57 Text feature [overexpression] present in test data point [True]
58 Text feature [tagged] present in test data point [True]
59 Text feature [bone] present in test data point [True]
60 Text feature [subsequent] present in test data point [True]
64 Text feature [cyclin] present in test data point [True]
65 Text feature [heterozygous] present in test data point [True]
67 Text feature [set] present in test data point [True]
77 Text feature [alternative] present in test data point [True]
80 Text feature [2010] present in test data point [True]
81 Text feature [3b] present in test data point [True]
89 Text feature [together] present in test data point [True]
91 Text feature [peptide] present in test data point [True]
93 Text feature [research] present in test data point [True]
97 Text feature [washed] present in test data point [True]
99 Text feature [reverse] present in test data point [True]
101 Text feature [factors] present in test data point [True]
102 Text feature [cells] present in test data point [True]
106 Text feature [target] present in test data point [True]
108 Text feature [lysates] present in test data point [True]
113 Text feature [contribute] present in test data point [True]
119 Text feature [expressing] present in test data point [True]
120 Text feature [increase] present in test data point [True]
```

```
121 Text feature [normalized] present in test data point [True]
125 Text feature [whether] present in test data point [True]
126 Text feature [44] present in test data point [True]
127 Text feature [cell] present in test data point [True]
129 Text feature [potential] present in test data point [True]
130 Text feature [atp] present in test data point [True]
132 Text feature [anti] present in test data point [True]
136 Text feature [stable] present in test data point [True]
137 Text feature [www] present in test data point [True]
138 Text feature [flag] present in test data point [True]
139 Text feature [relative] present in test data point [True]
140 Text feature [drug] present in test data point [True]
142 Text feature [impact] present in test data point [True]
146 Text feature [university] present in test data point [True]
149 Text feature [exons] present in test data point [True]
152 Text feature [state] present in test data point [True]
158 Text feature [direct] present in test data point [True]
162 Text feature [provided] present in test data point [True]
166 Text feature [interface] present in test data point [True]
169 Text feature [like] present in test data point [True]
170 Text feature [blue] present in test data point [True]
171 Text feature [using] present in test data point [True]
173 Text feature [exon] present in test data point [True]
175 Text feature [indicating] present in test data point [True]
176 Text feature [genes] present in test data point [True]
178 Text feature [performed] present in test data point [True]
179 Text feature [subunit] present in test data point [True]
180 Text feature [versus] present in test data point [True]
182 Text feature [malignant] present in test data point [True]
185 Text feature [regulation] present in test data point [True]
186 Text feature [expressed] present in test data point [True]
187 Text feature [de] present in test data point [True]
188 Text feature [000] present in test data point [True]
193 Text feature [recent] present in test data point [True]
194 Text feature [directed] present in test data point [True]
196 Text feature [sample] present in test data point [True]
197 Text feature [changes] present in test data point [True]
199 Text feature [level] present in test data point [True]
201 Text feature [next] present in test data point [True]
203 Text feature [standard] present in test data point [True]
205 Text feature [hr] present in test data point [True]
207 Text feature [confirmed] present in test data point [True]
213 Text feature [affinity] present in test data point [True]
```

```
470 Text feature [locus] present in test data point [True]
494 Text feature [following] present in test data point [True]
Out of the top  500  features  84 are present in query point
```

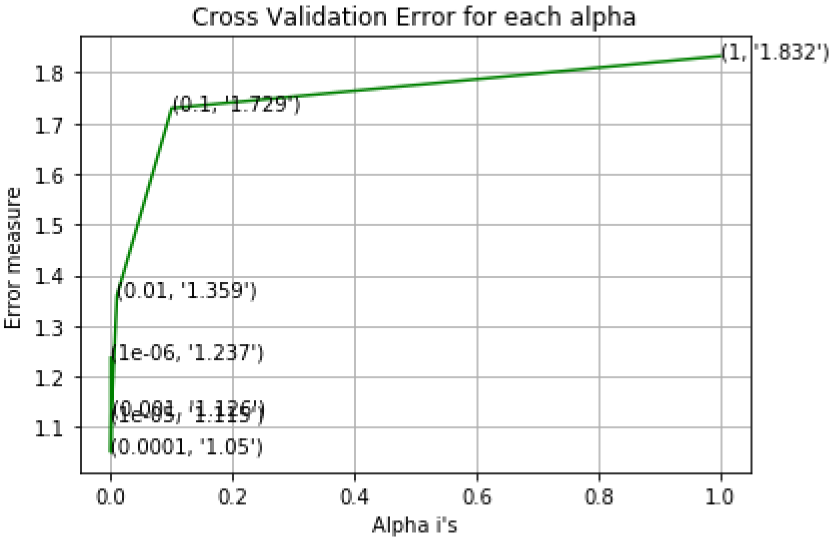
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [147]:

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

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2371095082718762
for alpha = 1e-05
Log Loss : 1.1145744034063754
for alpha = 0.0001
Log Loss : 1.0504606934375935
for alpha = 0.001
Log Loss : 1.1260980218560595
for alpha = 0.01
Log Loss : 1.359271824099488
for alpha = 0.1
Log Loss : 1.729396241314623
for alpha = 1
Log Loss : 1.8322767114368177
```



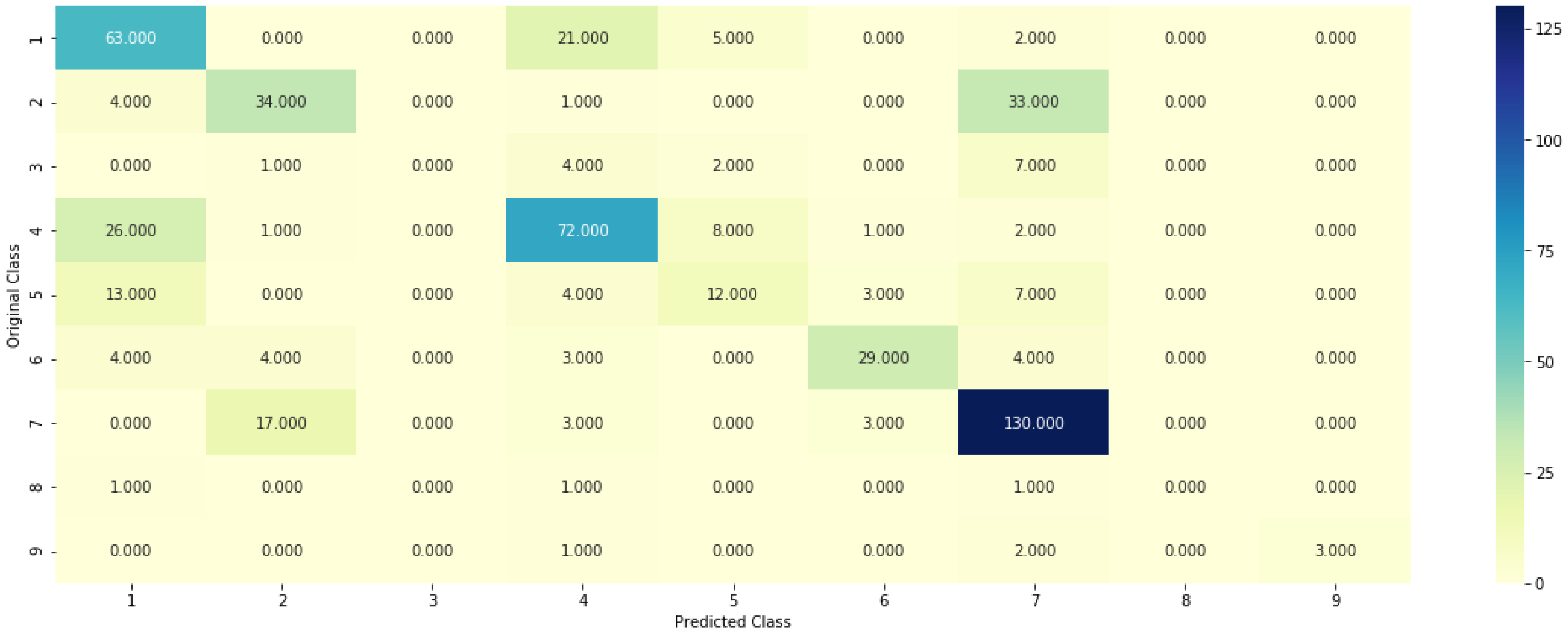
```
For values of best alpha =  0.0001 The train log loss is: 0.5749160961928956
For values of best alpha =  0.0001 The cross validation log loss is: 1.0504606934375935
For values of best alpha =  0.0001 The test log loss is: 1.0185354328631595
```

4.3.2.2. Testing model with best hyper parameters

In [148]:

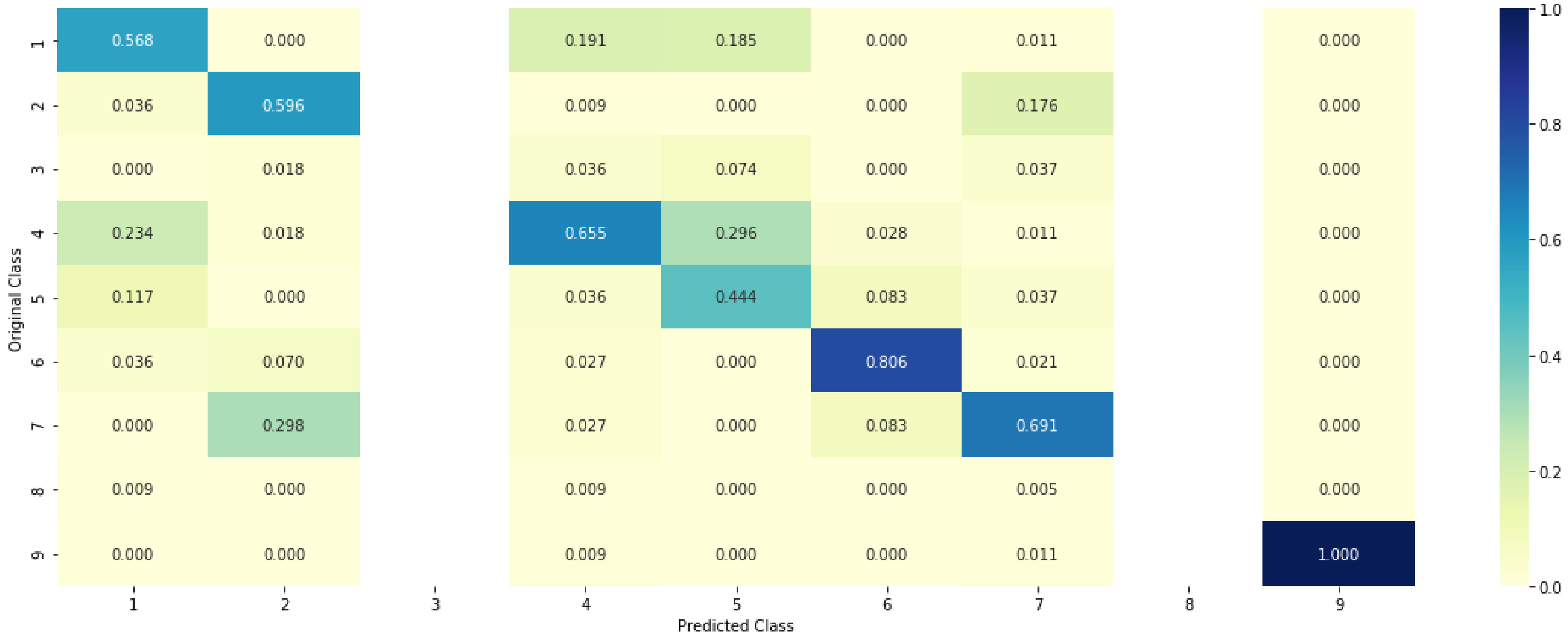
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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 : 1.0504606934375935
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```

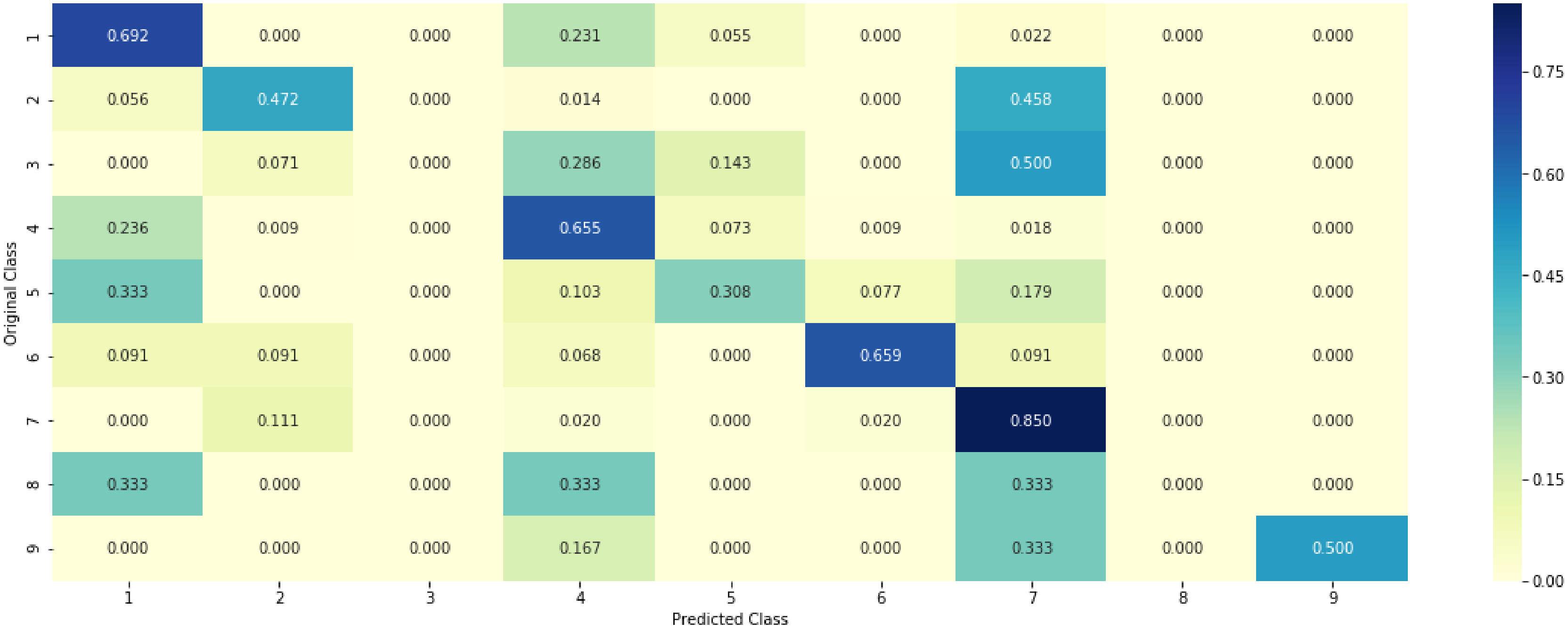


	1	2	3	4	5	6	7	8	9
1	63.000	0.000	0.000	21.000	5.000	0.000	2.000	0.000	0.000
2	4.000	34.000	0.000	1.000	0.000	0.000	33.000	0.000	0.000
3	0.000	1.000	0.000	4.000	2.000	0.000	7.000	0.000	0.000
4	26.000	1.000	0.000	72.000	8.000	1.000	2.000	0.000	0.000
5	13.000	0.000	0.000	4.000	12.000	3.000	7.000	0.000	0.000
6	4.000	4.000	0.000	3.000	0.000	29.000	4.000	0.000	0.000
7	0.000	17.000	0.000	3.000	0.000	3.000	130.000	0.000	0.000
8	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	3.000

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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [149]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4901 0.0705 0.01   0.0345 0.0232 0.0184 0.3448 0.006  0.0024]]
Actual Class : 1
--------------------------------------------------
55 Text feature [region] present in test data point [True]
85 Text feature [mutational] present in test data point [True]
89 Text feature [identify] present in test data point [True]
103 Text feature [59] present in test data point [True]
174 Text feature [loss] present in test data point [True]
180 Text feature [transcriptional] present in test data point [True]
182 Text feature [binding] present in test data point [True]
192 Text feature [strand] present in test data point [True]
197 Text feature [fold] present in test data point [True]
202 Text feature [wild] present in test data point [True]
205 Text feature [deletion] present in test data point [True]
207 Text feature [sequenced] present in test data point [True]
210 Text feature [insertion] present in test data point [True]
225 Text feature [sample] present in test data point [True]
235 Text feature [type] present in test data point [True]
243 Text feature [across] present in test data point [True]
244 Text feature [driven] present in test data point [True]
247 Text feature [reverse] present in test data point [True]
263 Text feature [dna] present in test data point [True]
270 Text feature [affect] present in test data point [True]
277 Text feature [one] present in test data point [True]
282 Text feature [ovarian] present in test data point [True]
286 Text feature [transcription] present in test data point [True]
293 Text feature [present] present in test data point [True]
311 Text feature [large] present in test data point [True]
315 Text feature [harboring] present in test data point [True]
325 Text feature [protein] present in test data point [True]
327 Text feature [subunit] present in test data point [True]
328 Text feature [often] present in test data point [True]
334 Text feature [obtained] present in test data point [True]
336 Text feature [proteins] present in test data point [True]
338 Text feature [interactions] present in test data point [True]
342 Text feature [somatic] present in test data point [True]
345 Text feature [genome] present in test data point [True]
355 Text feature [analyzed] present in test data point [True]
357 Text feature [22] present in test data point [True]
359 Text feature [role] present in test data point [True]
362 Text feature [dependent] present in test data point [True]
365 Text feature [conserved] present in test data point [True]
```

