Assignment 15 - Personalized Cancer Diagnosis

By Aziz Presswala

In [84]:

```
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 collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
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 [85]:

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

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

Out[85]:

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

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

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

3.1.2. Reading Text Data

In [86]:

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

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

Out[86]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 256.54801159899944 seconds
```

In [87]:

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

Out[87]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

In [88]:

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

Out[88]:

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

In [89]:

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

In [90]:

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

Out[90]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

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

In [11]:

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

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

Part 1 - Using TFIDF instead of CountVectorizer

In [18]:

```
# 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 da
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alp
# 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/var
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular
        # 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 f
            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, 0.1363
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733
    #      ...
    #     }
    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
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the
    # 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 [19]:

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

```
Number of Unique Genes : 228
BRCA1     173
TP53      103
EGFR       90
BRCA2      85
PTEN       82
BRAF       59
KIT        56
PDGFRA     45
ALK        41
CDKN2A     39
Name: Gene, dtype: int64
```

In [20]:

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

```
Ans: There are 228 different categories of genes in the train data, and they
are distibuted as follows
```

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 [29]:

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

In [30]:

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

Out[30]:

```
1599       VHL
1385     FGFR1
1607       VHL
232       EGFR
1964    CTNNB1
Name: Gene, dtype: object
```

In [31]:

```
gene_vectorizer.get_feature_names()
```

Out[31]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
```

In [32]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 227)
```

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 [33]:

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

```
Number of Unique Variations : 1939
Truncating_Mutations    63
Amplification           43
Deletion                40
Fusions                 17
T58I                     3
Q61R                     3
Overexpression           3
Q61K                     2
G67R                     2
K117N                    2
Name: Variation, dtype: int64
```

In [34]:

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

```
Ans: There are 1939 different categories of variations in the train data, an
d they are distibuted as follows
```

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 [35]:

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

In [36]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-ho
t encoding method. The shape of Variation feature: (2124, 1974)
```

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 [37]:

```
# 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 [38]:

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

In [39]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer()
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 o
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 occu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

```
Total number of unique words in train data : 124152
```

In [40]:

```
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 [41]:

```
# 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 [42]:

```
#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()))
```

4. Machine Learning Models

In [69]:

```
# 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

    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 d
    # 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 d
    # 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=la
    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=la
    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=la
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [70]:

```
#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
    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.sh
    plot_confusion_matrix(test_y, pred_y)
```

In [71]:

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

In [53]:

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

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

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

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

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

Stacking the three types of features

In [47]:

```
# 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_featu
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehot

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

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).
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']))
```

In [48]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 126353)
(number of data points * number of features) in test data =  (665, 126353)
(number of data points * number of features) in cross validation data = (53
2, 126353)
```

4.1. Base Line Model

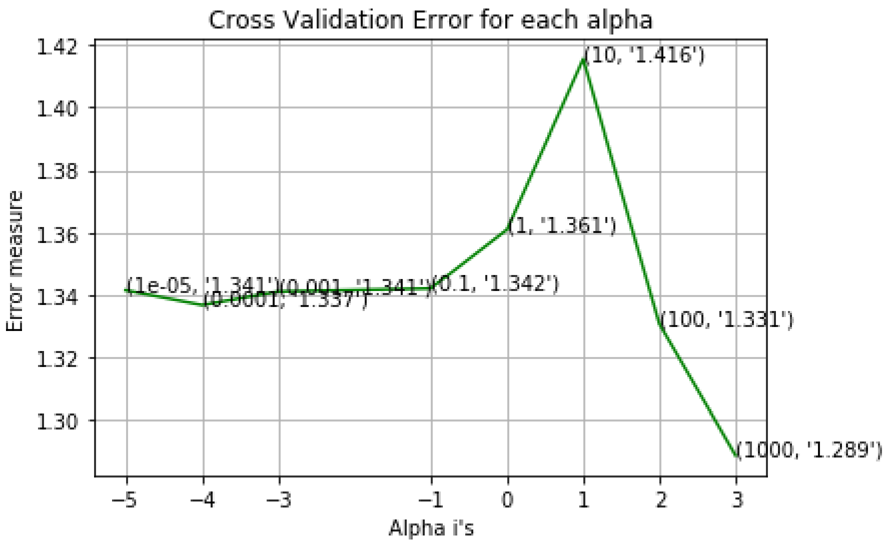
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [50]:

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

```
for alpha = 1e-05
Log Loss : 1.3414482490296533
for alpha = 0.0001
Log Loss : 1.3366731781303618
for alpha = 0.001
Log Loss : 1.341019023545034
for alpha = 0.1
Log Loss : 1.3420109809487226
for alpha = 1
Log Loss : 1.360950490506508
for alpha = 10
Log Loss : 1.4155370815885502
for alpha = 100
Log Loss : 1.3306303879652945
for alpha = 1000
Log Loss : 1.2885616476404635
```



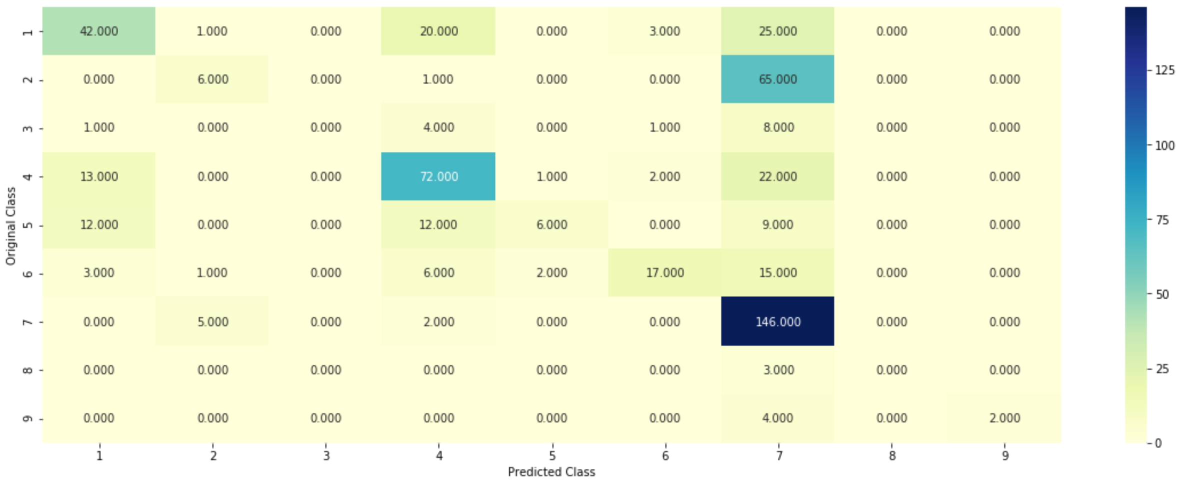

```
For values of best alpha =  1000 The train log loss is: 0.8336996713166448
For values of best alpha =  1000 The cross validation log loss is: 1.2885616
476404635
For values of best alpha =  1000 The test log loss is: 1.242693479837896
```

4.1.1.2. Testing the model with best hyper paramters

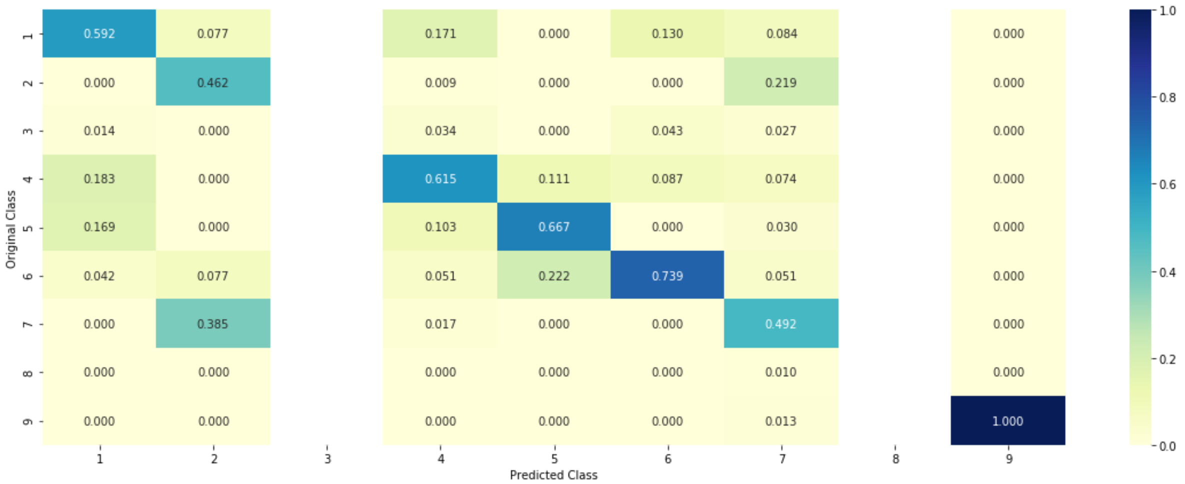
In [54]:

```
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_onehotCodi
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