```
366 Text feature [located] present in test data point [True]
370 Text feature [recently] present in test data point [True]
375 Text feature [development] present in test data point [True]
377 Text feature [previous] present in test data point [True]
378 Text feature [status] present in test data point [True]
389 Text feature [common] present in test data point [True]
390 Text feature [tp53] present in test data point [True]
395 Text feature [reported] present in test data point [True]
399 Text feature [deletions] present in test data point [True]
401 Text feature [selection] present in test data point [True]
406 Text feature [damage] present in test data point [True]
409 Text feature [progression] present in test data point [True]
411 Text feature [shows] present in test data point [True]
413 Text feature [example] present in test data point [True]
417 Text feature [cell] present in test data point [True]
425 Text feature [46] present in test data point [True]
429 Text feature [colony] present in test data point [True]
437 Text feature [next] present in test data point [True]
442 Text feature [expressed] present in test data point [True]
444 Text feature [single] present in test data point [True]
446 Text feature [available] present in test data point [True]
447 Text feature [www] present in test data point [True]
456 Text feature [terminal] present in test data point [True]
466 Text feature [cohort] present in test data point [True]
467 Text feature [analyses] present in test data point [True]
471 Text feature [17] present in test data point [True]
476 Text feature [observed] present in test data point [True]
477 Text feature [sequencing] present in test data point [True]
483 Text feature [total] present in test data point [True]
488 Text feature [splice] present in test data point [True]
490 Text feature [less] present in test data point [True]
491 Text feature [metastatic] present in test data point [True]
494 Text feature [supplementary] present in test data point [True]
495 Text feature [involved] present in test data point [True]
496 Text feature [based] present in test data point [True]
Out of the top  500  features  74 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [150]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2811 0.1823 0.0132 0.1633 0.025  0.0076 0.1477 0.0086 0.1713]]
Actual Class : 7
--------------------------------------------------
55 Text feature [region] present in test data point [True]
59 Text feature [colorectal] present in test data point [True]
69 Text feature [deficient] present in test data point [True]
85 Text feature [mutational] present in test data point [True]
100 Text feature [normalized] present in test data point [True]
108 Text feature [corresponding] present in test data point [True]
127 Text feature [21] present in test data point [True]
151 Text feature [screening] present in test data point [True]
171 Text feature [position] present in test data point [True]
172 Text feature [peptide] present in test data point [True]
173 Text feature [gel] present in test data point [True]
174 Text feature [loss] present in test data point [True]
179 Text feature [encoding] present in test data point [True]
182 Text feature [binding] present in test data point [True]
183 Text feature [defined] present in test data point [True]
186 Text feature [vitro] present in test data point [True]
189 Text feature [early] present in test data point [True]
202 Text feature [wild] present in test data point [True]
203 Text feature [whole] present in test data point [True]
205 Text feature [deletion] present in test data point [True]
210 Text feature [insertion] present in test data point [True]
221 Text feature [hotspot] present in test data point [True]
222 Text feature [pa] present in test data point [True]
225 Text feature [sample] present in test data point [True]
229 Text feature [indicated] present in test data point [True]
233 Text feature [frequency] present in test data point [True]
234 Text feature [reduced] present in test data point [True]
235 Text feature [type] present in test data point [True]
239 Text feature [subjected] present in test data point [True]
241 Text feature [positions] present in test data point [True]
243 Text feature [across] present in test data point [True]
247 Text feature [reverse] present in test data point [True]
254 Text feature [within] present in test data point [True]
259 Text feature [tagged] present in test data point [True]
260 Text feature [allele] present in test data point [True]
262 Text feature [possible] present in test data point [True]
263 Text feature [dna] present in test data point [True]
266 Text feature [showing] present in test data point [True]
267 Text feature [effect] present in test data point [True]
```

```
273 Text feature [relative] present in test data point [True]
276 Text feature [calculated] present in test data point [True]
277 Text feature [one] present in test data point [True]
286 Text feature [transcription] present in test data point [True]
288 Text feature [general] present in test data point [True]
291 Text feature [displayed] present in test data point [True]
294 Text feature [revealed] present in test data point [True]
303 Text feature [assays] present in test data point [True]
311 Text feature [large] present in test data point [True]
317 Text feature [tumors] present in test data point [True]
320 Text feature [39] present in test data point [True]
325 Text feature [protein] present in test data point [True]
326 Text feature [gst] present in test data point [True]
327 Text feature [subunit] present in test data point [True]
330 Text feature [red] present in test data point [True]
332 Text feature [nucleotide] present in test data point [True]
334 Text feature [obtained] present in test data point [True]
335 Text feature [42] present in test data point [True]
337 Text feature [age] present in test data point [True]
342 Text feature [somatic] present in test data point [True]
345 Text feature [genome] present in test data point [True]
347 Text feature [upon] present in test data point [True]
348 Text feature [primers] present in test data point [True]
351 Text feature [therefore] present in test data point [True]
352 Text feature [derived] present in test data point [True]
355 Text feature [analyzed] present in test data point [True]
358 Text feature [hr] present in test data point [True]
359 Text feature [role] present in test data point [True]
360 Text feature [carcinomas] present in test data point [True]
365 Text feature [conserved] present in test data point [True]
368 Text feature [flag] present in test data point [True]
370 Text feature [recently] present in test data point [True]
373 Text feature [samples] present in test data point [True]
375 Text feature [development] present in test data point [True]
377 Text feature [previous] present in test data point [True]
378 Text feature [status] present in test data point [True]
382 Text feature [impaired] present in test data point [True]
389 Text feature [common] present in test data point [True]
394 Text feature [purified] present in test data point [True]
395 Text feature [reported] present in test data point [True]
399 Text feature [deletions] present in test data point [True]
407 Text feature [types] present in test data point [True]
413 Text feature [example] present in test data point [True]
```

```
414 Text feature [interface] present in test data point [True]
415 Text feature [tumor] present in test data point [True]
416 Text feature [manner] present in test data point [True]
417 Text feature [cell] present in test data point [True]
419 Text feature [coding] present in test data point [True]
420 Text feature [assay] present in test data point [True]
421 Text feature [structural] present in test data point [True]
422 Text feature [containing] present in test data point [True]
430 Text feature [epithelial] present in test data point [True]
437 Text feature [next] present in test data point [True]
438 Text feature [heterozygous] present in test data point [True]
442 Text feature [expressed] present in test data point [True]
444 Text feature [single] present in test data point [True]
446 Text feature [available] present in test data point [True]
447 Text feature [www] present in test data point [True]
448 Text feature [manufacturer] present in test data point [True]
451 Text feature [80] present in test data point [True]
452 Text feature [line] present in test data point [True]
456 Text feature [terminal] present in test data point [True]
459 Text feature [method] present in test data point [True]
460 Text feature [exon] present in test data point [True]
465 Text feature [subsequent] present in test data point [True]
467 Text feature [analyses] present in test data point [True]
468 Text feature [pcr] present in test data point [True]
471 Text feature [17] present in test data point [True]
472 Text feature [de] present in test data point [True]
476 Text feature [observed] present in test data point [True]
477 Text feature [sequencing] present in test data point [True]
478 Text feature [years] present in test data point [True]
482 Text feature [eight] present in test data point [True]
483 Text feature [total] present in test data point [True]
485 Text feature [cause] present in test data point [True]
486 Text feature [major] present in test data point [True]
489 Text feature [many] present in test data point [True]
490 Text feature [less] present in test data point [True]
496 Text feature [based] present in test data point [True]
Out of the top  500  features  118 are present in query point
```

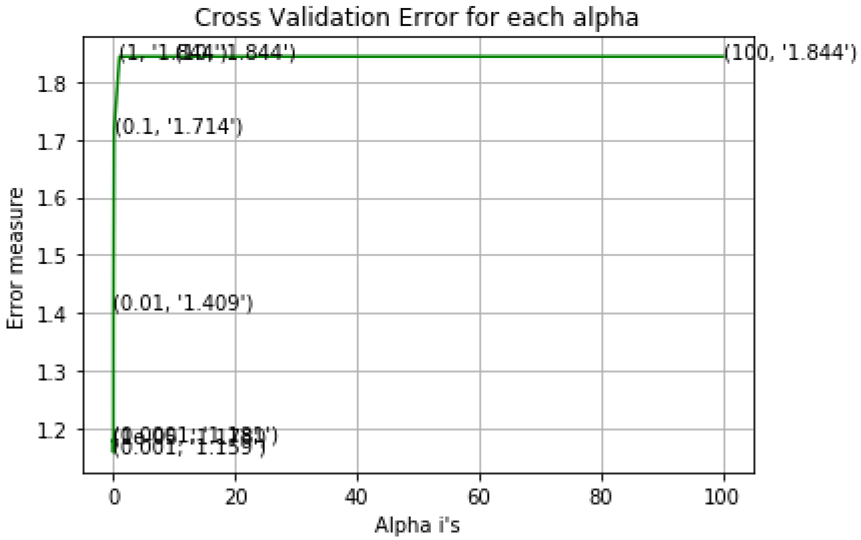
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [151]:

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

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

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



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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.1783592641340583
for C = 0.0001
Log Loss : 1.1812161356927675
for C = 0.001
Log Loss : 1.1586669123410203
for C = 0.01
Log Loss : 1.4094817231113508
for C = 0.1
Log Loss : 1.7142542435488812
for C = 1
Log Loss : 1.8436458729179581
for C = 10
Log Loss : 1.8436459028529832
for C = 100
Log Loss : 1.84364588220697
```



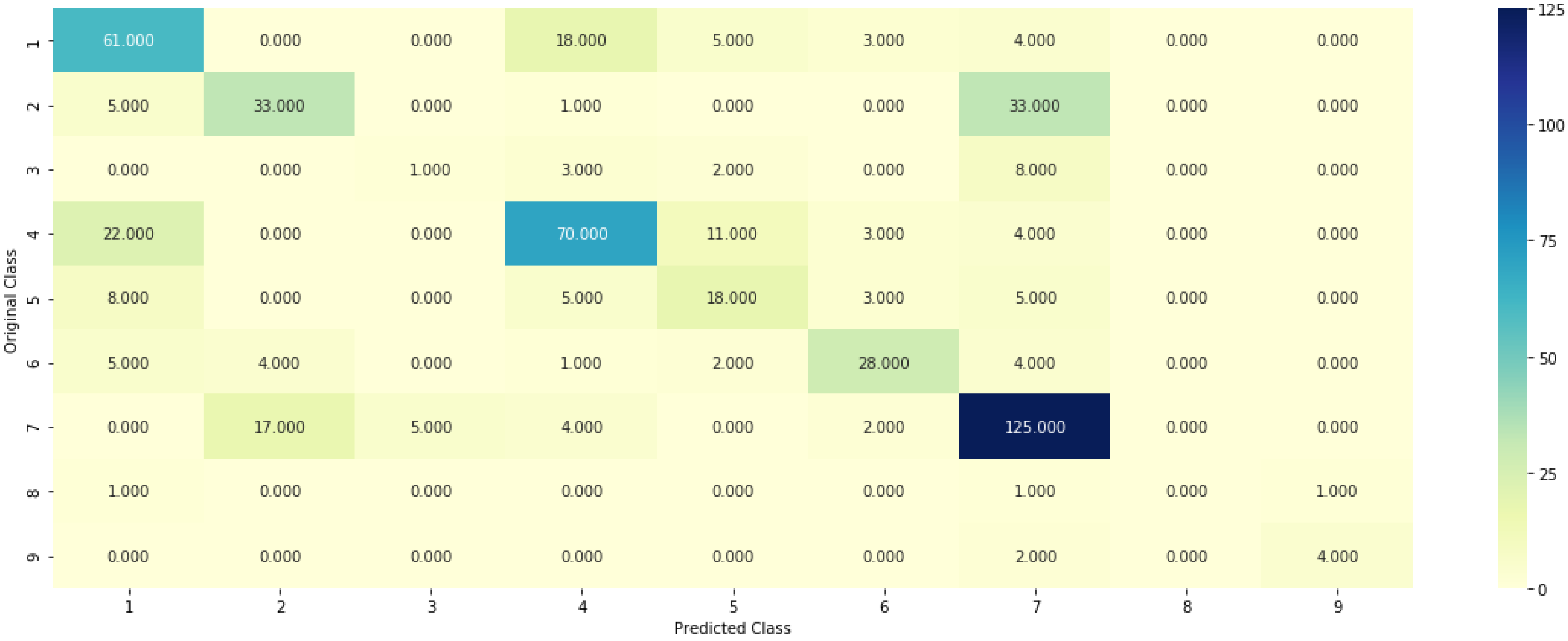
```
For values of best alpha =  0.001 The train log loss is: 0.8094454102790029
For values of best alpha =  0.001 The cross validation log loss is: 1.1586669123410203
For values of best alpha =  0.001 The test log loss is: 1.125786475804188
```

4.4.2. Testing model with best hyper parameters

In [152]:

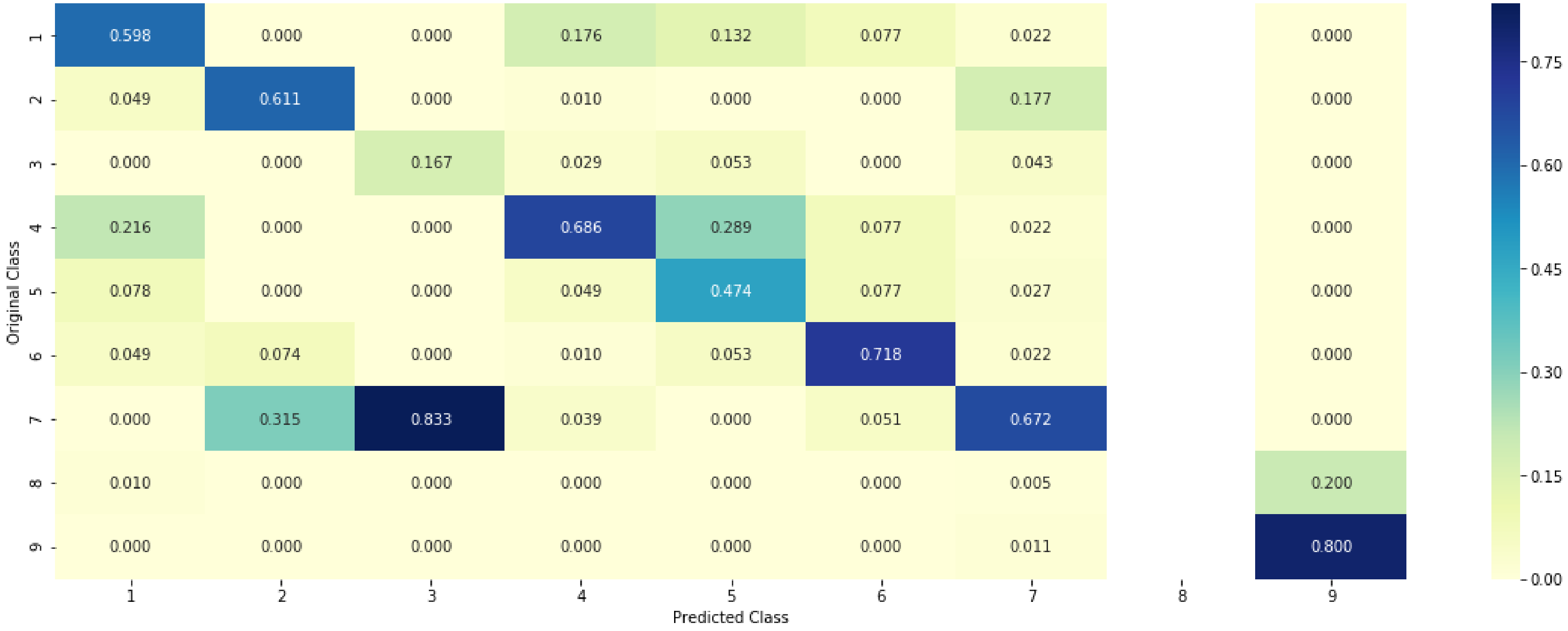
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 1.1586669123410203
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



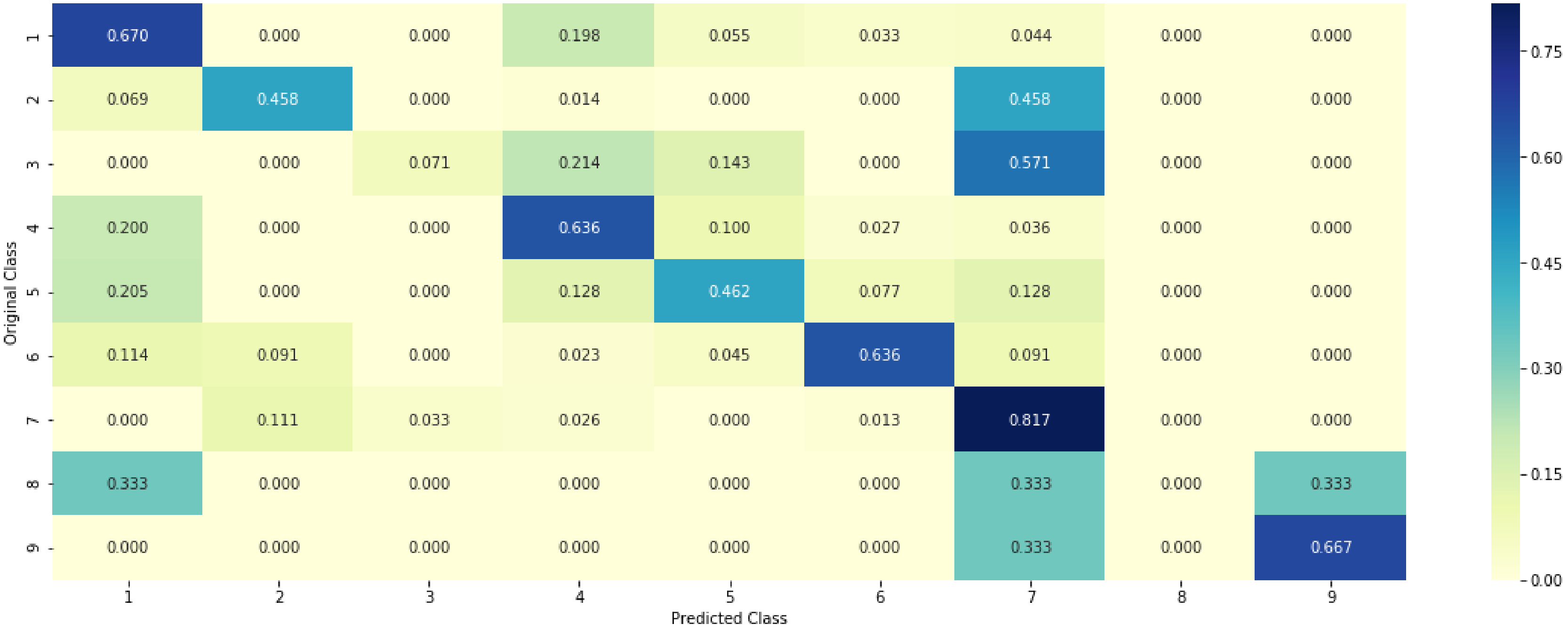
	1	2	3	4	5	6	7	8	9
1	61.000	0.000	0.000	18.000	5.000	3.000	4.000	0.000	0.000
2	5.000	33.000	0.000	1.000	0.000	0.000	33.000	0.000	0.000
3	0.000	0.000	1.000	3.000	2.000	0.000	8.000	0.000	0.000
4	22.000	0.000	0.000	70.000	11.000	3.000	4.000	0.000	0.000
5	8.000	0.000	0.000	5.000	18.000	3.000	5.000	0.000	0.000
6	5.000	4.000	0.000	1.000	2.000	28.000	4.000	0.000	0.000
7	0.000	17.000	5.000	4.000	0.000	2.000	125.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.598	0.000	0.000	0.176	0.132	0.077	0.022		0.000
2	0.049	0.611	0.000	0.010	0.000	0.000	0.177		0.000
3	0.000	0.000	0.167	0.029	0.053	0.000	0.043		0.000
4	0.216	0.000	0.000	0.686	0.289	0.077	0.022		0.000
5	0.078	0.000	0.000	0.049	0.474	0.077	0.027		0.000
6	0.049	0.074	0.000	0.010	0.053	0.718	0.022		0.000
7	0.000	0.315	0.833	0.039	0.000	0.051	0.672		0.000
8	0.010	0.000	0.000	0.000	0.000	0.000	0.005		0.200
9	0.000	0.000	0.000	0.000	0.000	0.000	0.011		0.800

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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [153]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4471 0.0702 0.0154 0.1164 0.0353 0.0391 0.2677 0.006  0.0028]]
Actual Class : 1
--------------------------------------------------
177 Text feature [binding] present in test data point [True]
183 Text feature [subunit] present in test data point [True]
184 Text feature [identify] present in test data point [True]
186 Text feature [region] present in test data point [True]
188 Text feature [harboring] present in test data point [True]
190 Text feature [59] present in test data point [True]
196 Text feature [affect] present in test data point [True]
200 Text feature [driven] present in test data point [True]
203 Text feature [interactions] present in test data point [True]
208 Text feature [loss] present in test data point [True]
212 Text feature [mutational] present in test data point [True]
213 Text feature [transcriptional] present in test data point [True]
214 Text feature [located] present in test data point [True]
221 Text feature [sequenced] present in test data point [True]
222 Text feature [insertion] present in test data point [True]
224 Text feature [reverse] present in test data point [True]
239 Text feature [progression] present in test data point [True]
243 Text feature [sample] present in test data point [True]
259 Text feature [fold] present in test data point [True]
260 Text feature [type] present in test data point [True]
263 Text feature [wild] present in test data point [True]
264 Text feature [next] present in test data point [True]
267 Text feature [expressed] present in test data point [True]
268 Text feature [deletion] present in test data point [True]
271 Text feature [one] present in test data point [True]
283 Text feature [al] present in test data point [True]
285 Text feature [et] present in test data point [True]
287 Text feature [cell] present in test data point [True]
291 Text feature [transcription] present in test data point [True]
302 Text feature [chemotherapy] present in test data point [True]
308 Text feature [46] present in test data point [True]
309 Text feature [obtained] present in test data point [True]
310 Text feature [reported] present in test data point [True]
316 Text feature [dna] present in test data point [True]
317 Text feature [proteins] present in test data point [True]
323 Text feature [deletions] present in test data point [True]
339 Text feature [supplementary] present in test data point [True]
341 Text feature [somatic] present in test data point [True]
344 Text feature [protein] present in test data point [True]
```

```
347 Text feature [stable] present in test data point [True]
349 Text feature [well] present in test data point [True]
354 Text feature [example] present in test data point [True]
355 Text feature [across] present in test data point [True]
367 Text feature [11] present in test data point [True]
379 Text feature [even] present in test data point [True]
381 Text feature [assessed] present in test data point [True]
383 Text feature [22] present in test data point [True]
385 Text feature [fig] present in test data point [True]
386 Text feature [previous] present in test data point [True]
389 Text feature [recently] present in test data point [True]
390 Text feature [conserved] present in test data point [True]
391 Text feature [development] present in test data point [True]
397 Text feature [analyses] present in test data point [True]
400 Text feature [ovarian] present in test data point [True]
405 Text feature [analyzed] present in test data point [True]
407 Text feature [metastatic] present in test data point [True]
410 Text feature [13] present in test data point [True]
413 Text feature [total] present in test data point [True]
418 Text feature [strand] present in test data point [True]
419 Text feature [significantly] present in test data point [True]
422 Text feature [treated] present in test data point [True]
425 Text feature [single] present in test data point [True]
428 Text feature [large] present in test data point [True]
429 Text feature [present] present in test data point [True]
432 Text feature [independent] present in test data point [True]
433 Text feature [growth] present in test data point [True]
449 Text feature [results] present in test data point [True]
452 Text feature [24] present in test data point [True]
457 Text feature [4a] present in test data point [True]
465 Text feature [multiple] present in test data point [True]
466 Text feature [rna] present in test data point [True]
472 Text feature [less] present in test data point [True]
474 Text feature [terminal] present in test data point [True]
476 Text feature [17] present in test data point [True]
478 Text feature [proliferation] present in test data point [True]
479 Text feature [observations] present in test data point [True]
481 Text feature [mediated] present in test data point [True]
482 Text feature [mutation] present in test data point [True]
483 Text feature [gene] present in test data point [True]
487 Text feature [role] present in test data point [True]
494 Text feature [compared] present in test data point [True]
497 Text feature [status] present in test data point [True]
```

```
499 Text feature [evidence] present in test data point [True]
Out of the top  500  features  83 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [154]:

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

```
Predicted Class : 9
Predicted Class Probabilities: [[0.1513 0.1506 0.0146 0.1607 0.027  0.0082 0.1808 0.0042 0.3026]]
Actual Class : 7
--------------------------------------------------
15 Text feature [mutant] present in test data point [True]
17 Text feature [rna] present in test data point [True]
18 Text feature [substrate] present in test data point [True]
21 Text feature [linked] present in test data point [True]
22 Text feature [wt] present in test data point [True]
30 Text feature [tagged] present in test data point [True]
33 Text feature [heterozygous] present in test data point [True]
35 Text feature [levels] present in test data point [True]
36 Text feature [bone] present in test data point [True]
37 Text feature [2012] present in test data point [True]
38 Text feature [2013] present in test data point [True]
40 Text feature [alternative] present in test data point [True]
46 Text feature [peptide] present in test data point [True]
49 Text feature [set] present in test data point [True]
56 Text feature [relative] present in test data point [True]
58 Text feature [standard] present in test data point [True]
60 Text feature [subsequent] present in test data point [True]
64 Text feature [research] present in test data point [True]
66 Text feature [series] present in test data point [True]
67 Text feature [impact] present in test data point [True]
68 Text feature [unique] present in test data point [True]
71 Text feature [core] present in test data point [True]
72 Text feature [example] present in test data point [True]
74 Text feature [state] present in test data point [True]
77 Text feature [system] present in test data point [True]
78 Text feature [significant] present in test data point [True]
79 Text feature [000] present in test data point [True]
80 Text feature [university] present in test data point [True]
81 Text feature [cellular] present in test data point [True]
82 Text feature [exons] present in test data point [True]
83 Text feature [44] present in test data point [True]
84 Text feature [forms] present in test data point [True]
85 Text feature [significantly] present in test data point [True]
87 Text feature [western] present in test data point [True]
88 Text feature [complex] present in test data point [True]
93 Text feature [materials] present in test data point [True]
94 Text feature [pattern] present in test data point [True]
95 Text feature [versus] present in test data point [True]
96 Text feature [malignant] present in test data point [True]
```

```
99 Text feature [using] present in test data point [True]
104 Text feature [followed] present in test data point [True]
107 Text feature [current] present in test data point [True]
108 Text feature [exon] present in test data point [True]
111 Text feature [many] present in test data point [True]
112 Text feature [clear] present in test data point [True]
114 Text feature [increase] present in test data point [True]
115 Text feature [washed] present in test data point [True]
116 Text feature [suggest] present in test data point [True]
118 Text feature [specific] present in test data point [True]
119 Text feature [status] present in test data point [True]
121 Text feature [tissue] present in test data point [True]
123 Text feature [blot] present in test data point [True]
124 Text feature [observed] present in test data point [True]
127 Text feature [sample] present in test data point [True]
129 Text feature [expressed] present in test data point [True]
132 Text feature [affinity] present in test data point [True]
135 Text feature [indicating] present in test data point [True]
137 Text feature [associated] present in test data point [True]
138 Text feature [wild] present in test data point [True]
139 Text feature [flag] present in test data point [True]
141 Text feature [samples] present in test data point [True]
145 Text feature [10] present in test data point [True]
146 Text feature [genes] present in test data point [True]
150 Text feature [positions] present in test data point [True]
151 Text feature [performed] present in test data point [True]
152 Text feature [locus] present in test data point [True]
155 Text feature [interface] present in test data point [True]
158 Text feature [purified] present in test data point [True]
160 Text feature [provided] present in test data point [True]
165 Text feature [displayed] present in test data point [True]
166 Text feature [subjected] present in test data point [True]
169 Text feature [genome] present in test data point [True]
170 Text feature [3b] present in test data point [True]
173 Text feature [particular] present in test data point [True]
177 Text feature [recent] present in test data point [True]
178 Text feature [changes] present in test data point [True]
179 Text feature [important] present in test data point [True]
180 Text feature [gel] present in test data point [True]
183 Text feature [top] present in test data point [True]
184 Text feature [applied] present in test data point [True]
186 Text feature [mediated] present in test data point [True]
189 Text feature [together] present in test data point [True]
```

```
190 Text feature [mutated] present in test data point [True]
194 Text feature [impaired] present in test data point [True]
195 Text feature [generation] present in test data point [True]
197 Text feature [indicated] present in test data point [True]
198 Text feature [determined] present in test data point [True]
201 Text feature [distribution] present in test data point [True]
203 Text feature [lysates] present in test data point [True]
204 Text feature [direct] present in test data point [True]
207 Text feature [target] present in test data point [True]
210 Text feature [normalized] present in test data point [True]
213 Text feature [cyclin] present in test data point [True]
217 Text feature [containing] present in test data point [True]
218 Text feature [cells] present in test data point [True]
223 Text feature [common] present in test data point [True]
224 Text feature [method] present in test data point [True]
225 Text feature [type] present in test data point [True]
227 Text feature [contribute] present in test data point [True]
229 Text feature [eight] present in test data point [True]
231 Text feature [subunit] present in test data point [True]
233 Text feature [50] present in test data point [True]
238 Text feature [19] present in test data point [True]
239 Text feature [whole] present in test data point [True]
240 Text feature [line] present in test data point [True]
241 Text feature [lower] present in test data point [True]
247 Text feature [cell] present in test data point [True]
250 Text feature [conditions] present in test data point [True]
256 Text feature [position] present in test data point [True]
258 Text feature [large] present in test data point [True]
259 Text feature [average] present in test data point [True]
261 Text feature [de] present in test data point [True]
262 Text feature [methods] present in test data point [True]
Out of the top  500  features  113 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [155]:

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

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

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

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
```

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

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

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

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

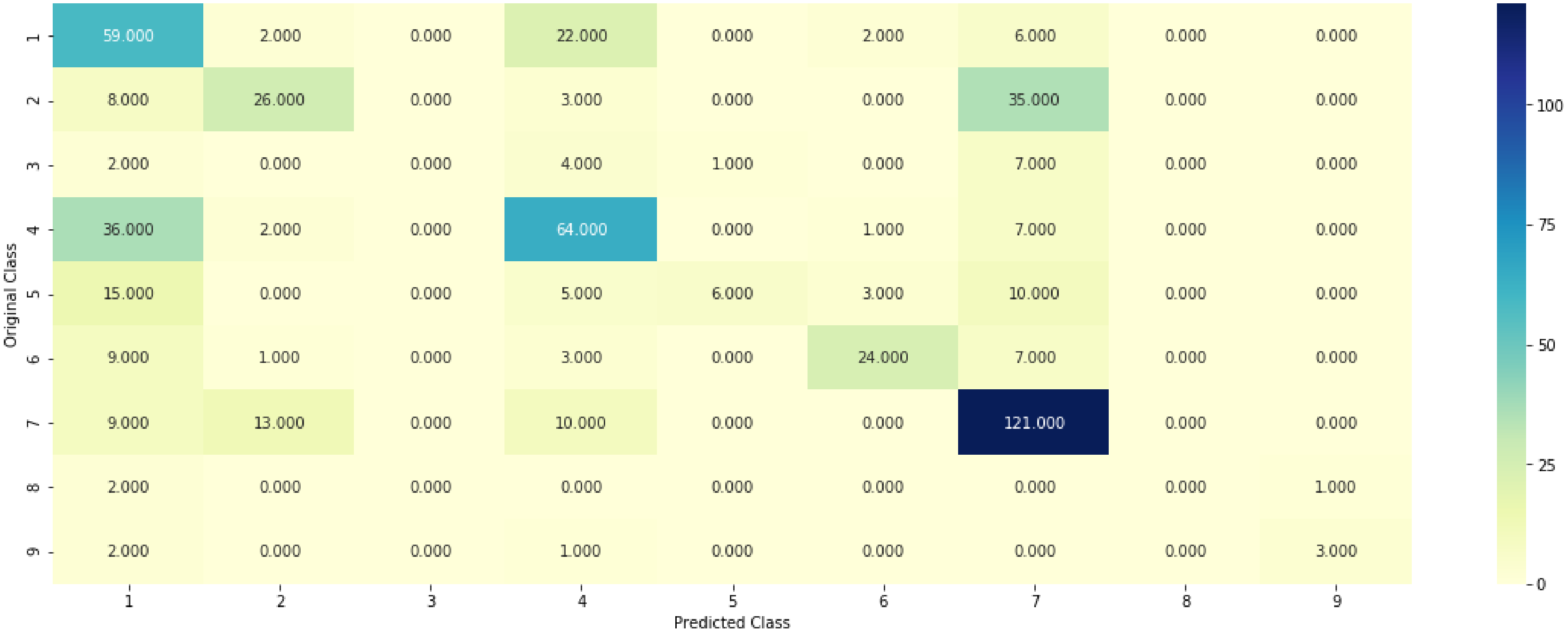
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2233579406586215
for n_estimators = 100 and max depth =  10
Log Loss : 1.2353843129471274
for n_estimators = 200 and max depth =  5
Log Loss : 1.2153609429924064
for n_estimators = 200 and max depth =  10
Log Loss : 1.2173018799045265
for n_estimators = 500 and max depth =  5
Log Loss : 1.196915115320543
for n_estimators = 500 and max depth =  10
Log Loss : 1.2188729901144488
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1933722852169342
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2213142366978234
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1917623602865268
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2215117014625336
For values of best estimator =  2000 The train log loss is: 0.8534874171416896
For values of best estimator =  2000 The cross validation log loss is: 1.1917623602865266
For values of best estimator =  2000 The test log loss is: 1.1785959633194643
```

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

In [156]:

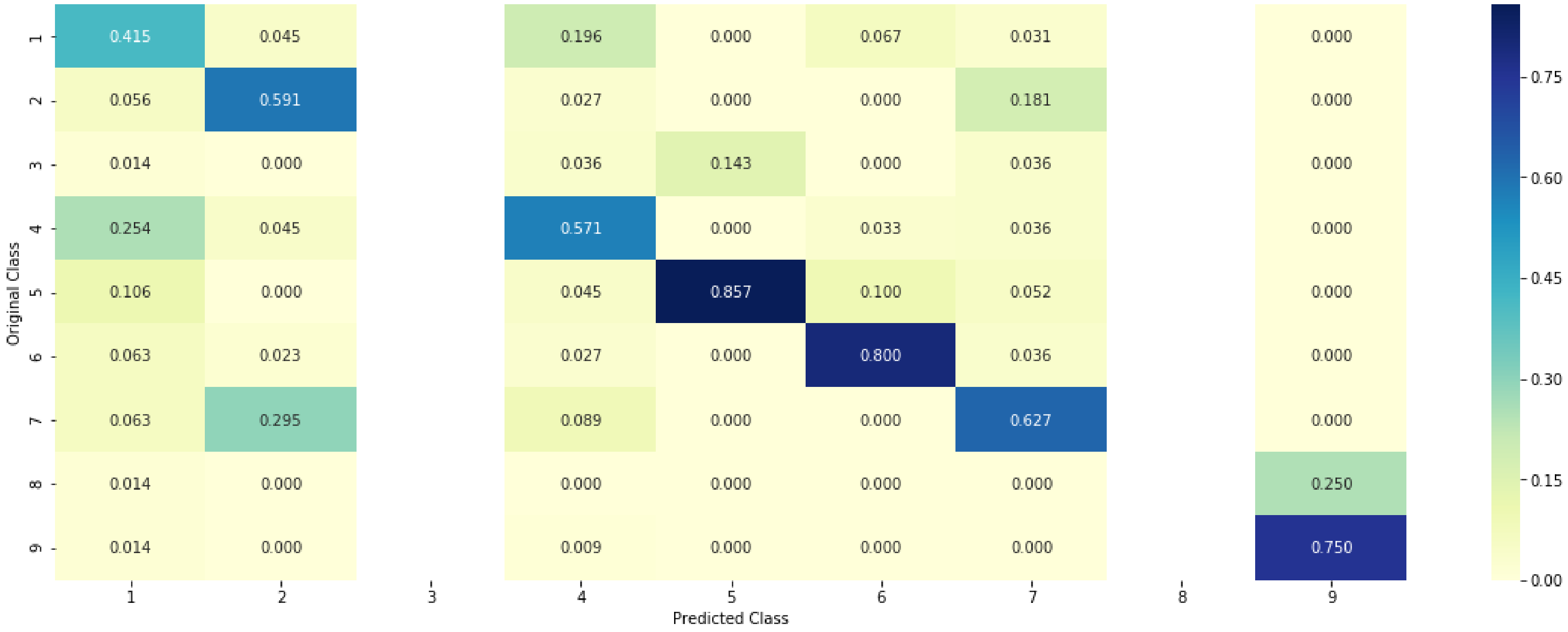
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.1917623602865268
Number of mis-classified points : 0.43045112781954886
-------------------- Confusion matrix --------------------
```