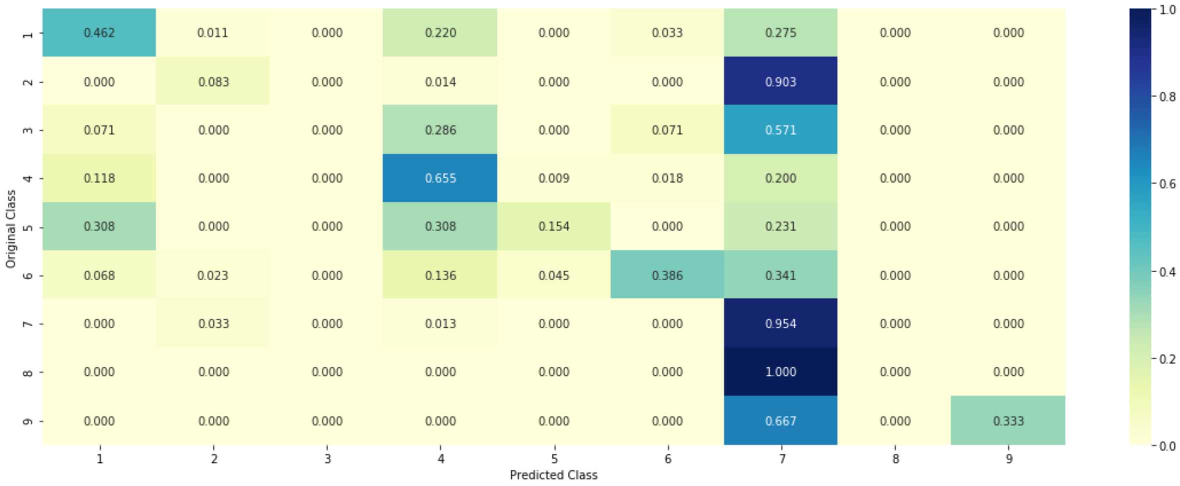
```
Log Loss : 1.2885616476404635
Number of missclassified point : 0.45300751879699247
-------------------- Confusion matrix --------------------
```




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




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



4.2. Logistic Regression

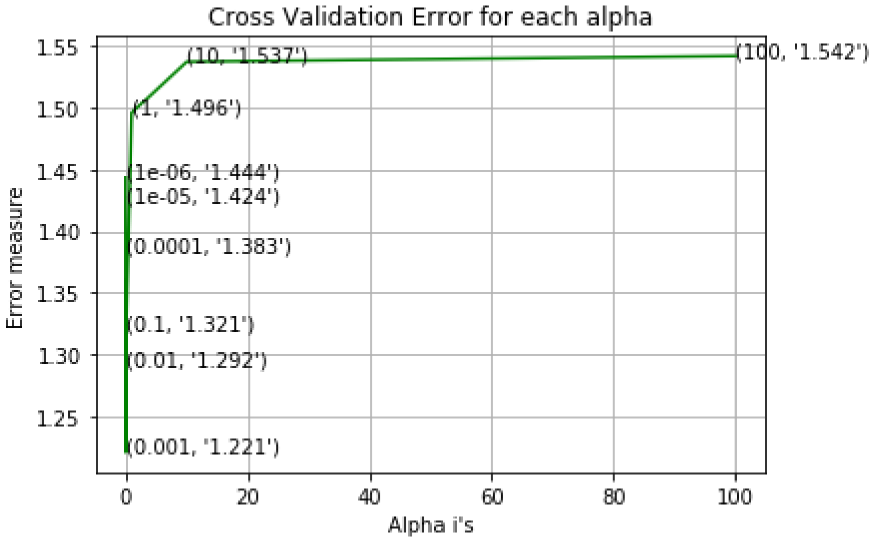
4.2.1. With Class balancing

4.2.1.1. Hyper paramter tuning

In [58]:

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

```
for alpha = 1e-06
Log Loss : 1.4436375573388014
for alpha = 1e-05
Log Loss : 1.4236134440878072
for alpha = 0.0001
Log Loss : 1.3832565146438798
for alpha = 0.001
Log Loss : 1.2207505328895352
for alpha = 0.01
Log Loss : 1.2916338909695328
for alpha = 0.1
Log Loss : 1.3205480181042961
for alpha = 1
Log Loss : 1.4959139518769007
for alpha = 10
Log Loss : 1.5372729889261745
for alpha = 100
Log Loss : 1.5419443908982053
```



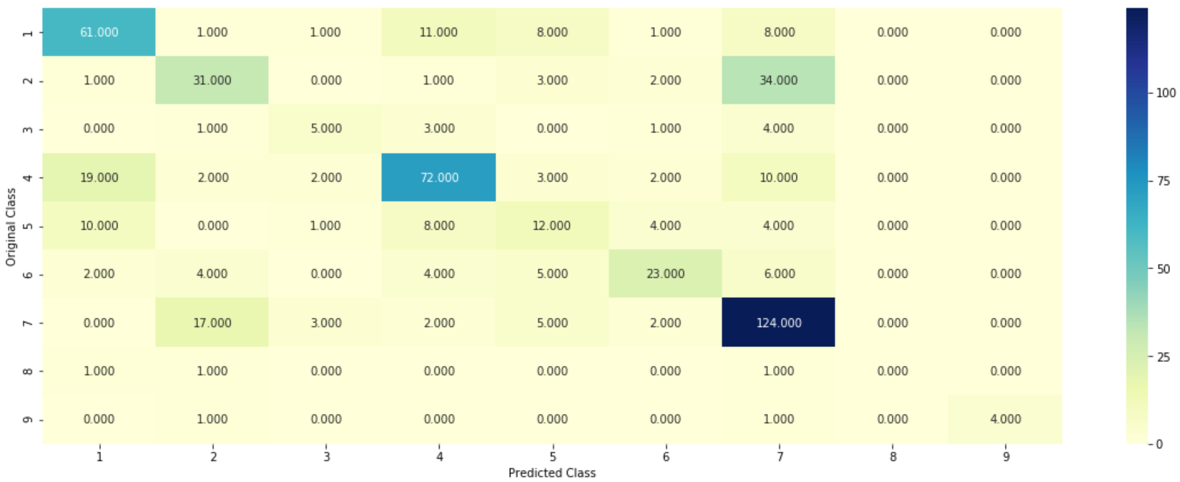

```
For values of best alpha =  0.001 The train log loss is: 0.5717535019817939
For values of best alpha =  0.001 The cross validation log loss is: 1.220750
5328895352
For values of best alpha =  0.001 The test log loss is: 1.1224455658747603
```

4.3.1.2. Testing the model with best hyper paramters

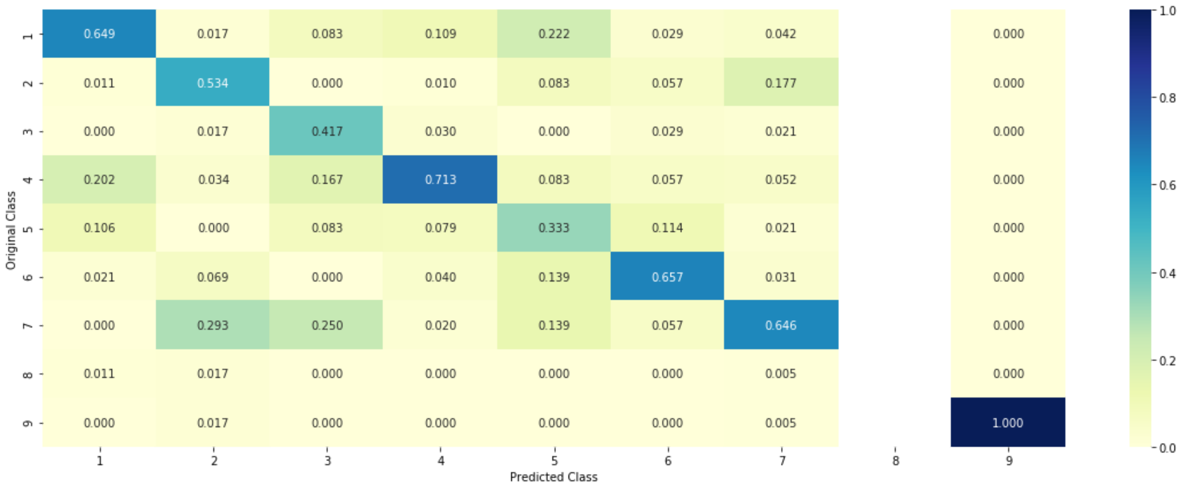
In [59]:

```
#-----------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

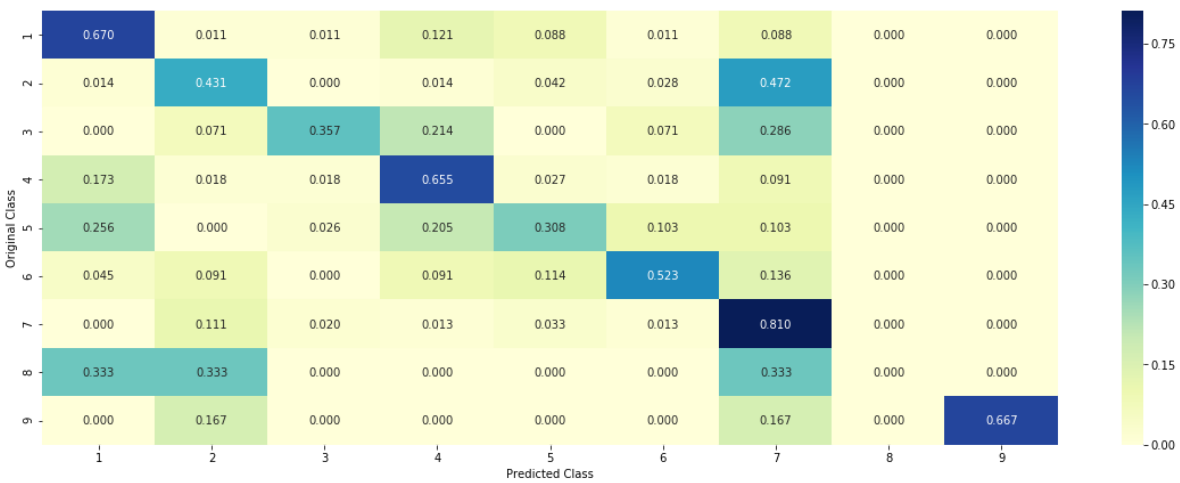
```
Log loss : 1.2207505328895352
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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




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



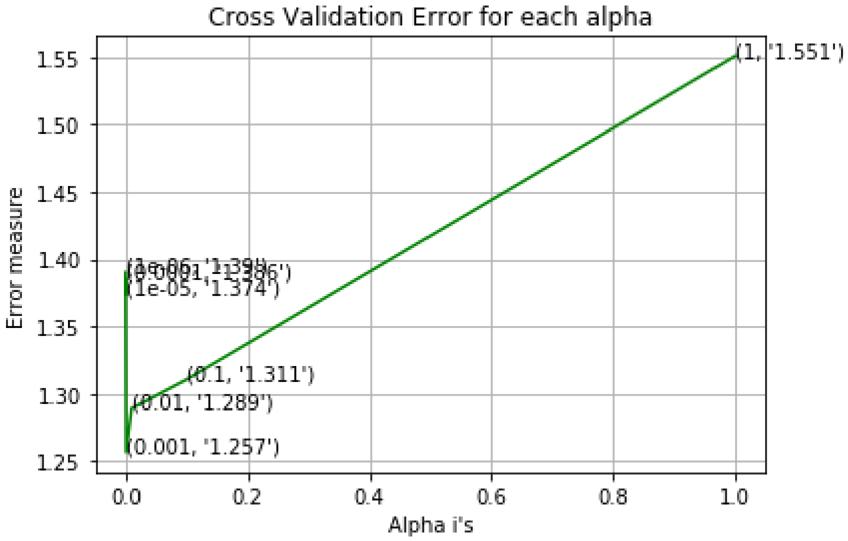
4.2.2. Without Class balancing

4.2.2.1. Hyper paramter tuning

In [62]:

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

```
for alpha = 1e-06
Log Loss : 1.390135258565077
for alpha = 1e-05
Log Loss : 1.373841255190033
for alpha = 0.0001
Log Loss : 1.3857132209241034
for alpha = 0.001
Log Loss : 1.2565610805567047
for alpha = 0.01
Log Loss : 1.2891458017193123
for alpha = 0.1
Log Loss : 1.3105190616395508
for alpha = 1
Log Loss : 1.5507272412655329
```



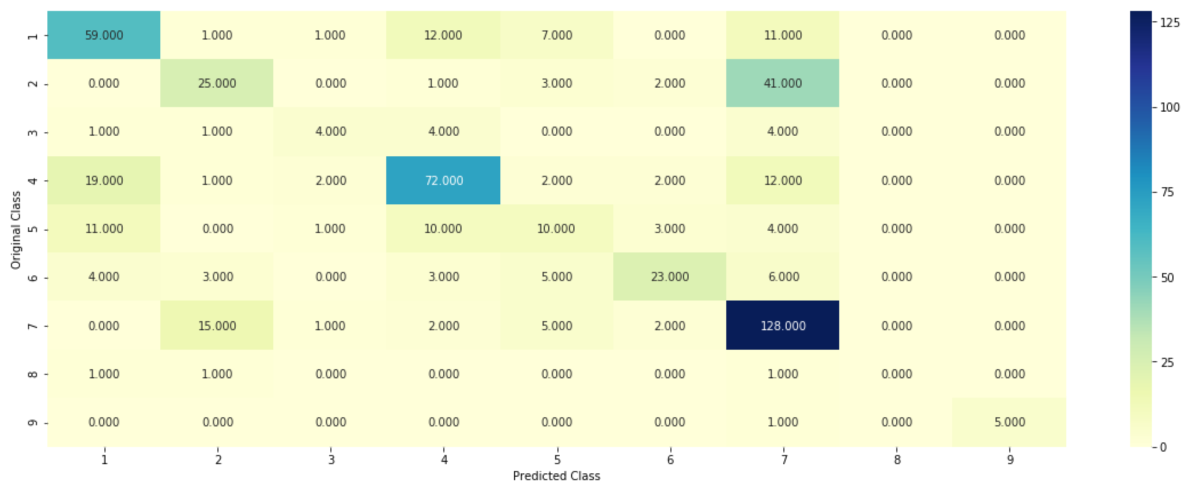

```
For values of best alpha =  0.001 The train log loss is: 0.5574483911808396
For values of best alpha =  0.001 The cross validation log loss is: 1.256561
0805567047
For values of best alpha =  0.001 The test log loss is: 1.1406495636857226
```

4.2.2.2. Testing model with best hyper parameters

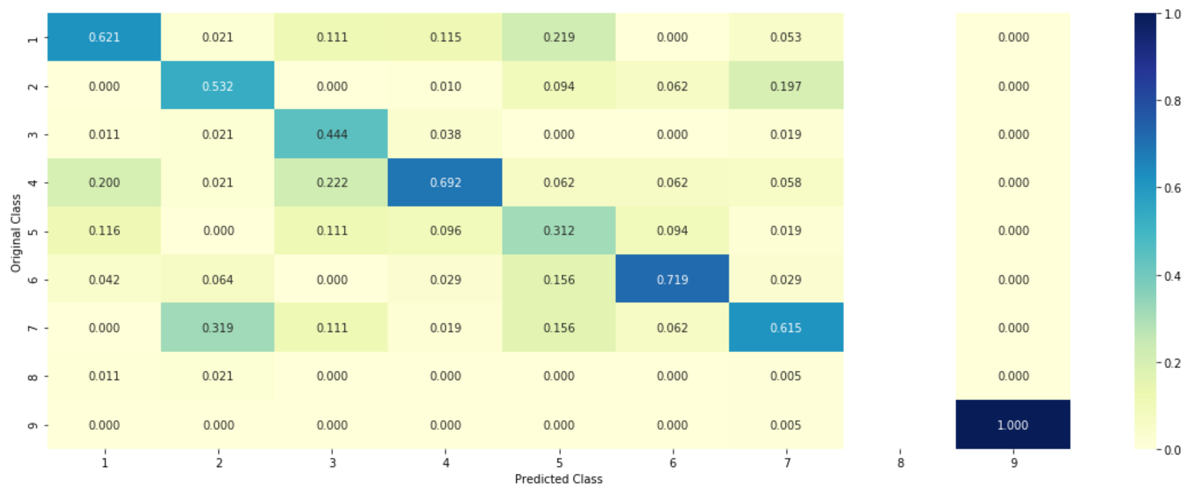
In [63]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# 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 Desce
# 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, c
```

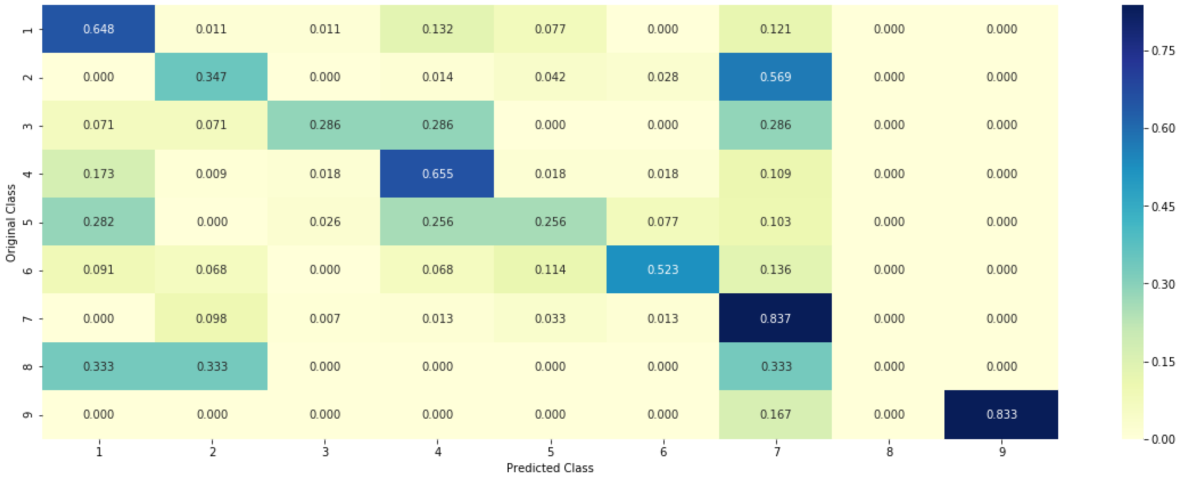
```
Log loss : 1.2565610805567047
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```