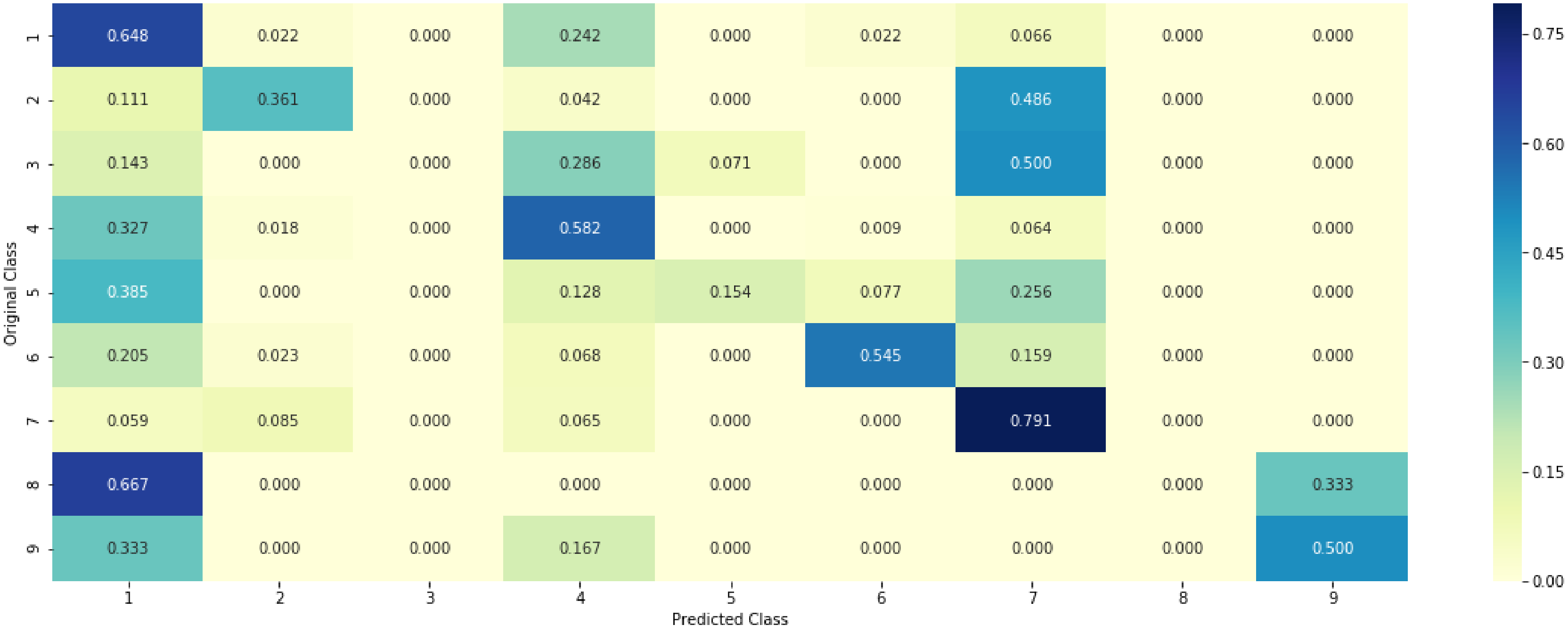
Original Class	1	2	3	4	5	6	7	8	9
1	59.000	2.000	0.000	22.000	0.000	2.000	6.000	0.000	0.000
2	8.000	26.000	0.000	3.000	0.000	0.000	35.000	0.000	0.000
3	2.000	0.000	0.000	4.000	1.000	0.000	7.000	0.000	0.000
4	36.000	2.000	0.000	64.000	0.000	1.000	7.000	0.000	0.000
5	15.000	0.000	0.000	5.000	6.000	3.000	10.000	0.000	0.000
6	9.000	1.000	0.000	3.000	0.000	24.000	7.000	0.000	0.000
7	9.000	13.000	0.000	10.000	0.000	0.000	121.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	2.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	3.000

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



	1	2	3	4	5	6	7	8	9
1	0.415	0.045		0.196	0.000	0.067	0.031		0.000
2	0.056	0.591		0.027	0.000	0.000	0.181		0.000
3	0.014	0.000		0.036	0.143	0.000	0.036		0.000
4	0.254	0.045		0.571	0.000	0.033	0.036		0.000
5	0.106	0.000		0.045	0.857	0.100	0.052		0.000
6	0.063	0.023		0.027	0.000	0.800	0.036		0.000
7	0.063	0.295		0.089	0.000	0.000	0.627		0.000
8	0.014	0.000		0.000	0.000	0.000	0.000		0.250
9	0.014	0.000		0.009	0.000	0.000	0.000		0.750

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.648	0.022	0.000	0.242	0.000	0.022	0.066	0.000	0.000
2	0.111	0.361	0.000	0.042	0.000	0.000	0.486	0.000	0.000
3	0.143	0.000	0.000	0.286	0.071	0.000	0.500	0.000	0.000
4	0.327	0.018	0.000	0.582	0.000	0.009	0.064	0.000	0.000
5	0.385	0.000	0.000	0.128	0.154	0.077	0.256	0.000	0.000
6	0.205	0.023	0.000	0.068	0.000	0.545	0.159	0.000	0.000
7	0.059	0.085	0.000	0.065	0.000	0.000	0.791	0.000	0.000
8	0.667	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333
9	0.333	0.000	0.000	0.167	0.000	0.000	0.000	0.000	0.500

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [157]:

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

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3086 0.1808 0.0238 0.1798 0.0756 0.0861 0.1257 0.01   0.0097]]
Actual Class : 1
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
3 Text feature [activation] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
15 Text feature [protein] present in test data point [True]
18 Text feature [signaling] present in test data point [True]
20 Text feature [pten] present in test data point [True]
22 Text feature [deleterious] present in test data point [True]
25 Text feature [expression] present in test data point [True]
26 Text feature [therapy] present in test data point [True]
28 Text feature [treated] present in test data point [True]
29 Text feature [receptor] present in test data point [True]
31 Text feature [variants] present in test data point [True]
33 Text feature [cells] present in test data point [True]
37 Text feature [growth] present in test data point [True]
38 Text feature [cell] present in test data point [True]
42 Text feature [proteins] present in test data point [True]
44 Text feature [inhibited] present in test data point [True]
48 Text feature [repair] present in test data point [True]
52 Text feature [ovarian] present in test data point [True]
53 Text feature [resistance] present in test data point [True]
54 Text feature [dna] present in test data point [True]
59 Text feature [inhibition] present in test data point [True]
67 Text feature [damage] present in test data point [True]
78 Text feature [patients] present in test data point [True]
80 Text feature [response] present in test data point [True]
91 Text feature [proliferation] present in test data point [True]
94 Text feature [binding] present in test data point [True]
97 Text feature [stimulation] present in test data point [True]
Out of the top  100  features  29 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [158]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2763 0.0829 0.0214 0.4084 0.0684 0.0636 0.0588 0.0052 0.015 ]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
5 Text feature [missense] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
15 Text feature [protein] present in test data point [True]
25 Text feature [expression] present in test data point [True]
31 Text feature [variants] present in test data point [True]
33 Text feature [cells] present in test data point [True]
35 Text feature [functional] present in test data point [True]
38 Text feature [cell] present in test data point [True]
54 Text feature [dna] present in test data point [True]
60 Text feature [clinical] present in test data point [True]
61 Text feature [drug] present in test data point [True]
69 Text feature [expressing] present in test data point [True]
78 Text feature [patients] present in test data point [True]
81 Text feature [mammalian] present in test data point [True]
83 Text feature [information] present in test data point [True]
88 Text feature [affected] present in test data point [True]
94 Text feature [binding] present in test data point [True]
96 Text feature [expected] present in test data point [True]
Out of the top  100  features  20 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [159]:

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

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

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

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
```

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


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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1194163964355197
for n_estimators = 10 and max depth =  3
Log Loss : 1.7972466519537873
for n_estimators = 10 and max depth =  5
Log Loss : 1.6450996666246311
for n_estimators = 10 and max depth =  10
Log Loss : 1.6533476377963305
for n_estimators = 50 and max depth =  2
Log Loss : 1.6526156770316953
for n_estimators = 50 and max depth =  3
Log Loss : 1.471340879427387
for n_estimators = 50 and max depth =  5
Log Loss : 1.4551958854648934
for n_estimators = 50 and max depth =  10
Log Loss : 1.7096299628324423
for n_estimators = 100 and max depth =  2
Log Loss : 1.5157813772512414
for n_estimators = 100 and max depth =  3
Log Loss : 1.5232884608613027
for n_estimators = 100 and max depth =  5
Log Loss : 1.399528635047226
for n_estimators = 100 and max depth =  10
Log Loss : 1.7262448382978766
for n_estimators = 200 and max depth =  2
Log Loss : 1.6244232740044444
for n_estimators = 200 and max depth =  3
Log Loss : 1.5417481516786928
for n_estimators = 200 and max depth =  5
Log Loss : 1.433870347459857
for n_estimators = 200 and max depth =  10
Log Loss : 1.8121161665912138
for n_estimators = 500 and max depth =  2
Log Loss : 1.698700436677204
for n_estimators = 500 and max depth =  3
Log Loss : 1.5774100759348613
for n_estimators = 500 and max depth =  5
Log Loss : 1.448708835792067
for n_estimators = 500 and max depth =  10
Log Loss : 1.8451319798385446
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6586743440035843
for n_estimators = 1000 and max depth =  3
```

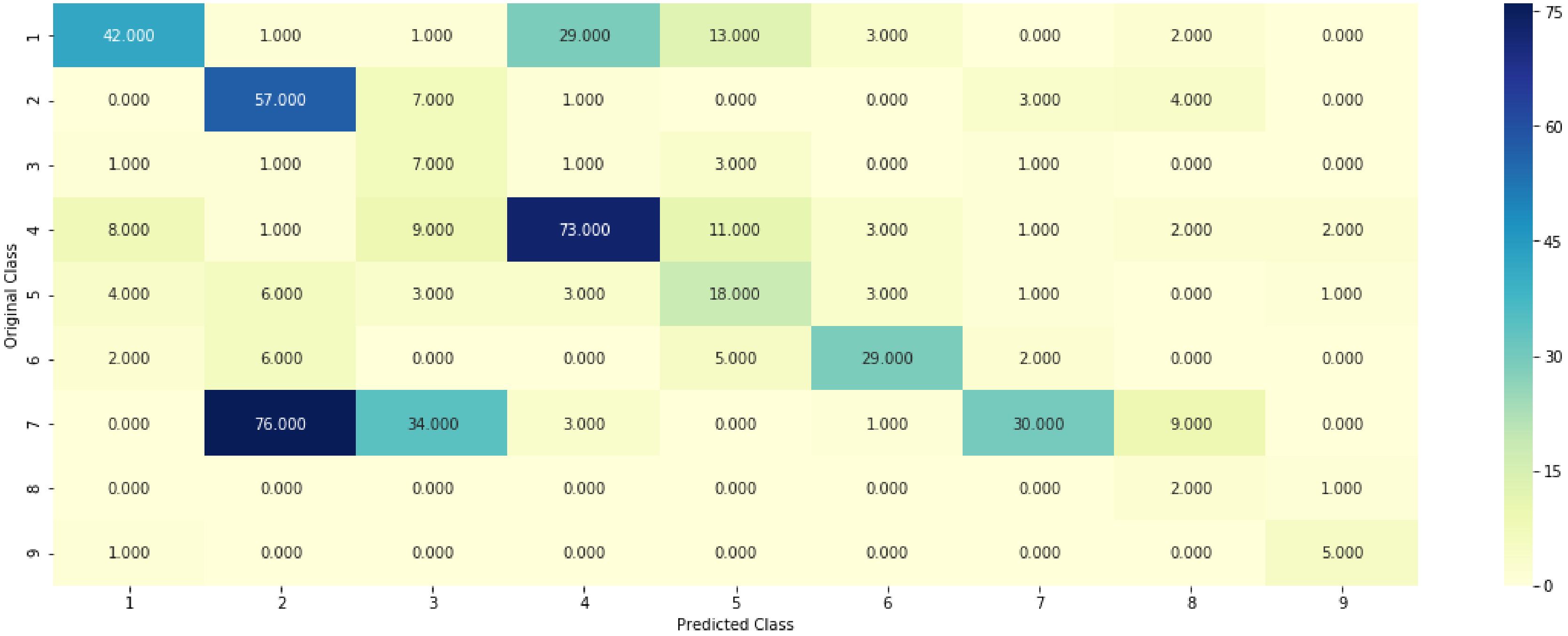
```
Log Loss : 1.5937127690413737
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4313536963978786
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8393549092127728
For values of best alpha =  100 The train log loss is: 0.0597631982773824 95
For values of best alpha =  100 The cross validation log loss is: 1.399528635047226
For values of best alpha =  100 The test log loss is: 1.429569874040215
```

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

In [160]:

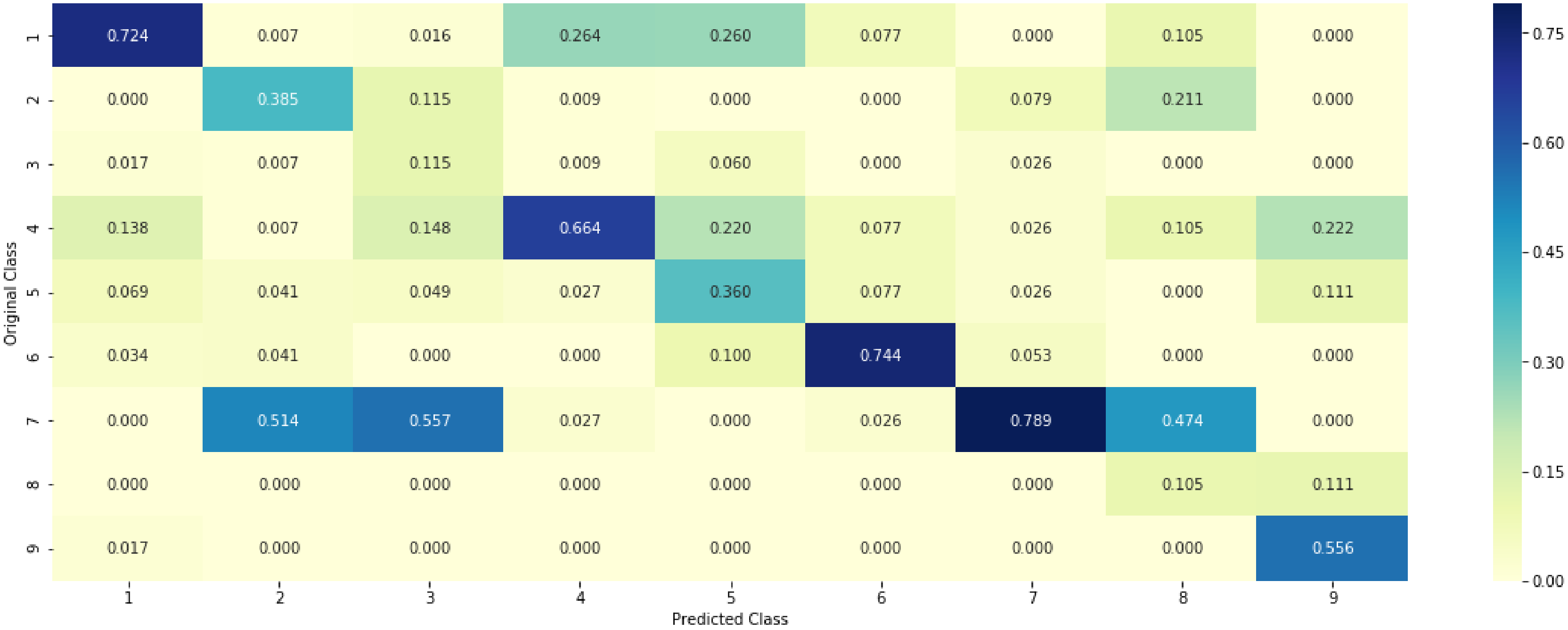
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.399528635047226
Number of mis-classified points : 0.5056390977443609
-------------------- Confusion matrix --------------------
```



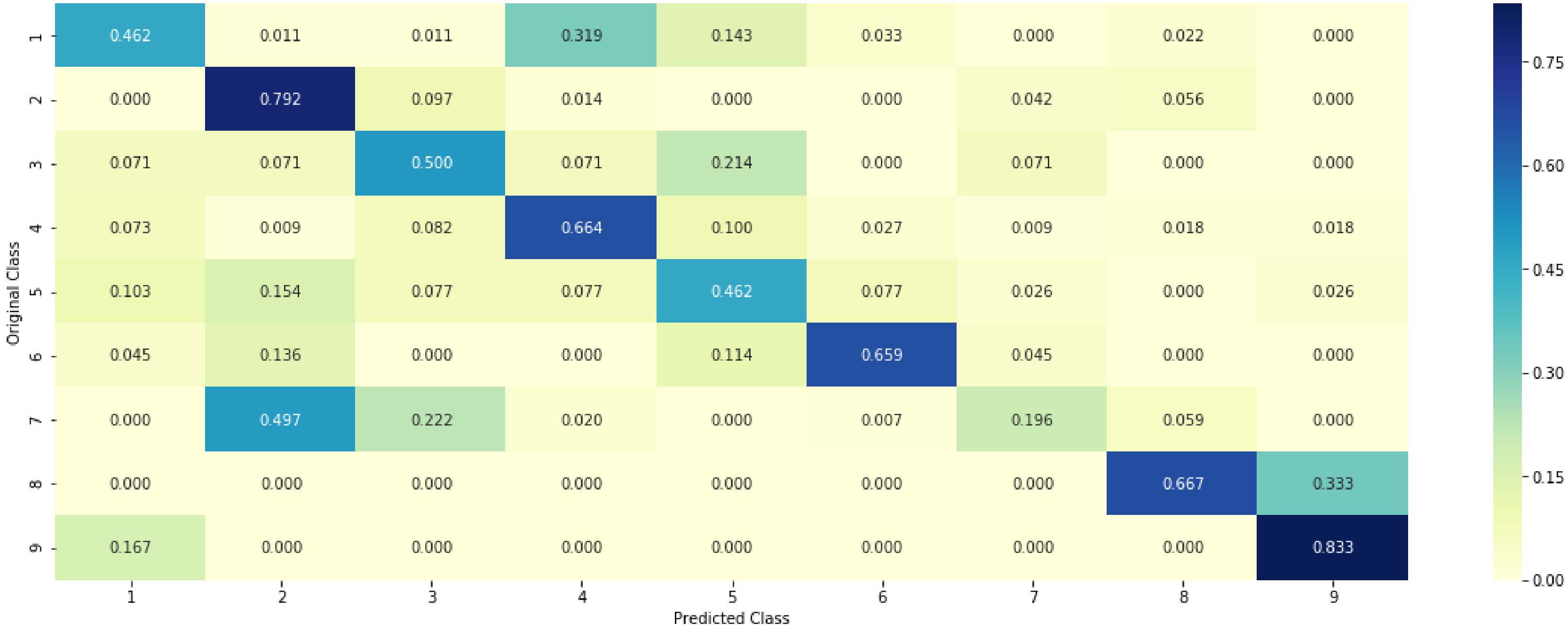
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	42.000	1.000	1.000	29.000	13.000	3.000	0.000	2.000	0.000
2	0.000	57.000	7.000	1.000	0.000	0.000	3.000	4.000	0.000
3	1.000	1.000	7.000	1.000	3.000	0.000	1.000	0.000	0.000
4	8.000	1.000	9.000	73.000	11.000	3.000	1.000	2.000	2.000
5	4.000	6.000	3.000	3.000	18.000	3.000	1.000	0.000	1.000
6	2.000	6.000	0.000	0.000	5.000	29.000	2.000	0.000	0.000
7	0.000	76.000	34.000	3.000	0.000	1.000	30.000	9.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.724	0.007	0.016	0.264	0.260	0.077	0.000	0.105	0.000
2	0.000	0.385	0.115	0.009	0.000	0.000	0.079	0.211	0.000
3	0.017	0.007	0.115	0.009	0.060	0.000	0.026	0.000	0.000
4	0.138	0.007	0.148	0.664	0.220	0.077	0.026	0.105	0.222
5	0.069	0.041	0.049	0.027	0.360	0.077	0.026	0.000	0.111
6	0.034	0.041	0.000	0.000	0.100	0.744	0.053	0.000	0.000
7	0.000	0.514	0.557	0.027	0.000	0.026	0.789	0.474	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.105	0.111
9	0.017	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.556

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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [161]:

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


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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0909 0.123  0.2529 0.0438 0.0956 0.1046 0.1083 0.1342 0.0468]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene 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 [162]:

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