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




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



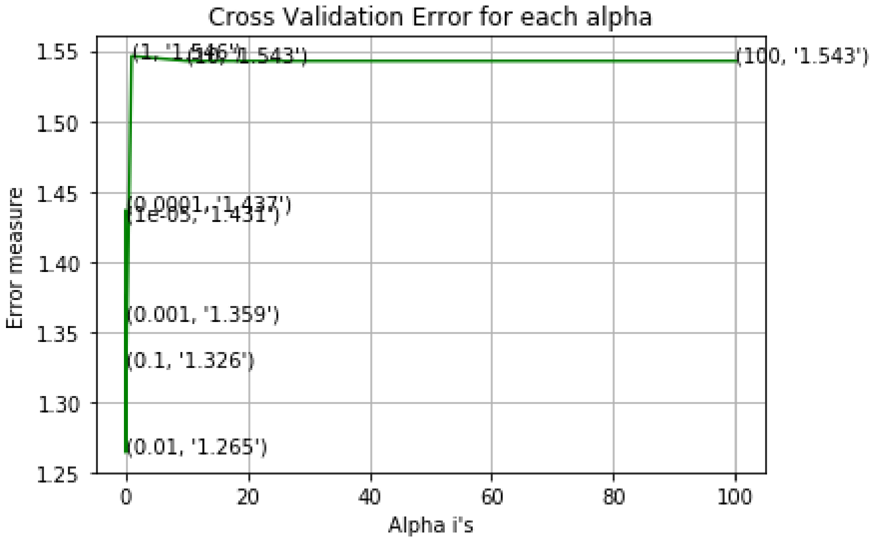
4.3. Linear Support Vector Machines

4.3.1. Hyper paramter tuning

In [64]:

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

```
for C = 1e-05
Log Loss : 1.4310323875498414
for C = 0.0001
Log Loss : 1.4368784715977974
for C = 0.001
Log Loss : 1.359375538223838
for C = 0.01
Log Loss : 1.2647565841093598
for C = 0.1
Log Loss : 1.3263405636856178
for C = 1
Log Loss : 1.5462759874504772
for C = 10
Log Loss : 1.5427609798690156
for C = 100
Log Loss : 1.54276102102633
```




```
For values of best alpha =  0.01 The train log loss is: 0.6582224358479796
For values of best alpha =  0.01 The cross validation log loss is: 1.2647565
841093598
For values of best alpha =  0.01 The test log loss is: 1.154647682145246
```

4.3.2. Testing model with best hyper parameters

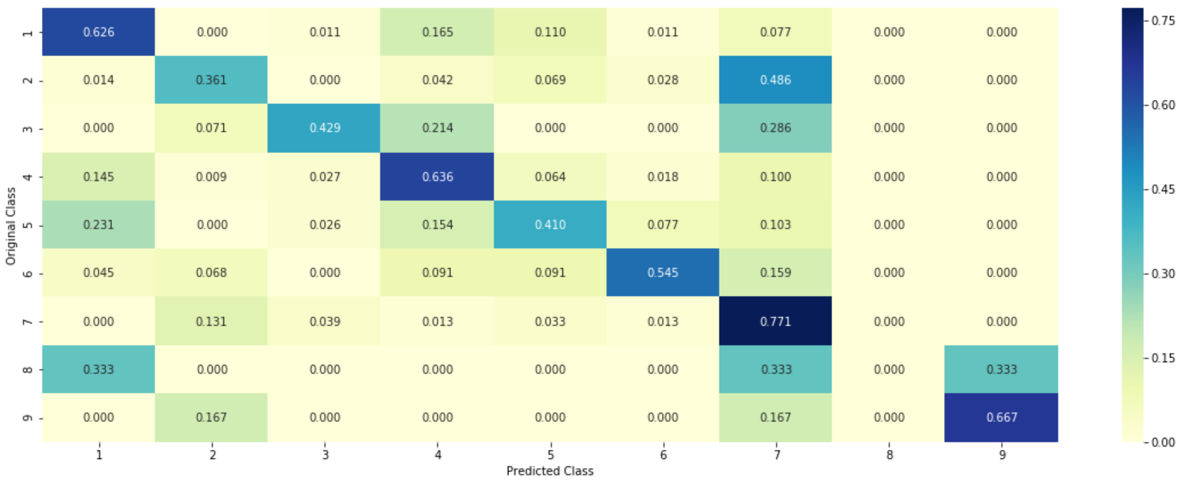
In [65]:

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

```
Log loss : 1.2647565841093598
Number of mis-classified points : 0.39661654135338 35
-------------------- Confusion matrix --------------------
```


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


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



4.4 Random Forest Classifier

4.4.1. Hyper paramter tuning (With One hot Encoding)

In [66]:

```
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_
        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
        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_a
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_d
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:"
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo
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:",
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2995007985198626
for n_estimators = 100 and max depth =  10
Log Loss : 1.2500748285683327
for n_estimators = 200 and max depth =  5
Log Loss : 1.2803540144944439
for n_estimators = 200 and max depth =  10
Log Loss : 1.238150579347561
for n_estimators = 500 and max depth =  5
Log Loss : 1.2753628100588525
for n_estimators = 500 and max depth =  10
Log Loss : 1.2297728653723108
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2737378818255538
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2256946961728419
for n_estimators = 2000 and max depth =  5
Log Loss : 1.27040202711567
for n_estimators = 2000 and max depth =  10
```

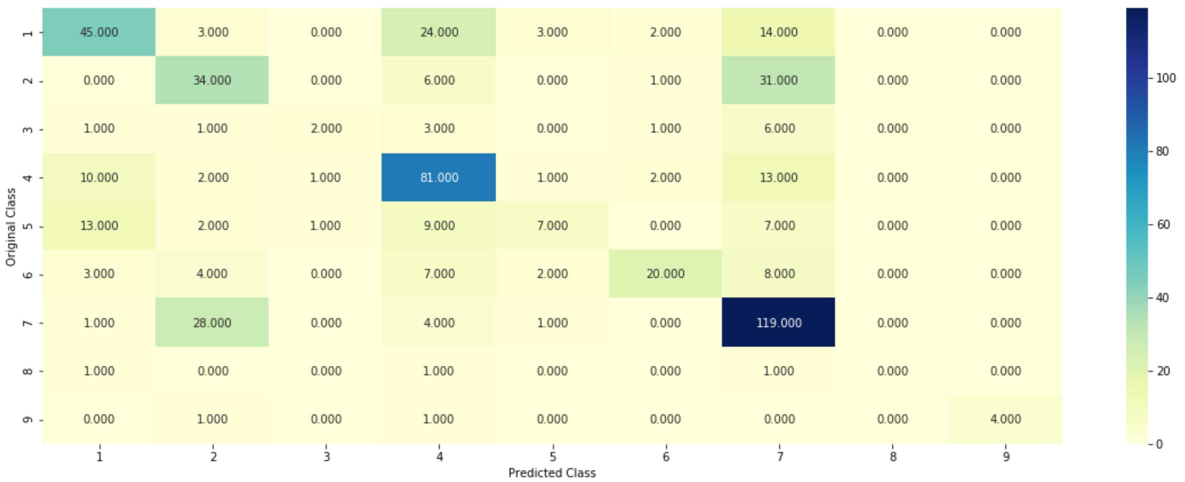
```
Log Loss : 1.2220463287306031
For values of best estimator =  2000 The train log loss is: 0.66187453987164
2
For values of best estimator =  2000 The cross validation log loss is: 1.222
0463287306031
For values of best estimator =  2000 The test log loss is: 1.178871620711187
3
```

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

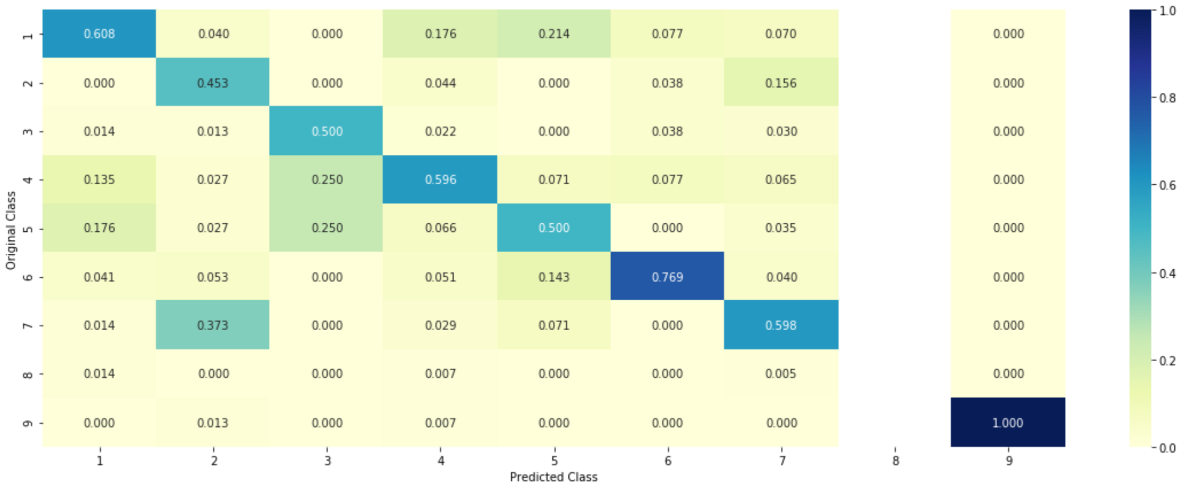
In [67]:

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

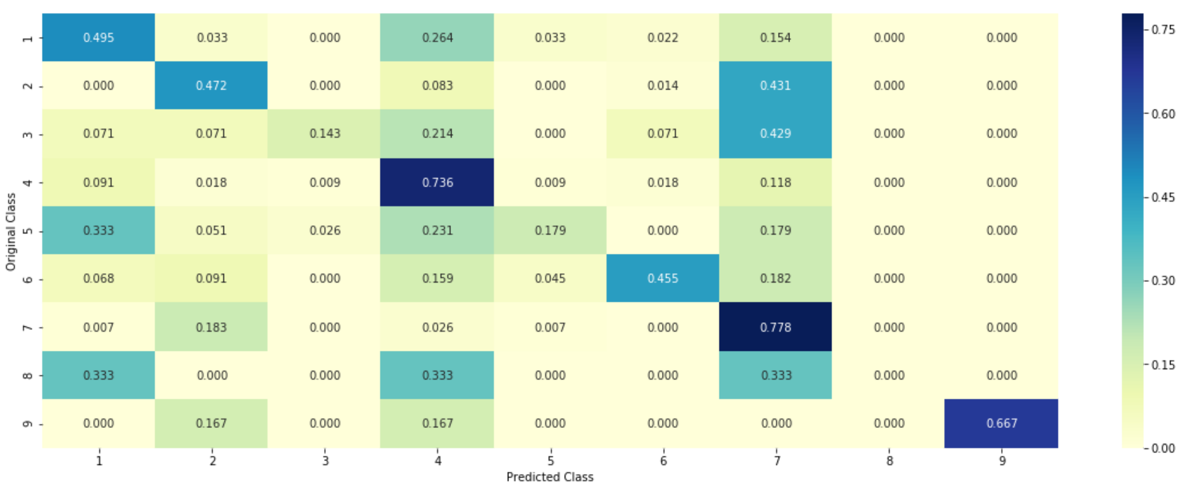
```
Log loss : 1.2220463287306031
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```




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




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



4.5 Stack the models

4.5.1 testing with hyper parameter tuning

In [68]:

```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', randon
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_s
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_
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(
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_onehotC
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=l
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

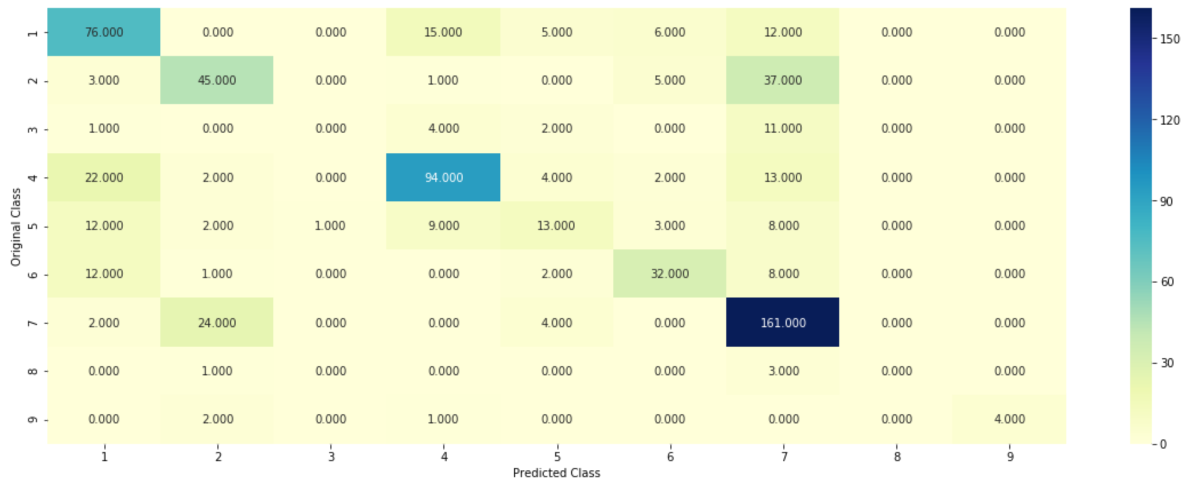
```
Logistic Regression :  Log Loss: 1.22
Support vector machines : Log Loss: 1.55
Naive Bayes : Log Loss: 1.34
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.041
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.544
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.204
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.330
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.541
```

4.5.2 testing the model with the best hyper parameters

In [69]:

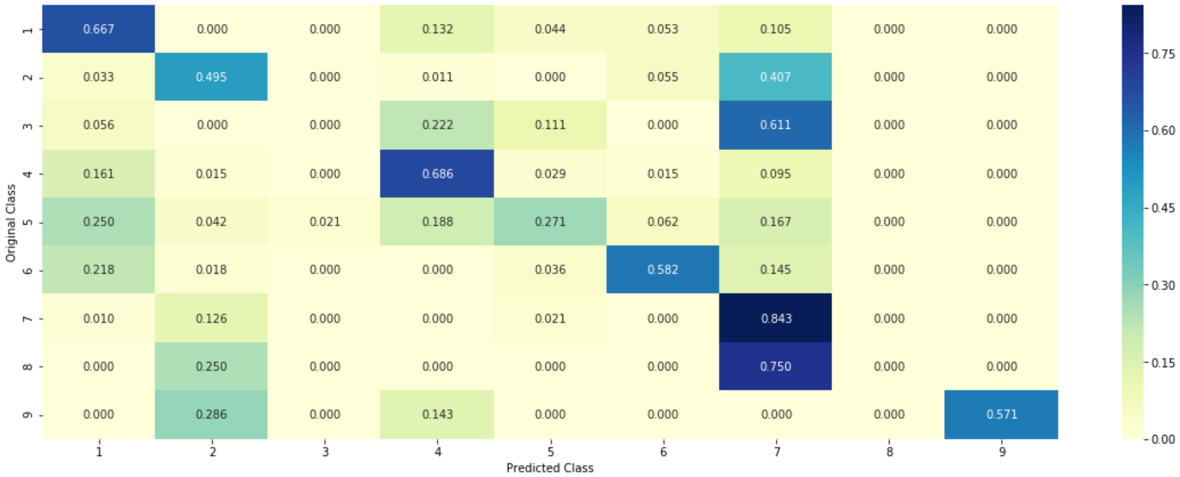
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, u
sclf.fit(train_x_onehotCoding, train_y)

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_onehotCodir
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.6310243731140416
Log loss (CV) on the stacking classifier : 1.2040018380569308
Log loss (test) on the stacking classifier : 1.16049600435379
Number of missclassified point : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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

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

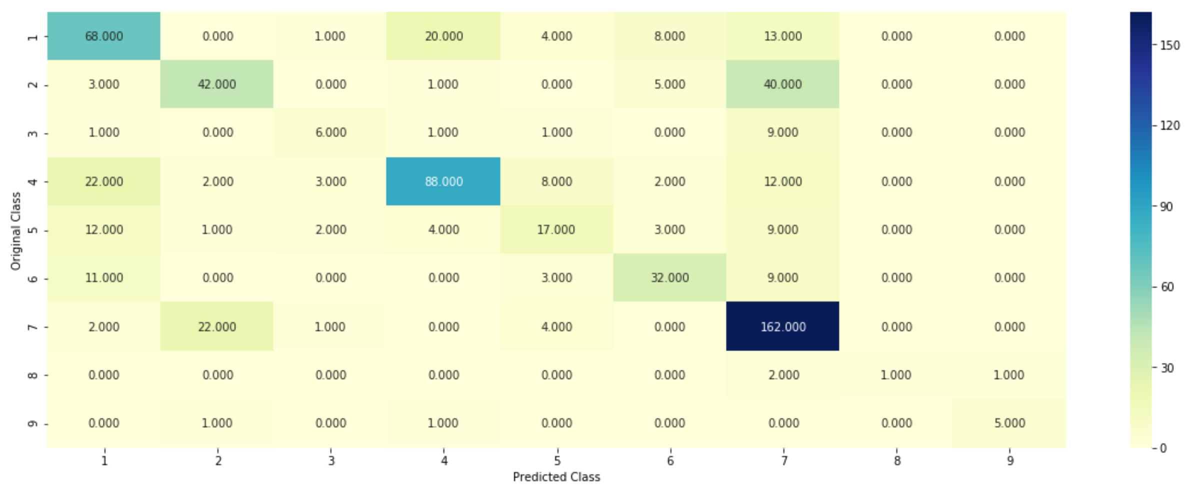


4.5.3 Maximum Voting classifier

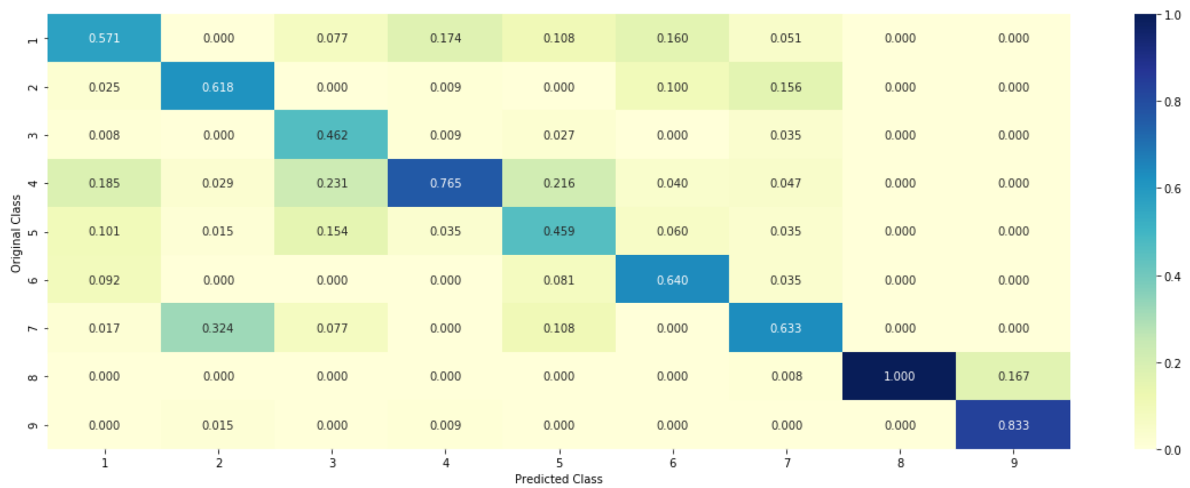
In [70]:

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

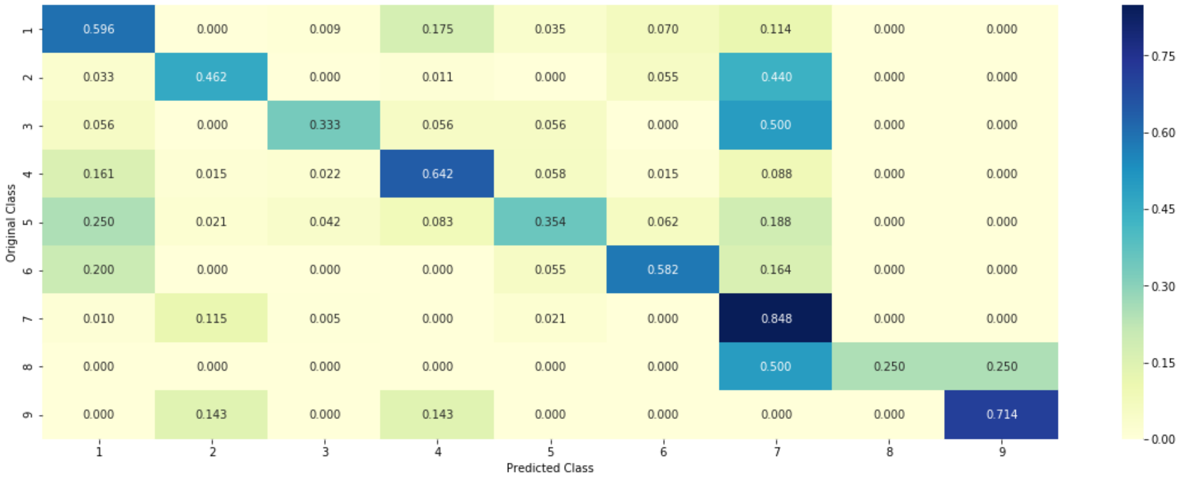
```
Log loss (train) on the VotingClassifier : 0.8447538604417815
Log loss (CV) on the VotingClassifier : 1.2370163897256812
Log loss (test) on the VotingClassifier : 1.1904063722408995
Number of missclassified point : 0.36691729323308270
-------------------- Confusion matrix --------------------
```




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




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



Summary Table

In [72]:

```
from prettytable import PrettyTable
t=PrettyTable()
t.field_names = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss']
t.add_row(['Naive Bayes', '0.83369', '1.28856', '1.24269'])
t.add_row(['Linear Regression with Class balancing', '0.57175', '1.22075', '1.12244'])
t.add_row(['Linear Regression without Class balancing', '0.55744', '1.25656', '1.14064'])
t.add_row(['Support Vector Machine', '0.65822', '1.26475', '1.15464'])
t.add_row(['Random Forest', '0.66187', '1.22200', '1.17887'])
t.add_row(['Stacking Classifier', '0.63102', '1.20400', '1.16049'])
t.add_row(['Maximum Voting Classifier', '0.84475', '1.23701', '1.19040'])

print(t)
```

```
+-------------------------------------------+----------------+-------------+
---------------+
|                   Model                   | Train Log Loss | CV Log Loss |
Test Log Loss |
+-------------------------------------------+----------------+-------------+
---------------+
|                Naive Bayes                |    0.83369     |   1.28856   |
1.24269    |
|    Linear Regression with Class balancing  |    0.57175     |   1.22075   |
1.12244    |
|  Linear Regression without Class balancing |    0.55744     |   1.25656   |
1.14064    |
|           Support Vector Machine          |    0.65822     |   1.26475   |
1.15464    |
|               Random Forest               |    0.66187     |   1.22200   |
1.17887    |
|            Stacking Classifier            |    0.63102     |   1.20400   |
1.16049    |
|         Maximum Voting Classifier         |    0.84475     |   1.23701   |
1.19040    |
+-------------------------------------------+----------------+-------------+
---------------+
```

Part 2 - Using Top 1000 TFIDF features

3.2.1 Univariate Analysis on Gene Feature

In [74]:

```
# 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 [75]:

```
gene_vectorizer.get_feature_names()
```

Out[75]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
```

In [76]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 227)
```

3.2.2 Univariate Analysis on Variation Feature

In [77]:

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

In [78]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-ho
t encoding method. The shape of Variation feature: (2124, 1000)
```

3.2.3 Univariate Analysis on Text Feature

In [79]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number o
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 occu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

In [80]:

```
# 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 [81]:

```
#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()))
```

Stacking the three types of features

In [82]:

```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_featu
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehot

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

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).
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']))
```

In [83]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2227)
(number of data points * number of features) in test data =  (665, 2227)
(number of data points * number of features) in cross validation data = (53
2, 2227)
```

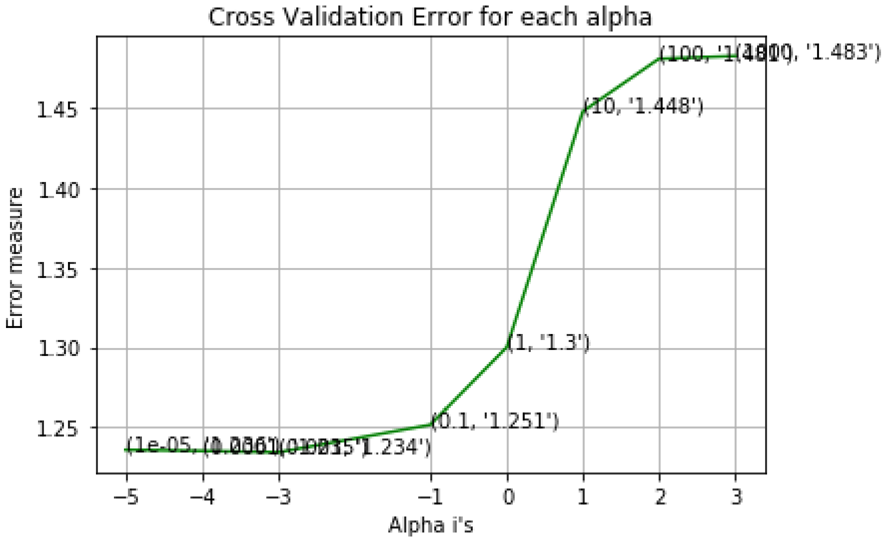
4. Machine Learning Models

4.1.1. Naive Bayes

In [84]:

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

```
for alpha = 1e-05
Log Loss : 1.2355022588682
for alpha = 0.0001
Log Loss : 1.2348799403403456
for alpha = 0.001
Log Loss : 1.2339588999382418
for alpha = 0.1
Log Loss : 1.251270623932791
for alpha = 1
Log Loss : 1.299864073021911
for alpha = 10
Log Loss : 1.448144059255936
for alpha = 100
Log Loss : 1.4812653256312
for alpha = 1000
Log Loss : 1.4830546327048448
```

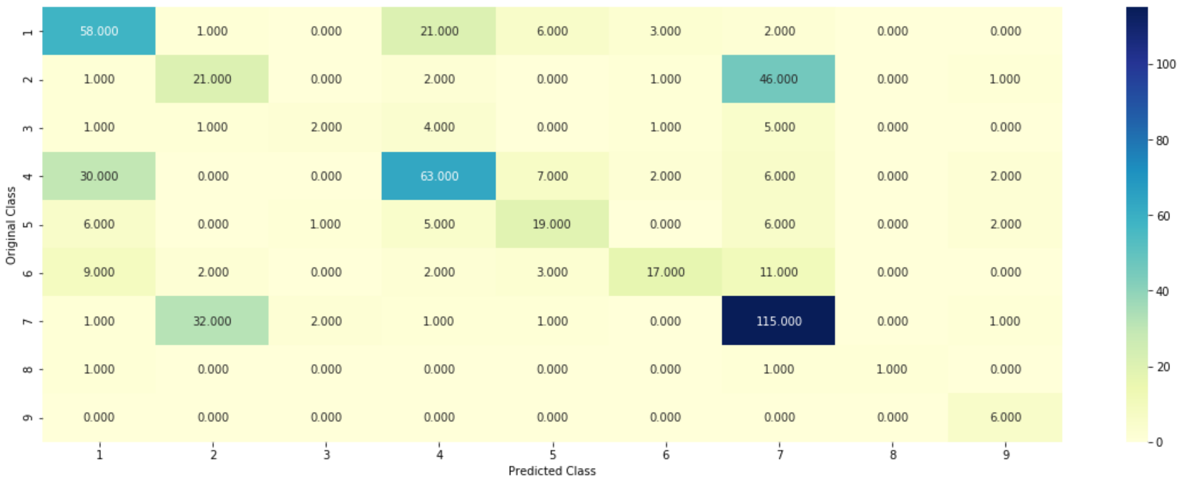



```
For values of best alpha =  0.001 The train log loss is: 0.7285292492932098
For values of best alpha =  0.001 The cross validation log loss is: 1.233958
8999382418
For values of best alpha =  0.001 The test log loss is: 1.1659319858571338
```

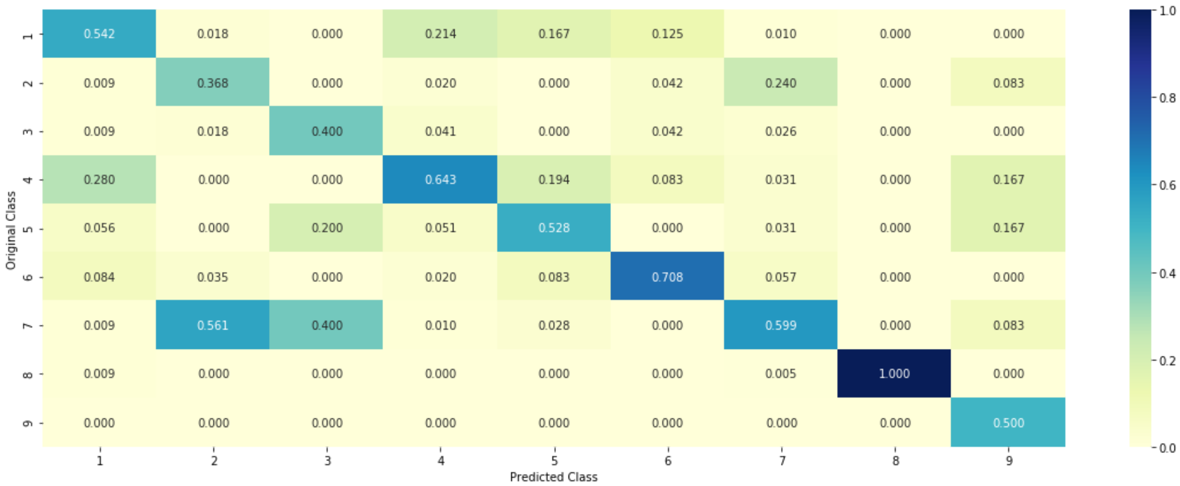
In [85]:

```
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_onehotCodi
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

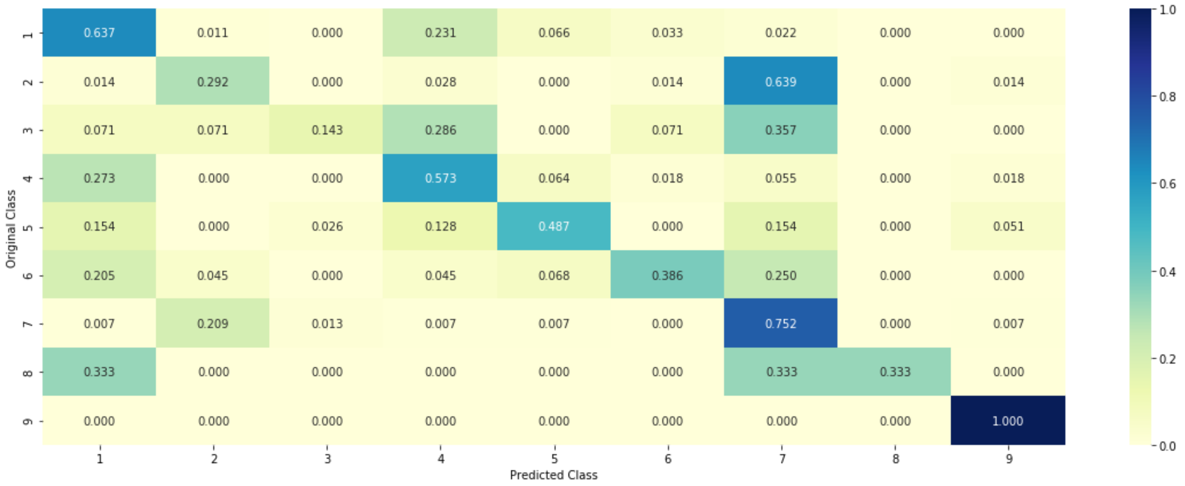
```
Log Loss : 1.2339588999382418
Number of missclassified point : 0.4323308270676692
-------------------- Confusion matrix --------------------
```




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




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



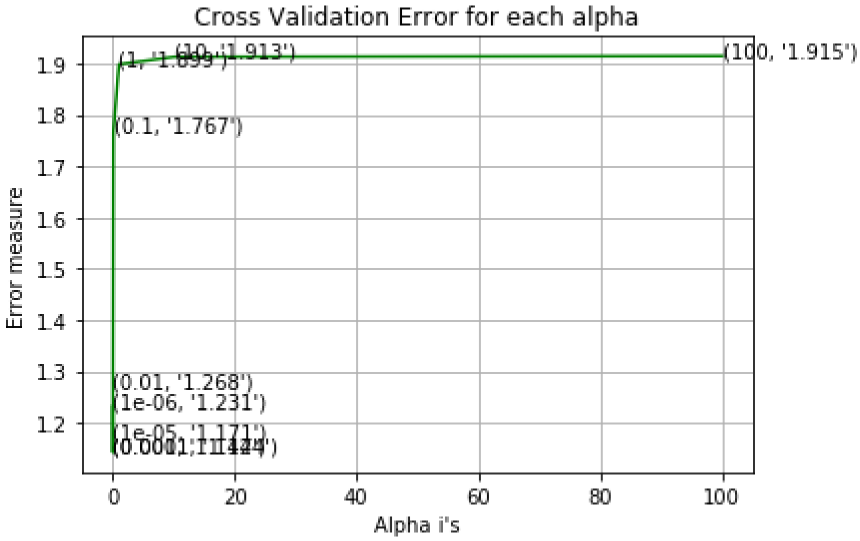
4.3. Logistic Regression

4.3.1. With Class balancing

In [86]:

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

```
for alpha = 1e-06
Log Loss : 1.230993092908709
for alpha = 1e-05
Log Loss : 1.1706380135890329
for alpha = 0.0001
Log Loss : 1.143971044997503
for alpha = 0.001
Log Loss : 1.1421580146097203
for alpha = 0.01
Log Loss : 1.2683535032774222
for alpha = 0.1
Log Loss : 1.7674110224663384
for alpha = 1
Log Loss : 1.8985997685648925
for alpha = 10
Log Loss : 1.9130995776121724
for alpha = 100
Log Loss : 1.9148451464968697
```

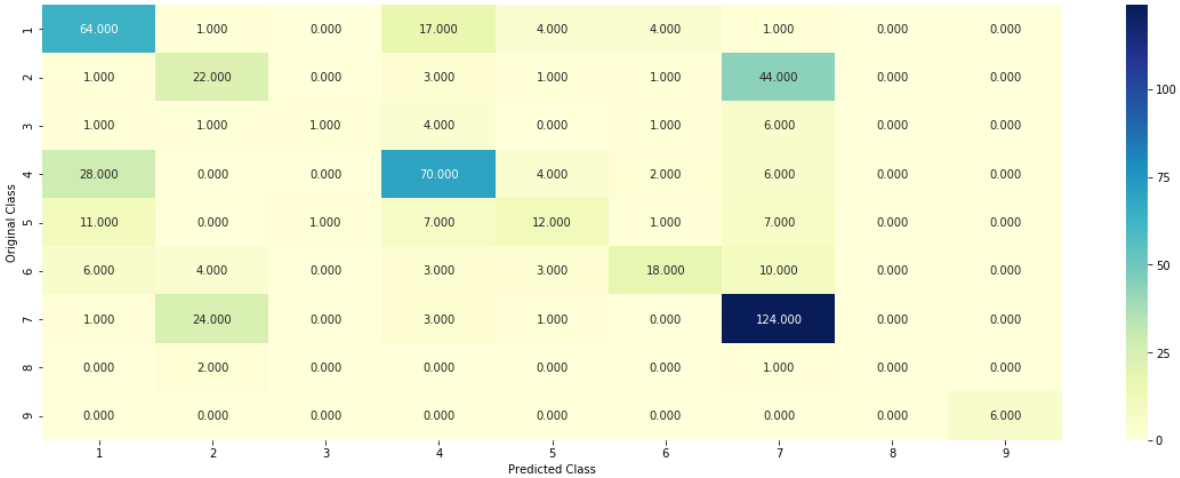