```
Predicted Class : 8
Predicted Class Probabilities: [[0.0399 0.0469 0.0476 0.028  0.0242 0.0235 0.01   0.4924 0.2876]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene 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 [163]:

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

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

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


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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_leaf_nodes=None, min_impurity_de
crease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_star
t=False,
# class_weight=None)

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

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

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding
))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
```

```
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True
)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict
_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

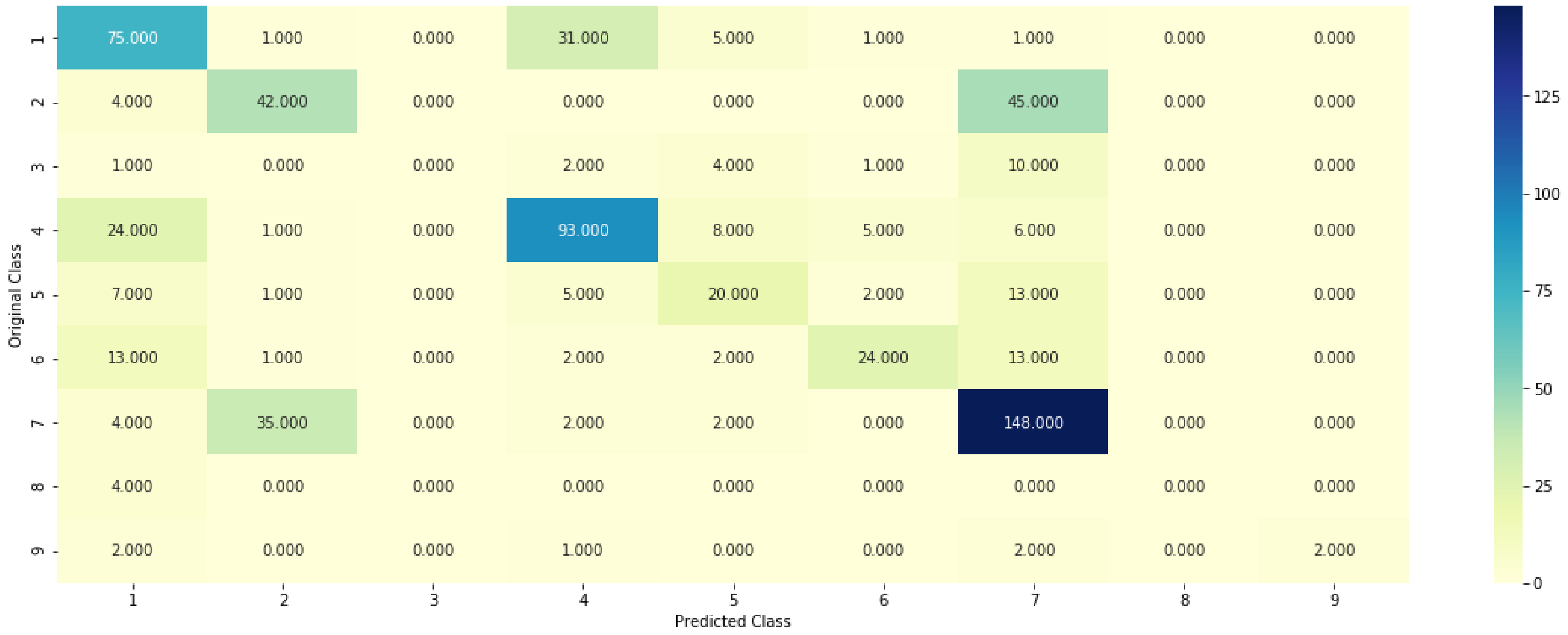
```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.84
Naive Bayes : Log Loss: 1.18
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.038
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.520
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.143
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.171
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.256
```

4.7.2 testing the model with the best hyper parameters

In [164]:

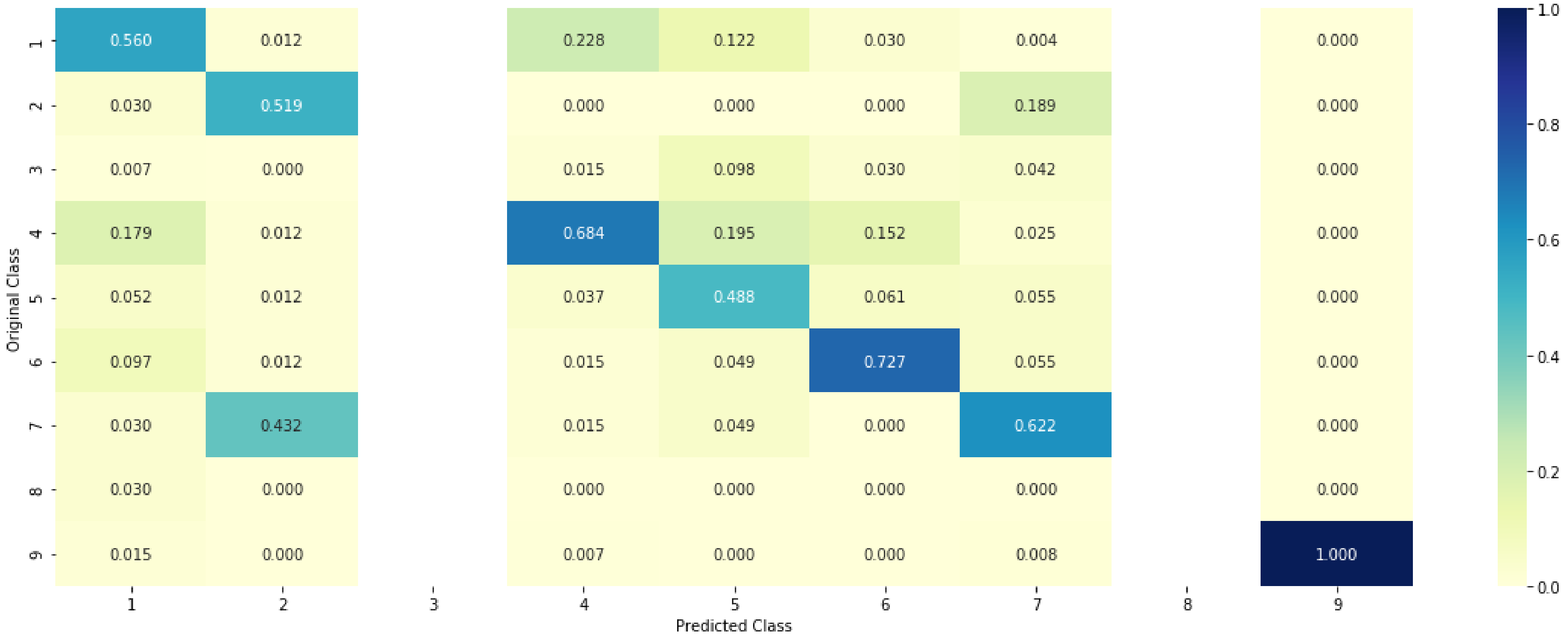
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

```
Log loss (train) on the stacking classifier : 0.8115530653461235
Log loss (CV) on the stacking classifier : 1.1434227675004534
Log loss (test) on the stacking classifier : 1.1539710376323749
Number of missclassified point : 0.3924812030075188
-------------------- Confusion matrix --------------------
```



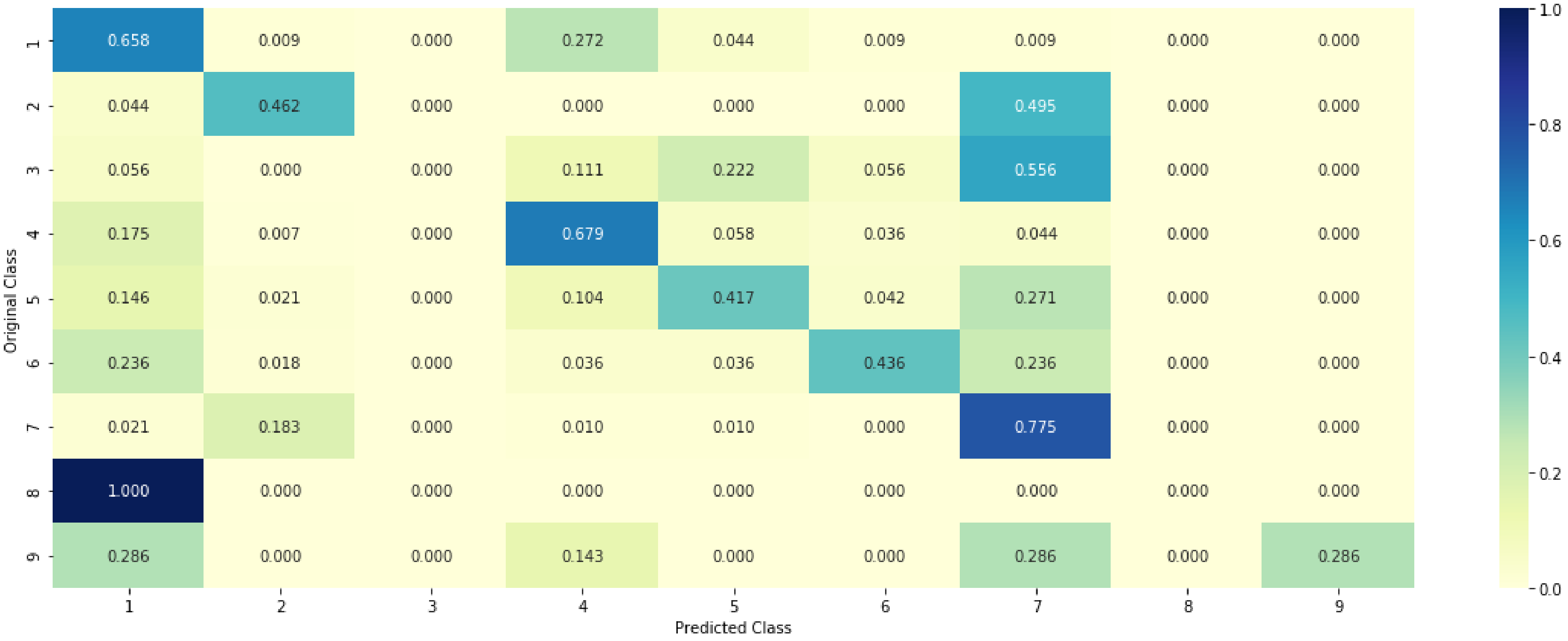
	1	2	3	4	5	6	7	8	9
1	75.000	1.000	0.000	31.000	5.000	1.000	1.000	0.000	0.000
2	4.000	42.000	0.000	0.000	0.000	0.000	45.000	0.000	0.000
3	1.000	0.000	0.000	2.000	4.000	1.000	10.000	0.000	0.000
4	24.000	1.000	0.000	93.000	8.000	5.000	6.000	0.000	0.000
5	7.000	1.000	0.000	5.000	20.000	2.000	13.000	0.000	0.000
6	13.000	1.000	0.000	2.000	2.000	24.000	13.000	0.000	0.000
7	4.000	35.000	0.000	2.000	2.000	0.000	148.000	0.000	0.000
8	4.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	2.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.560	0.012		0.228	0.122	0.030	0.004		0.000
2	0.030	0.519		0.000	0.000	0.000	0.189		0.000
3	0.007	0.000		0.015	0.098	0.030	0.042		0.000
4	0.179	0.012		0.684	0.195	0.152	0.025		0.000
5	0.052	0.012		0.037	0.488	0.061	0.055		0.000
6	0.097	0.012		0.015	0.049	0.727	0.055		0.000
7	0.030	0.432		0.015	0.049	0.000	0.622		0.000
8	0.030	0.000		0.000	0.000	0.000	0.000		0.000
9	0.015	0.000		0.007	0.000	0.000	0.008		1.000

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



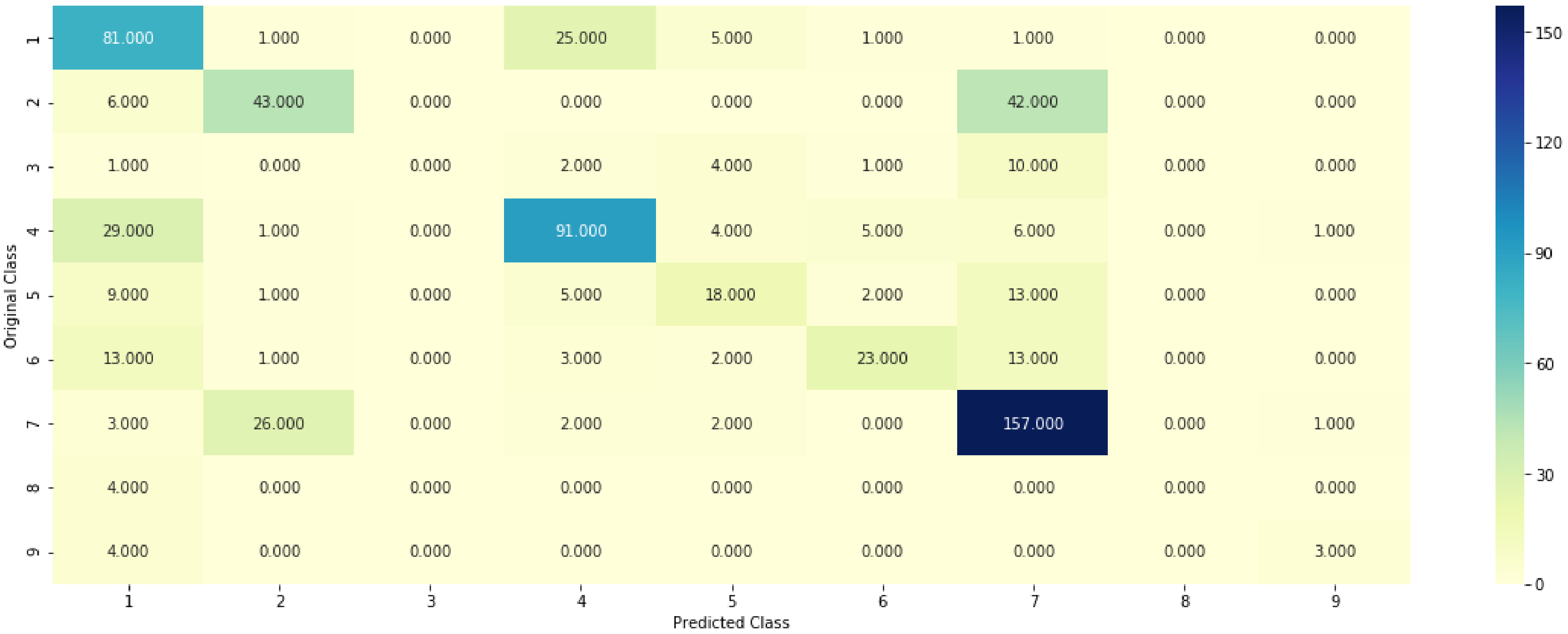
	1	2	3	4	5	6	7	8	9
1	0.658	0.009	0.000	0.272	0.044	0.009	0.009	0.000	0.000
2	0.044	0.462	0.000	0.000	0.000	0.000	0.495	0.000	0.000
3	0.056	0.000	0.000	0.111	0.222	0.056	0.556	0.000	0.000
4	0.175	0.007	0.000	0.679	0.058	0.036	0.044	0.000	0.000
5	0.146	0.021	0.000	0.104	0.417	0.042	0.271	0.000	0.000
6	0.236	0.018	0.000	0.036	0.036	0.436	0.236	0.000	0.000
7	0.021	0.183	0.000	0.010	0.010	0.000	0.775	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.286	0.000	0.000	0.143	0.000	0.000	0.286	0.000	0.286

4.7.3 Maximum Voting classifier

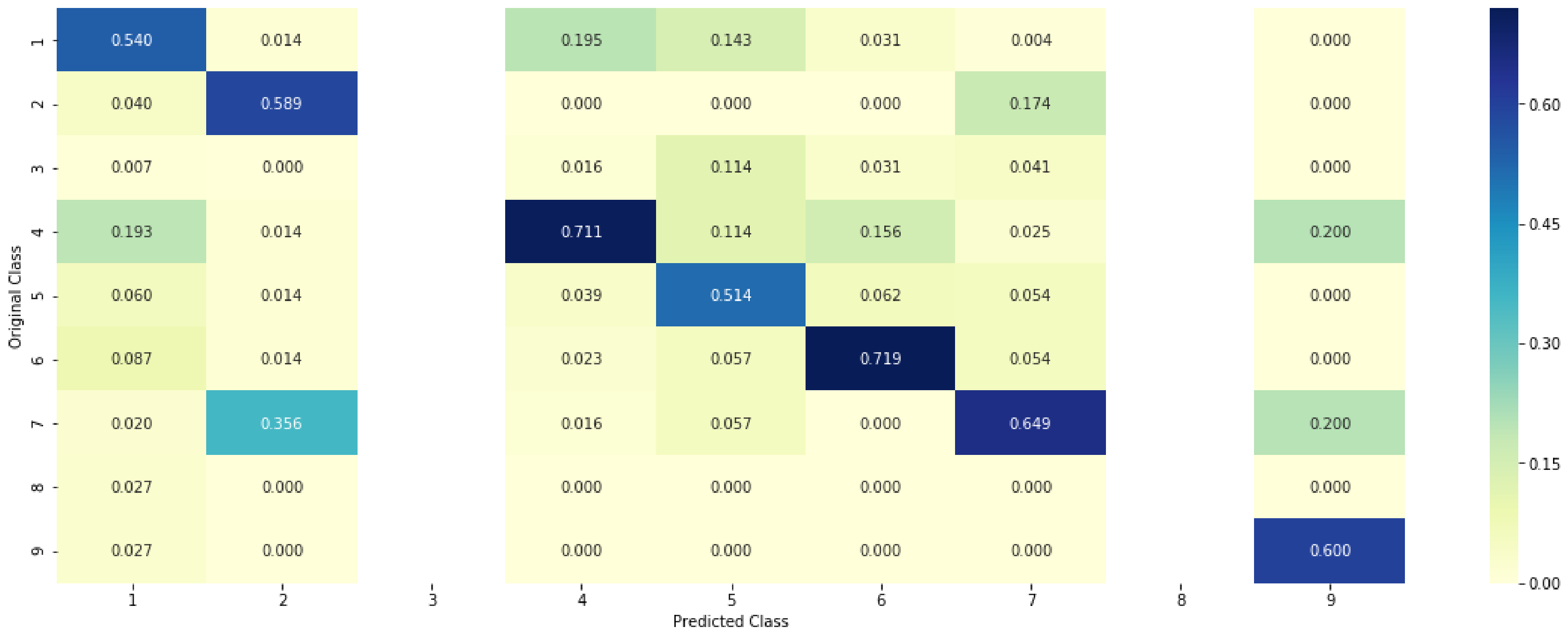
In [165]:

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

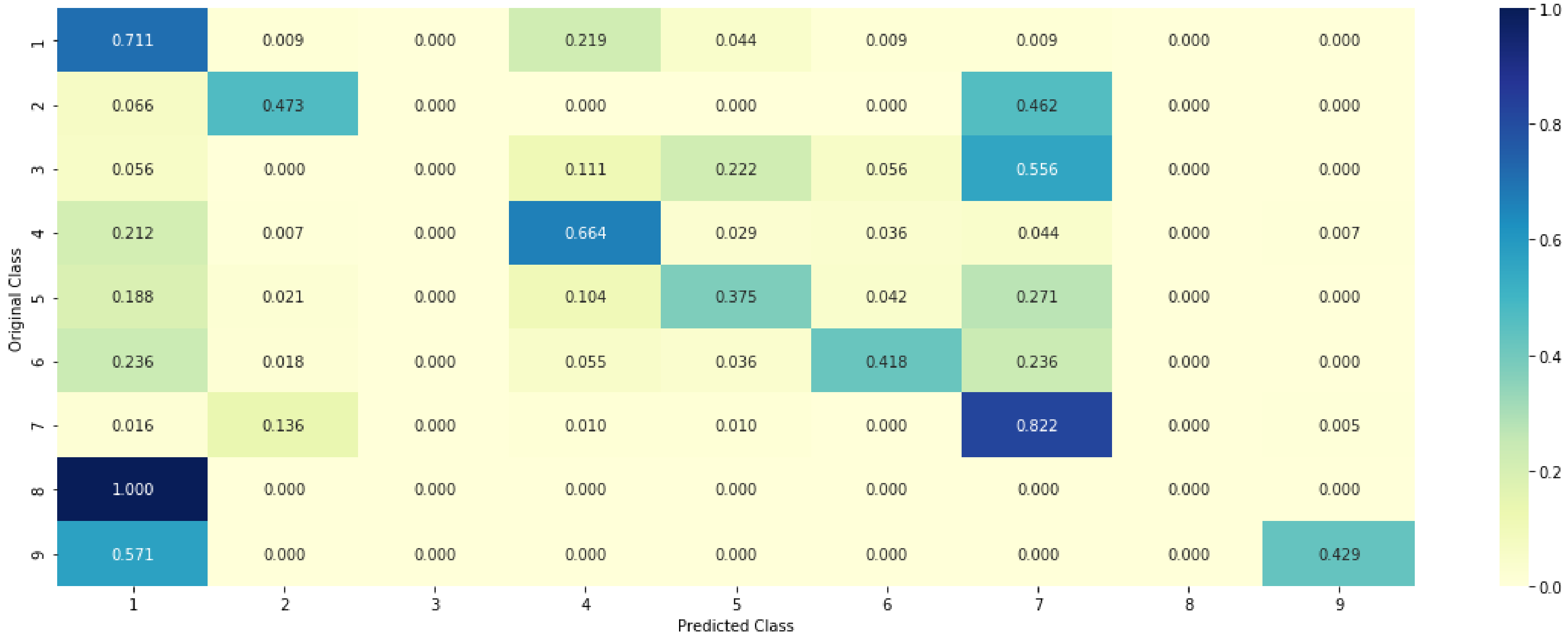
```
Log loss (train) on the VotingClassifier : 0.9553953825140634
Log loss (CV) on the VotingClassifier : 1.195371909243399
Log loss (test) on the VotingClassifier : 1.2063064818034483
Number of missclassified point : 0.37443609022556390
-------------------- Confusion matrix --------------------
```



```
-------------------- 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

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

In [184]:

```
train_df.head()
```

Out[184]:

	ID	Gene	Variation	Class	TEXT
3103	3103	RAC1	G12V	7	members ras superfamily small guanosine tripho...
3051	3051	KIT	V654A	7	gist patients develop clinical resistance kit ...
953	953	PDGFRB	V665A	4	penttinen syndrome distinctive disorder charac...
1723	1723	APC	I1307K	1	classical familial adenomatous polyposis fap h...
1337	1337	KMT2A	Amplification	6	human chromosome 11q23 translocations disrupti...

In [167]:

```
#onehot encoding of gene feature
gene_vec = CountVectorizer(ngram_range=(1,2))
train_gene_onehot = gene_vec.fit_transform(train_df['Gene'])
test_gene_onehot = gene_vec.transform(test_df['Gene'])
cv_gene_onehot = gene_vec.transform(cv_df['Gene'])
```

In [168]:

```
#onehot encoding of variation feature
Variation_vec = CountVectorizer(ngram_range=(1,2))
train_Variation_onehot = Variation_vec.fit_transform(train_df['Variation'])
test_Variation_onehot = Variation_vec.transform(test_df['Variation'])
cv_Variation_onehot = Variation_vec.transform(cv_df['Variation'])
```

In [169]:

```
#onehot encoding of text feature
text_vec = CountVectorizer(ngram_range=(1,2))
train_text_onehot = text_vec.fit_transform(train_df['TEXT'])
test_text_onehot = text_vec.transform(test_df['TEXT'])
cv_text_onehot = text_vec.transform(cv_df['TEXT'])
```

In [170]:

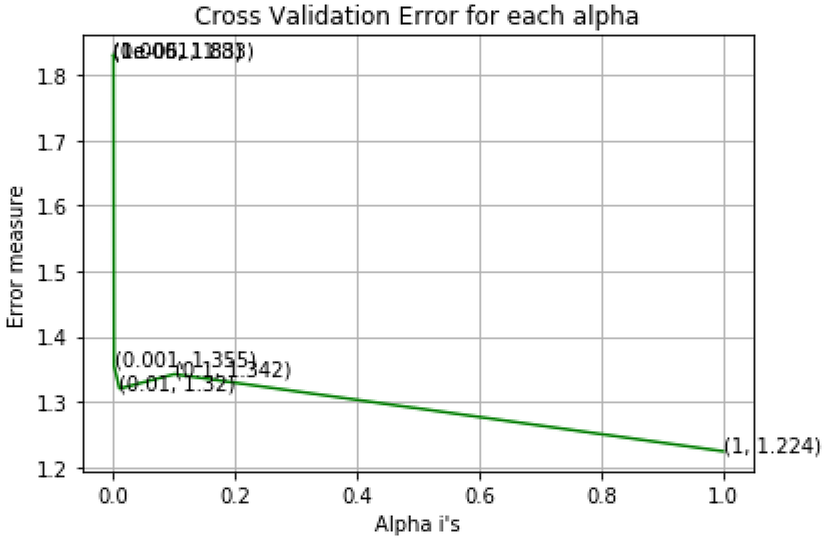
```
train_gene_var_onehot = hstack([train_gene_onehot,train_Variation_onehot])
test_gene_var_onehot = hstack([test_gene_onehot,test_Variation_onehot])
cv_gene_var_onehot = hstack([cv_gene_onehot,cv_Variation_onehot])

train_x_onehot = hstack([train_gene_var_onehot,train_text_onehot]).tocsr()
test_x_onehot = hstack([test_gene_var_onehot,test_text_onehot]).tocsr()
cv_x_onehot = hstack([cv_gene_var_onehot,cv_text_onehot]).tocsr()
```

In [ ]:

In [171]:

```python
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(class_weight='balanced',alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehot, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehot, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehot)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_x_onehot)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehot)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehot)
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-06 The log loss is: 1.8304997567764278
For values of alpha =  1e-05 The log loss is: 1.8304997567764278
For values of alpha =  0.0001 The log loss is: 1.8304997567764278
For values of alpha =  0.001 The log loss is: 1.3554907538062044
For values of alpha =  0.01 The log loss is: 1.3201363410588172
For values of alpha =  0.1 The log loss is: 1.3421741511696772
For values of alpha =  1 The log loss is: 1.2241363441089368
```

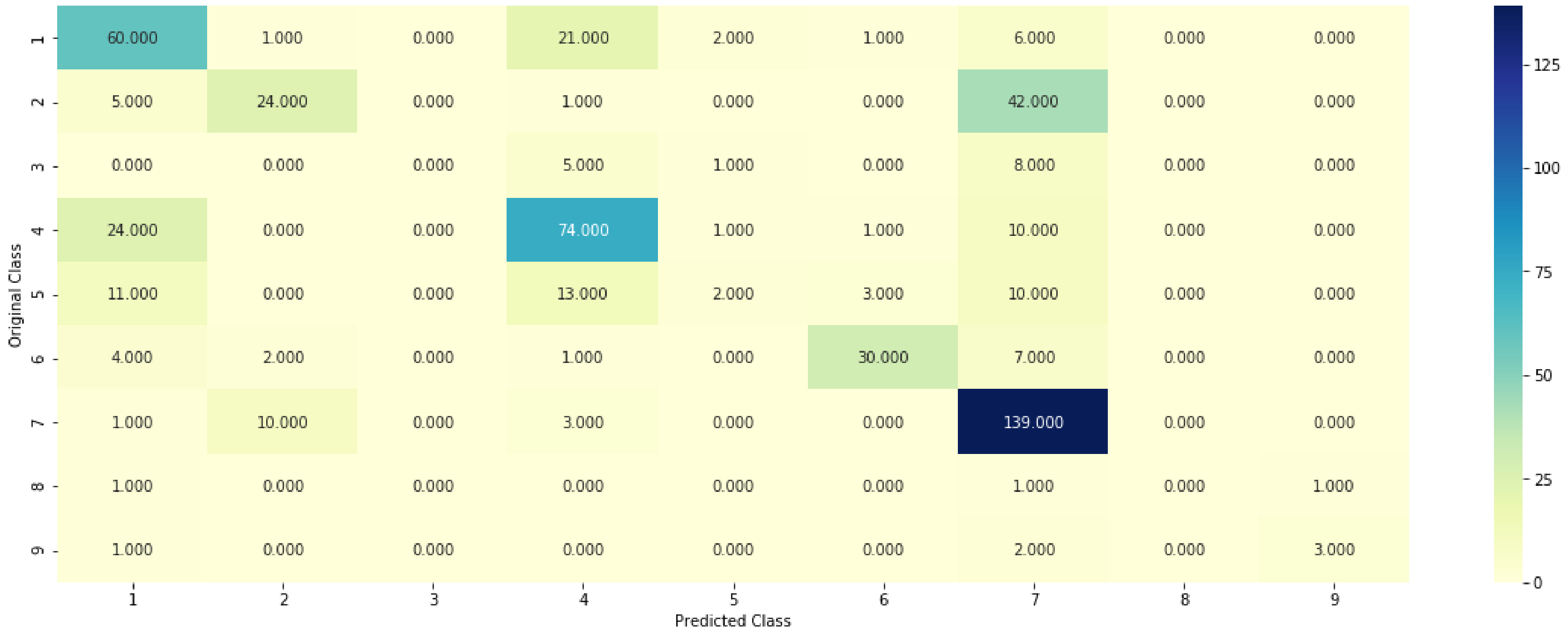


```
For values of best alpha =  1 The train log loss is: 0.9009436536681545
For values of best alpha =  1 The cross validation log loss is: 1.229977670019873
For values of best alpha =  1 The test log loss is: 1.163543495060012
```

In [172]:

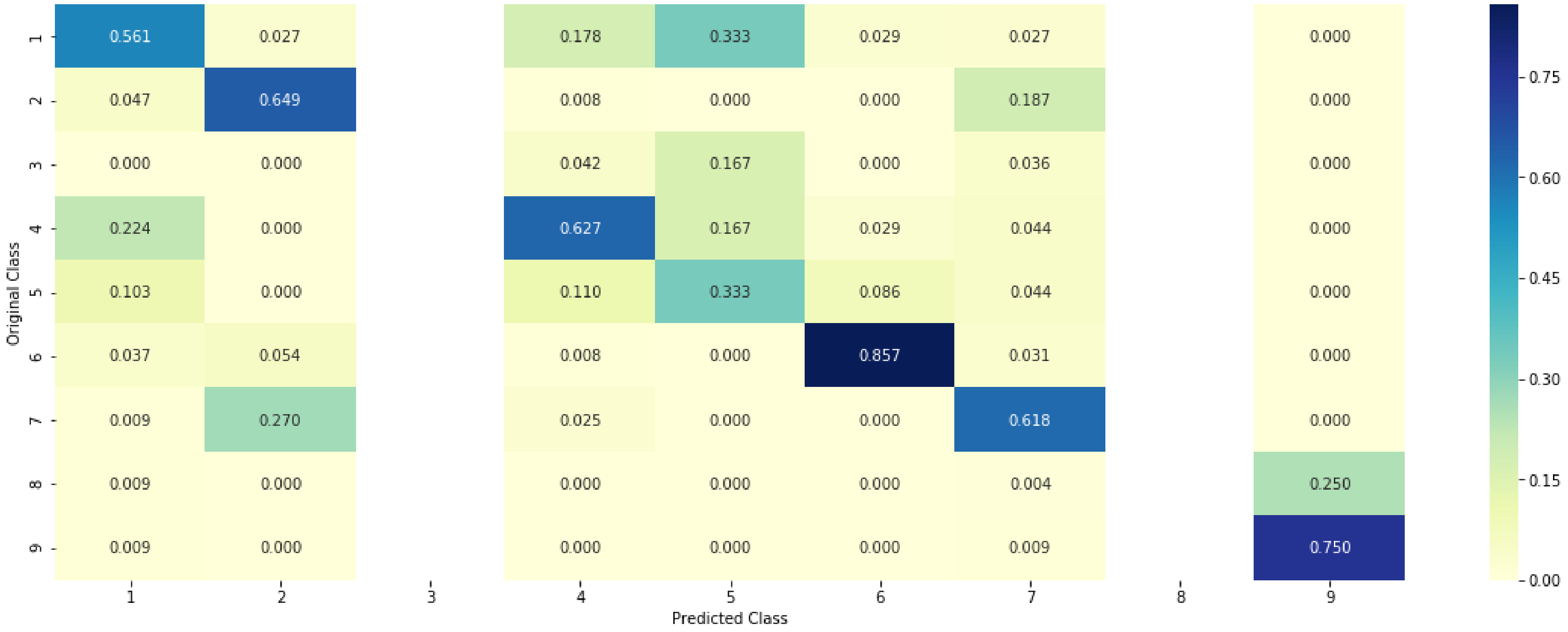
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
42)
predict_and_plot_confusion_matrix(train_x_onehot, y_train, cv_x_onehot, y_cv, clf)
```

```
Log loss : 1.2241363441089368
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

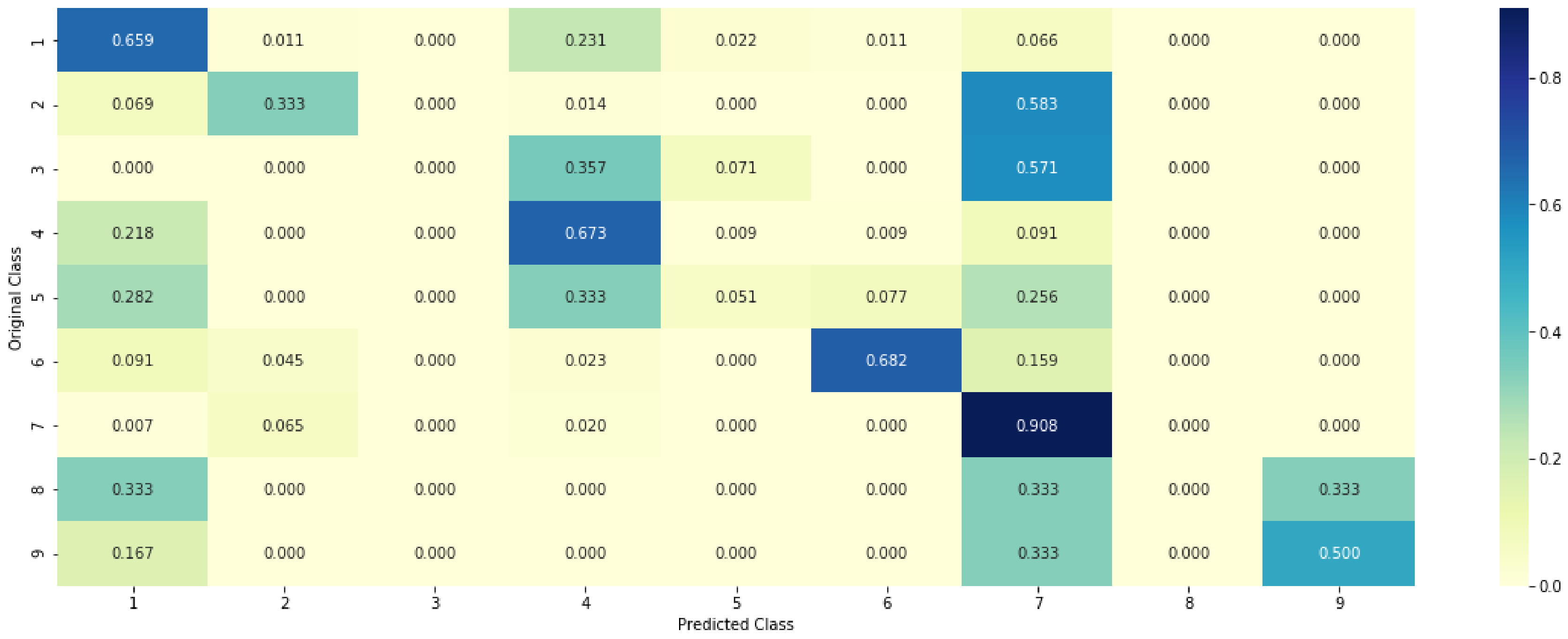


	1	2	3	4	5	6	7	8	9
1	60.000	1.000	0.000	21.000	2.000	1.000	6.000	0.000	0.000
2	5.000	24.000	0.000	1.000	0.000	0.000	42.000	0.000	0.000
3	0.000	0.000	0.000	5.000	1.000	0.000	8.000	0.000	0.000
4	24.000	0.000	0.000	74.000	1.000	1.000	10.000	0.000	0.000
5	11.000	0.000	0.000	13.000	2.000	3.000	10.000	0.000	0.000
6	4.000	2.000	0.000	1.000	0.000	30.000	7.000	0.000	0.000
7	1.000	10.000	0.000	3.000	0.000	0.000	139.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	3.000

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



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



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

TFIDF for text

ngram_range = (1,2)

```
In [173]: #onehot encoding of gene feature
          gene_vec = TfidfVectorizer(ngram_range=(1,2))
          train_gene_onehot = gene_vec.fit_transform(train_df['Gene'])
          test_gene_onehot = gene_vec.transform(test_df['Gene'])
          cv_gene_onehot = gene_vec.transform(cv_df['Gene'])
```

In [174]:

```
#onehot encoding of variation feature
Variation_vec = TfidfVectorizer(ngram_range=(1,2))
train_Variation_onehot = Variation_vec.fit_transform(train_df['Variation'])
test_Variation_onehot = Variation_vec.transform(test_df['Variation'])
cv_Variation_onehot = Variation_vec.transform(cv_df['Variation'])
```

In [175]:

```
#onehot encoding of text feature
text_vec = TfidfVectorizer(ngram_range=(1,2))
train_text_onehot = text_vec.fit_transform(train_df['TEXT'])
test_text_onehot = text_vec.transform(test_df['TEXT'])
cv_text_onehot = text_vec.transform(cv_df['TEXT'])
```

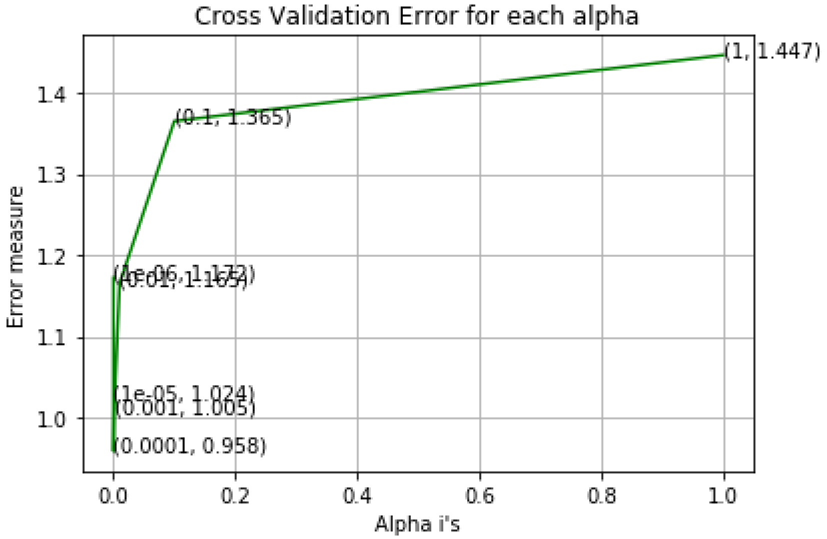
In [176]:

```
train_gene_var_onehot = hstack([train_gene_onehot,train_Variation_onehot])
test_gene_var_onehot = hstack([test_gene_onehot,test_Variation_onehot])
cv_gene_var_onehot = hstack([cv_gene_onehot,cv_Variation_onehot])

train_x_onehot = hstack([train_gene_var_onehot,train_text_onehot]).tocsr()
test_x_onehot = hstack([test_gene_var_onehot,test_text_onehot]).tocsr()
cv_x_onehot = hstack([cv_gene_var_onehot,cv_text_onehot]).tocsr()
```

In [177]:

```
alpha = [10 ** x for x in range(-6, 1)] # hyperparam for SGD classifier.


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(class_weight='balanced',alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehot, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehot, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehot)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=
1e-15))

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


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

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

```
For values of alpha =  1e-06 The log loss is: 1.1724205392614786
For values of alpha =  1e-05 The log loss is: 1.023540199402943
For values of alpha =  0.0001 The log loss is: 0.9582016820827511
For values of alpha =  0.001 The log loss is: 1.0054637105999793
For values of alpha =  0.01 The log loss is: 1.164861925668012
For values of alpha =  0.1 The log loss is: 1.3651810480405386
For values of alpha =  1 The log loss is: 1.4466225174263438
```

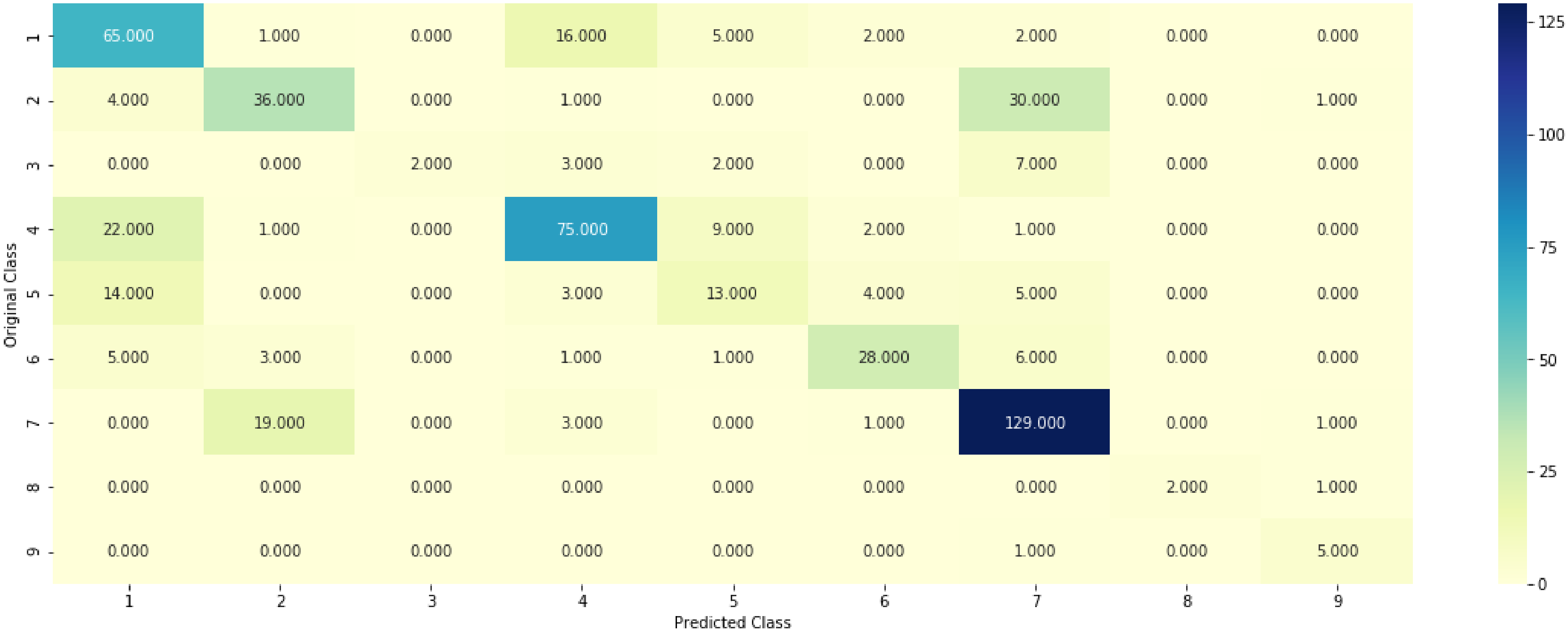


```
For values of best alpha =  0.0001 The train log loss is: 0.40054994521240567
For values of best alpha =  0.0001 The cross validation log loss is: 0.9585738541617781
For values of best alpha =  0.0001 The test log loss is: 0.9545922597671851
```

In [178]:

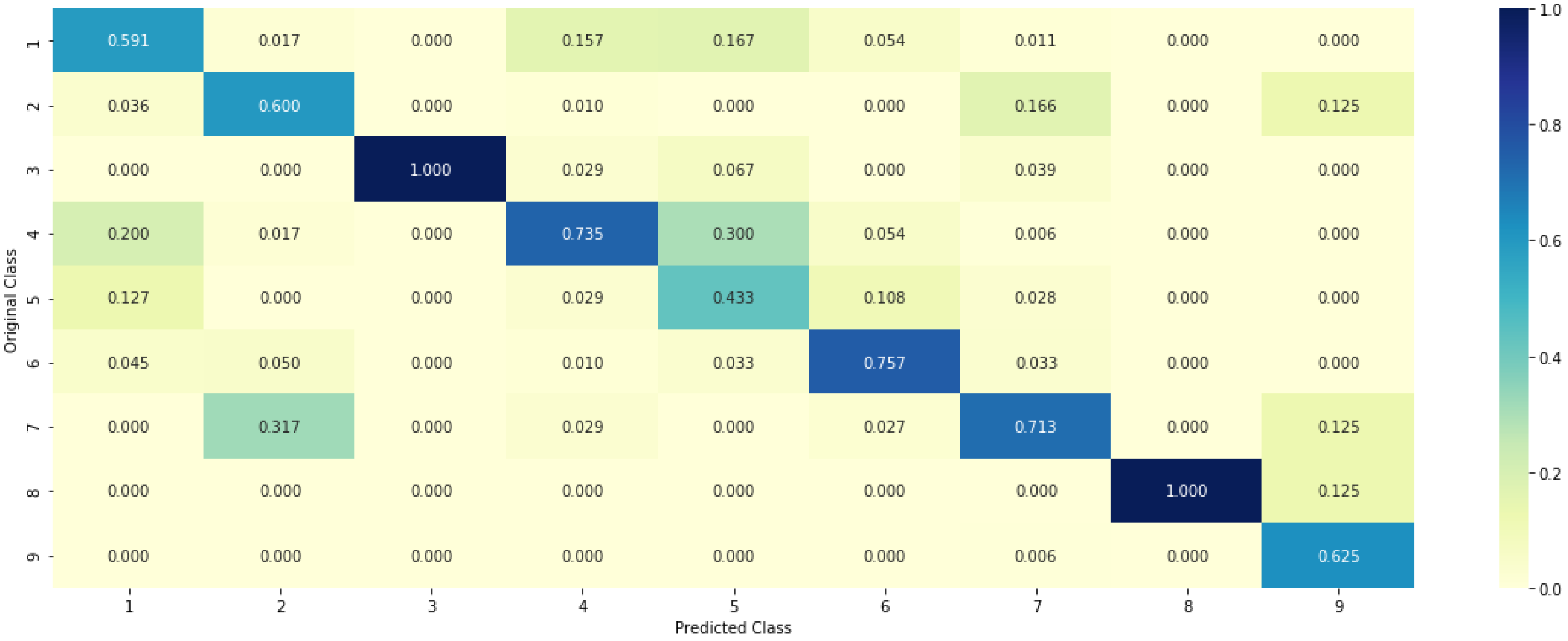
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
42)
predict_and_plot_confusion_matrix(train_x_onehot, y_train, cv_x_onehot, y_cv, clf)
```

```
Log loss : 0.9582016820827511
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```



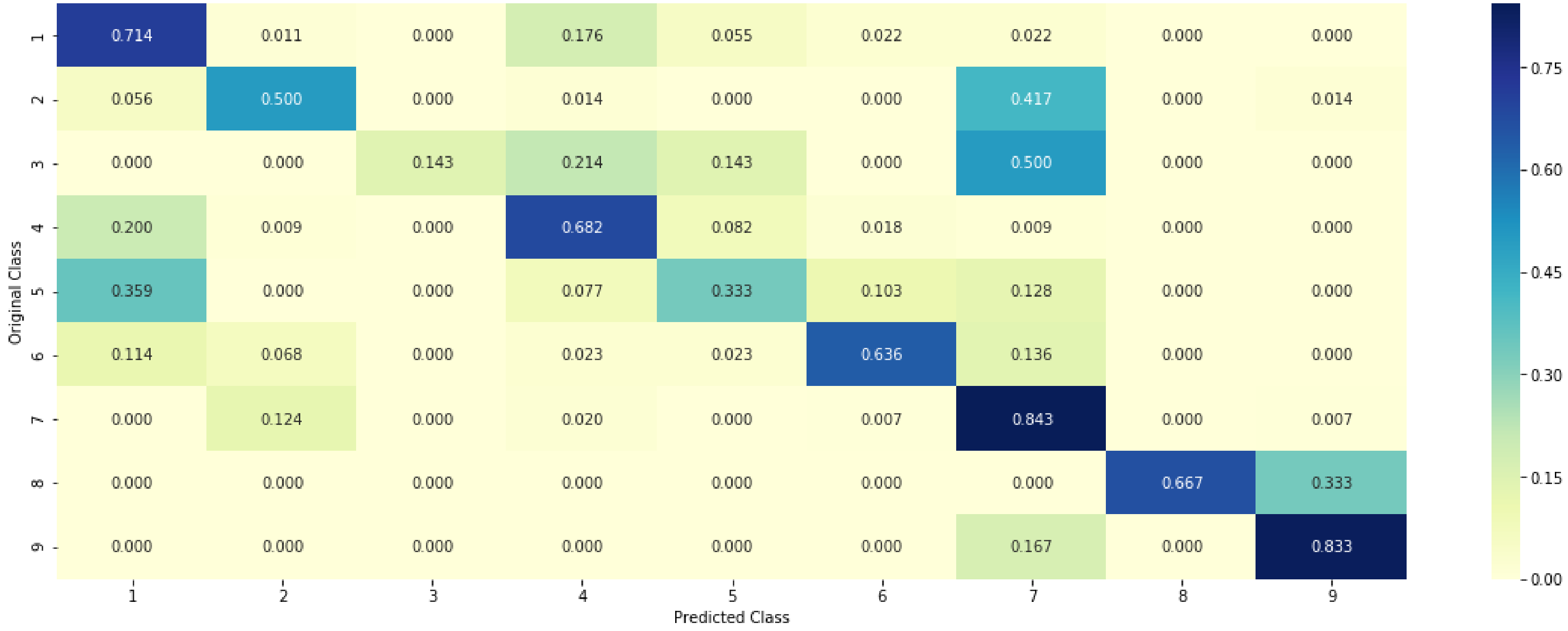
	1	2	3	4	5	6	7	8	9
1	65.000	1.000	0.000	16.000	5.000	2.000	2.000	0.000	0.000
2	4.000	36.000	0.000	1.000	0.000	0.000	30.000	0.000	1.000
3	0.000	0.000	2.000	3.000	2.000	0.000	7.000	0.000	0.000
4	22.000	1.000	0.000	75.000	9.000	2.000	1.000	0.000	0.000
5	14.000	0.000	0.000	3.000	13.000	4.000	5.000	0.000	0.000
6	5.000	3.000	0.000	1.000	1.000	28.000	6.000	0.000	0.000
7	0.000	19.000	0.000	3.000	0.000	1.000	129.000	0.000	1.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.591	0.017	0.000	0.157	0.167	0.054	0.011	0.000	0.000
2	0.036	0.600	0.000	0.010	0.000	0.000	0.166	0.000	0.125
3	0.000	0.000	1.000	0.029	0.067	0.000	0.039	0.000	0.000
4	0.200	0.017	0.000	0.735	0.300	0.054	0.006	0.000	0.000
5	0.127	0.000	0.000	0.029	0.433	0.108	0.028	0.000	0.000
6	0.045	0.050	0.000	0.010	0.033	0.757	0.033	0.000	0.000
7	0.000	0.317	0.000	0.029	0.000	0.027	0.713	0.000	0.125
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.125
9	0.000	0.000	0.000	0.000	0.000	0.000	0.006	0.000	0.625

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



TFIDF for text

ngram_range=(1,4)
max_features=6000

In [179]:
```
#onehot encoding of text feature
text_vec = TfidfVectorizer(ngram_range=(1,4), max_features=6000)
train_text_onehot = text_vec.fit_transform(train_df['TEXT'])
test_text_onehot = text_vec.transform(test_df['TEXT'])
cv_text_onehot = text_vec.transform(cv_df['TEXT'])
```

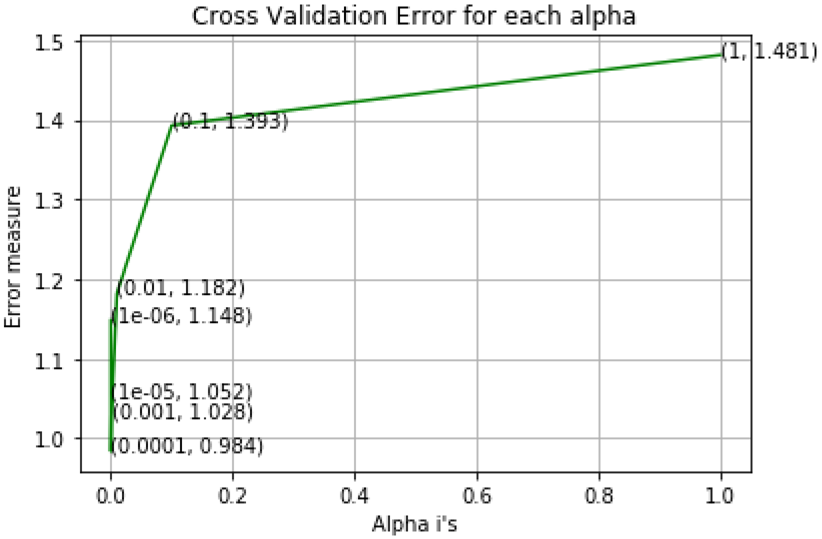
In [180]:

```
train_gene_var_onehot = hstack([train_gene_onehot,train_Variation_onehot])
test_gene_var_onehot = hstack([test_gene_onehot,test_Variation_onehot])
cv_gene_var_onehot = hstack([cv_gene_onehot,cv_Variation_onehot])

train_x_onehot = hstack([train_gene_var_onehot,train_text_onehot]).tocsr()
test_x_onehot = hstack([test_gene_var_onehot,test_text_onehot]).tocsr()
cv_x_onehot = hstack([cv_gene_var_onehot,cv_text_onehot]).tocsr()
```

In [181]:

```
alpha = [10 ** x for x in range(-6, 1)] # hyperparam for SGD classifier.


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(class_weight='balanced',alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehot, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehot, y_train)
    predict_y = sig_clf.predict_proba(cv_x_onehot)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=
1e-15))

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


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

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

```
For values of alpha =  1e-06 The log loss is: 1.1484098178469415
For values of alpha =  1e-05 The log loss is: 1.0515466492542762
For values of alpha =  0.0001 The log loss is: 0.9835840955161868
For values of alpha =  0.001 The log loss is: 1.0277167418825555
For values of alpha =  0.01 The log loss is: 1.1820776966314812
For values of alpha =  0.1 The log loss is: 1.3928586423674376
For values of alpha =  1 The log loss is: 1.481426851159669
```

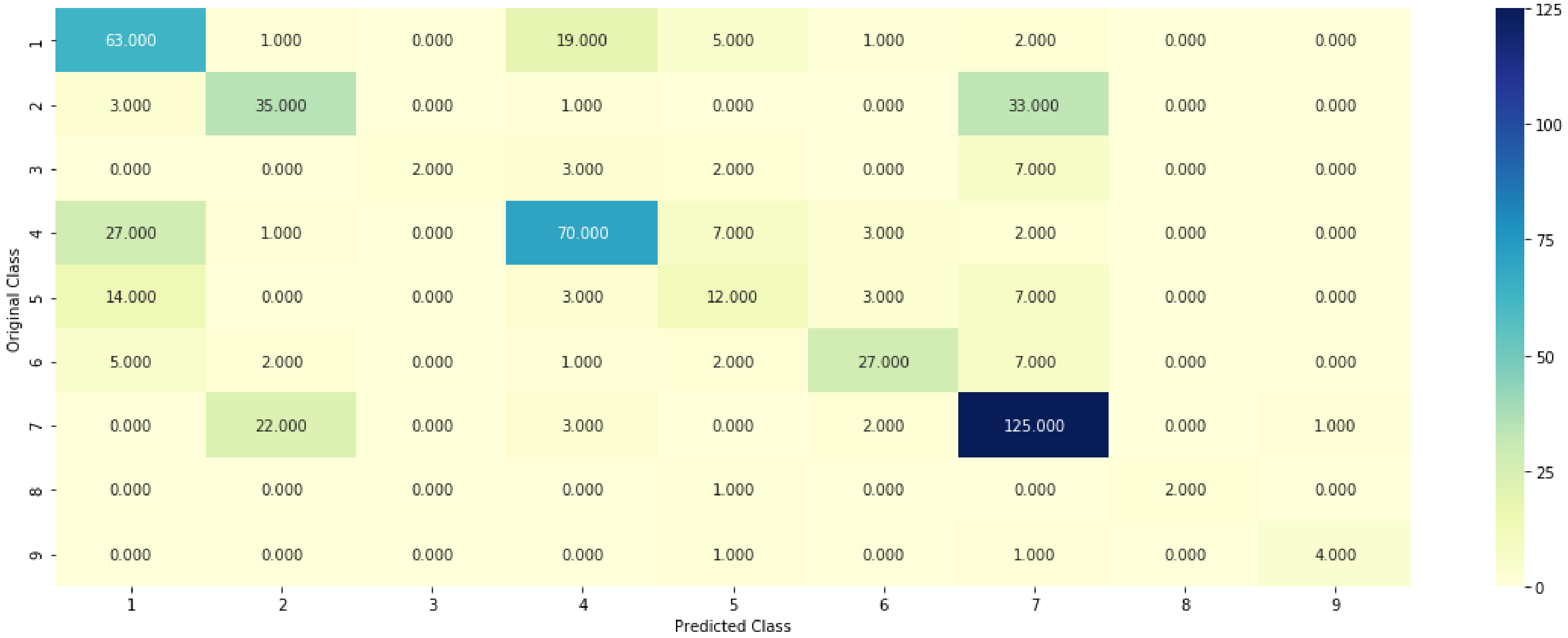


```
For values of best alpha =  0.0001 The train log loss is: 0.43658409980956725
For values of best alpha =  0.0001 The cross validation log loss is: 0.9935226923707908
For values of best alpha =  0.0001 The test log loss is: 0.9971535149059286
```

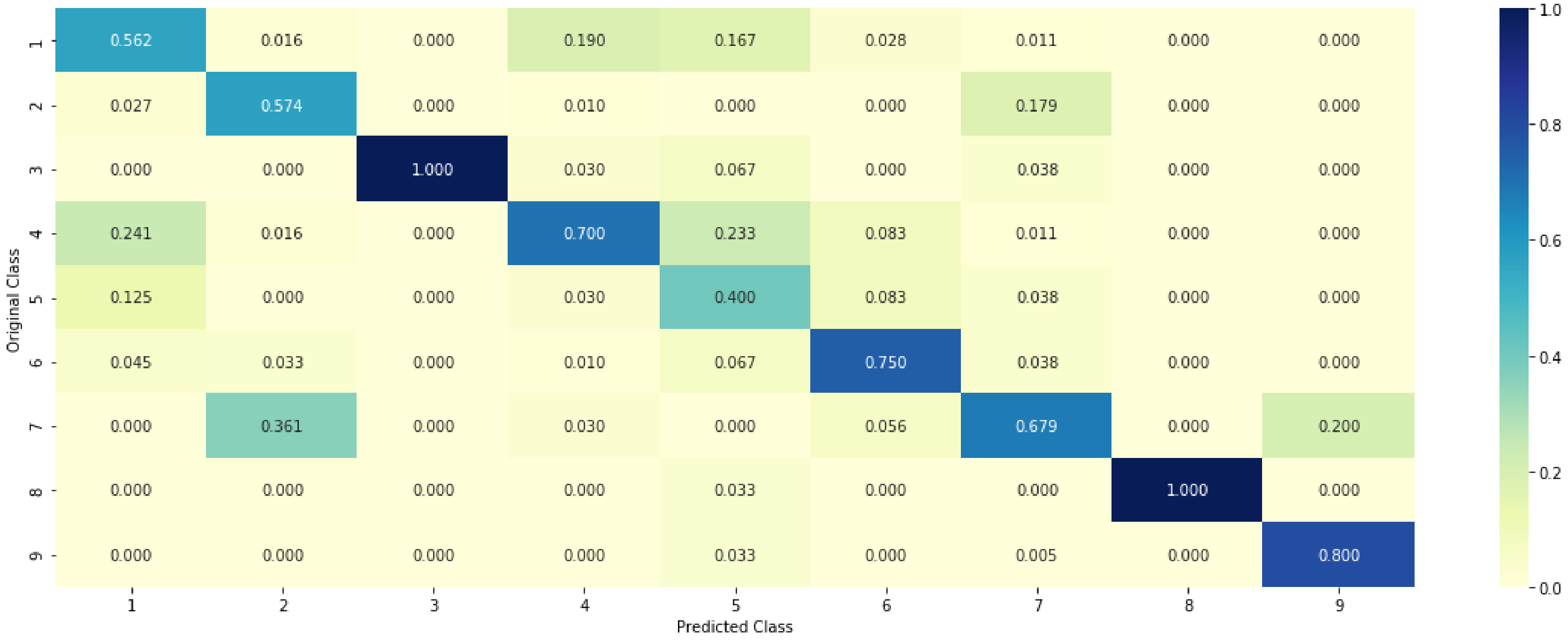
In [182]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
42)
predict_and_plot_confusion_matrix(train_x_onehot, y_train, cv_x_onehot, y_cv, clf)
```

```
Log loss : 0.9835840955161868
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

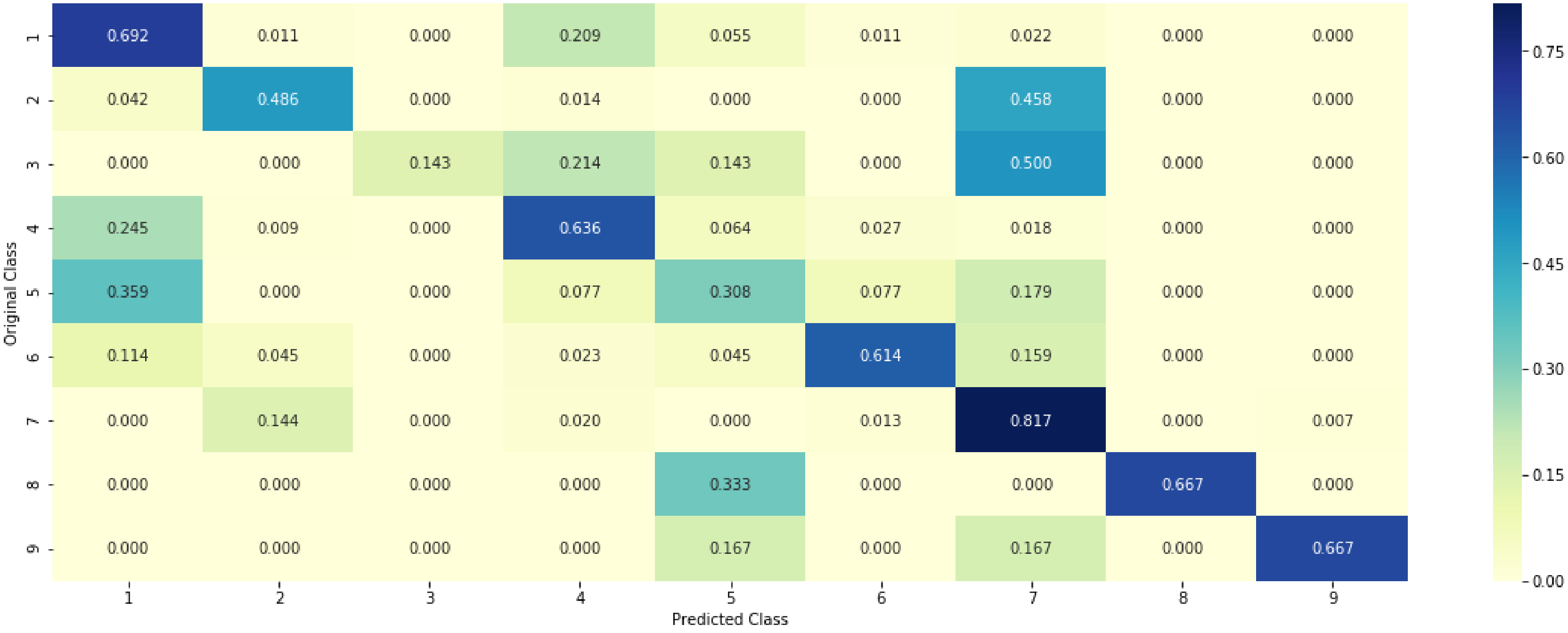


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.562	0.016	0.000	0.190	0.167	0.028	0.011	0.000	0.000
2	0.027	0.574	0.000	0.010	0.000	0.000	0.179	0.000	0.000
3	0.000	0.000	1.000	0.030	0.067	0.000	0.038	0.000	0.000
4	0.241	0.016	0.000	0.700	0.233	0.083	0.011	0.000	0.000
5	0.125	0.000	0.000	0.030	0.400	0.083	0.038	0.000	0.000
6	0.045	0.033	0.000	0.010	0.067	0.750	0.038	0.000	0.000
7	0.000	0.361	0.000	0.030	0.000	0.056	0.679	0.000	0.200
8	0.000	0.000	0.000	0.000	0.033	0.000	0.000	1.000	0.000
9	0.000	0.000	0.000	0.000	0.033	0.000	0.005	0.000	0.800

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



Original Class	1	2	3	4	5	6	7	8	9
1	0.692	0.011	0.000	0.209	0.055	0.011	0.022	0.000	0.000
2	0.042	0.486	0.000	0.014	0.000	0.000	0.458	0.000	0.000
3	0.000	0.000	0.143	0.214	0.143	0.000	0.500	0.000	0.000
4	0.245	0.009	0.000	0.636	0.064	0.027	0.018	0.000	0.000
5	0.359	0.000	0.000	0.077	0.308	0.077	0.179	0.000	0.000
6	0.114	0.045	0.000	0.023	0.045	0.614	0.159	0.000	0.000
7	0.000	0.144	0.000	0.020	0.000	0.013	0.817	0.000	0.007
8	0.000	0.000	0.000	0.000	0.333	0.000	0.000	0.667	0.000
9	0.000	0.000	0.000	0.000	0.167	0.000	0.167	0.000	0.667

Conclusion and Summary

In [183]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names=['Model','Train Log Loss','CV Los Loss','Test Log Loss']

x.add_row(['Naive Bayes',0.748,1.184,1.181])
x.add_row(['KNN',0.646,1.061, 1.072])
x.add_row(['Logistic regression (class balancing)',0.575, 1.01, 0.995])
x.add_row(['Logistic regression (no class balancing)',0.574,1.05,1.018])
x.add_row(['SVM',0.853, 1.191,1.178])
x.add_row(['RF (one hot encoded)',0.853, 1.191,1.178])
x.add_row(['RF (response coded)',0.059,1.399,1.429])
x.add_row(['Stacking (NB,SVM,LR)', 0.811,1.143,1.153])
x.add_row(['Max Voting',0.955,1.195,1.206])
x.add_row(['Logistic Regression(unigram biram)',0.900,1.229,1.163])
x.add_row(['Logistic Regression(feature engineering)',0.400,0.958,0.954])
x.add_row(['Logistic Regression(feature engineering 2)',0.430,0.993,0.997])

print(x)
```

```
+--------------------------------------------+----------------+-------------+---------------+
|                   Model                    | Train Log Loss | CV Los Loss | Test Log Loss |
+--------------------------------------------+----------------+-------------+---------------+
|                Naive Bayes                 |     0.748      |    1.184    |     1.181     |
|                    KNN                     |     0.646      |    1.061    |     1.072     |
|   Logistic regression (class balancing)    |     0.575      |     1.01    |     0.995     |
|  Logistic regression (no class balancing)  |     0.574      |     1.05    |     1.018     |
|                    SVM                     |     0.853      |    1.191    |     1.178     |
|            RF (one hot encoded)            |     0.853      |    1.191    |     1.178     |
|            RF (response coded)             |     0.059      |    1.399    |     1.429     |
|            Stacking (NB,SVM,LR)            |     0.811      |    1.143    |     1.153     |
|                 Max Voting                 |     0.955      |    1.195    |     1.206     |
|     Logistic Regression(unigram biram)     |      0.9       |    1.229    |     1.163     |
|  Logistic Regression(feature engineering)  |      0.4       |    0.958    |     0.954     |
| Logistic Regression(feature engineering 2) |      0.43      |    0.993    |     0.997     |
+--------------------------------------------+----------------+-------------+---------------+
```

We observe that class balancing has a significant impact on model performance.
Logistic Regression comes up as the top performing model(0.995 Test Log Loss).
With the help of simple feature engineering,the value of Test Log loss was brought down below 1..