```
For values of best alpha =  0.001 The train log loss is: 0.7898475305677636
For values of best alpha =  0.001 The cross validation log loss is: 1.142158
0146097203
For values of best alpha =  0.001 The test log loss is: 1.0806870264944455
```

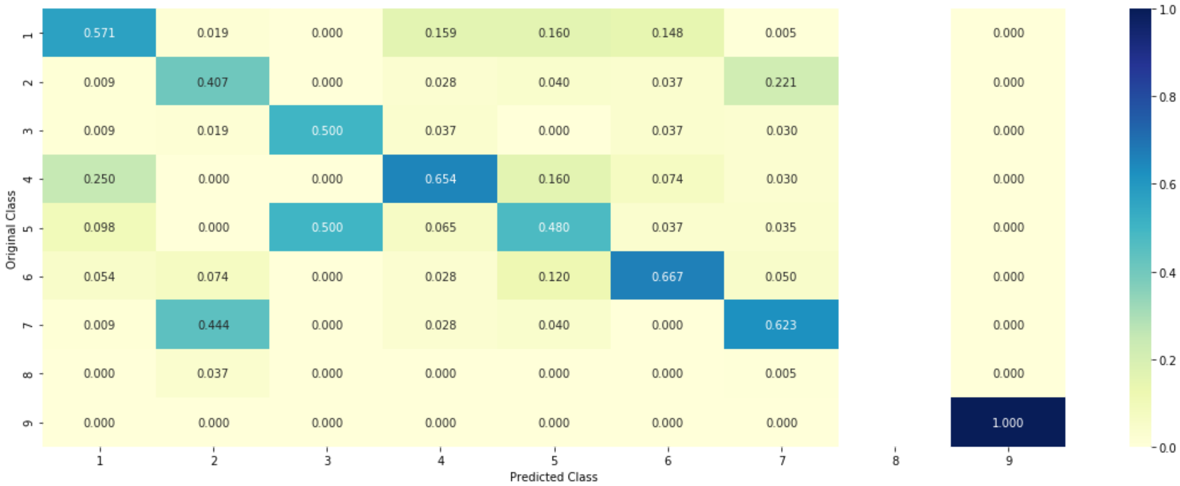
In [87]:

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

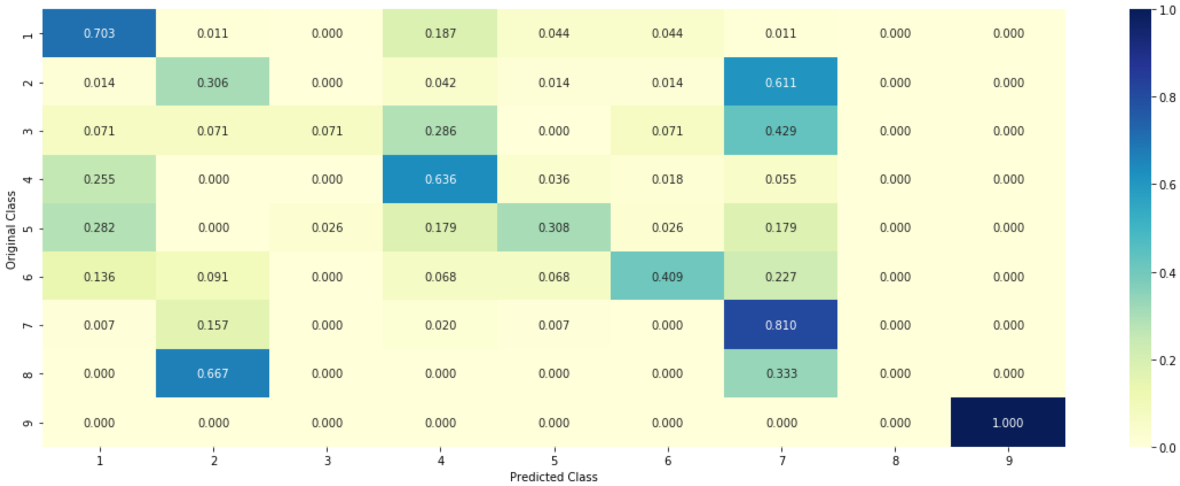
```
Log loss : 1.1421580146097203
Number of mis-classified points : 0.40413533383458647
-------------------- Confusion matrix --------------------
```




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




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

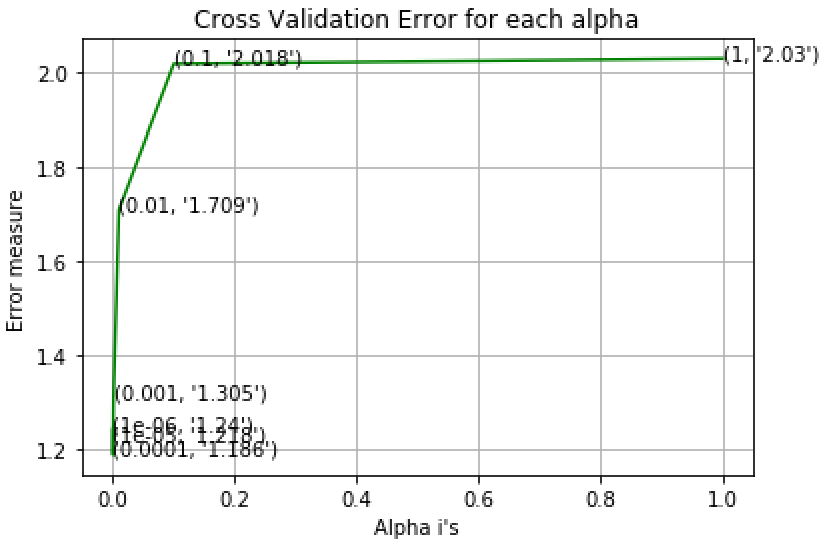


4.3.2. Without Class balancing

In [88]:

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

```
for alpha = 1e-06
Log Loss : 1.2404488470608344
for alpha = 1e-05
Log Loss : 1.2181349796192469
for alpha = 0.0001
Log Loss : 1.1861084709067171
for alpha = 0.001
Log Loss : 1.3049478138782458
for alpha = 0.01
Log Loss : 1.7088649752297245
for alpha = 0.1
Log Loss : 2.018488583320742
for alpha = 1
Log Loss : 2.0298755346633546
```

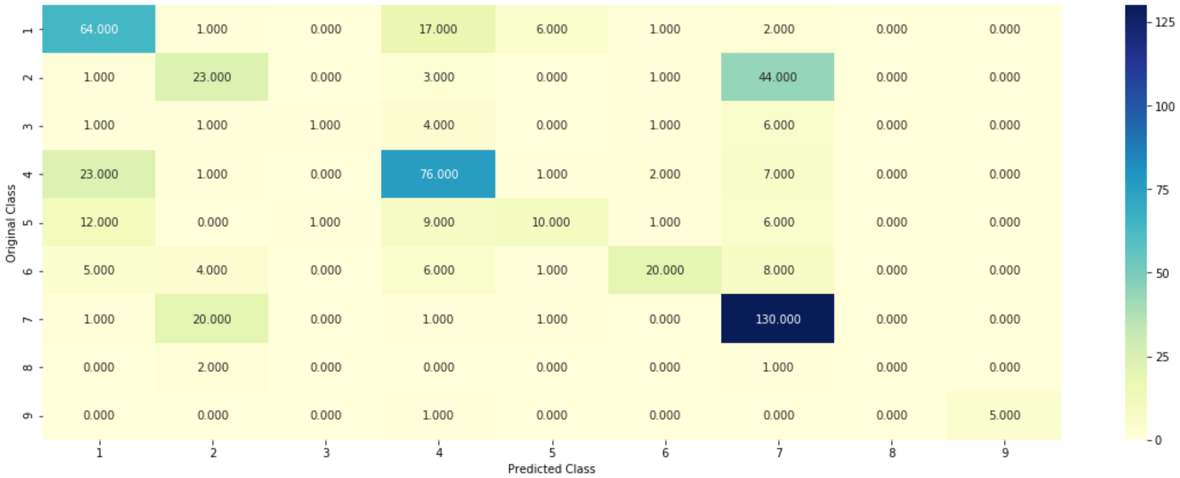



```
For values of best alpha =  0.0001 The train log loss is: 0.5665224692442247
For values of best alpha =  0.0001 The cross validation log loss is: 1.18610
84709067171
For values of best alpha =  0.0001 The test log loss is: 1.1086969230320527
```

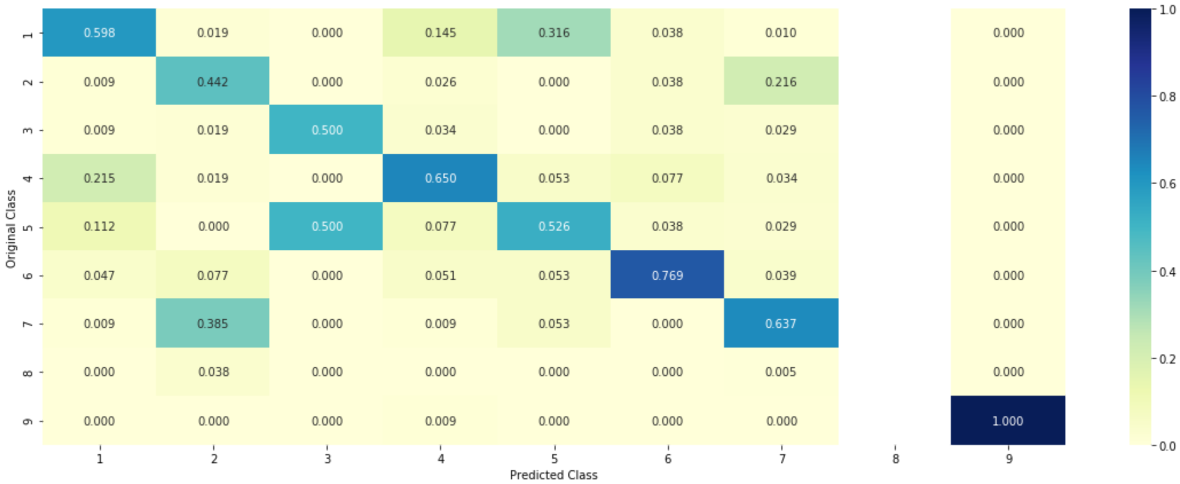
In [89]:

```
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, c
```

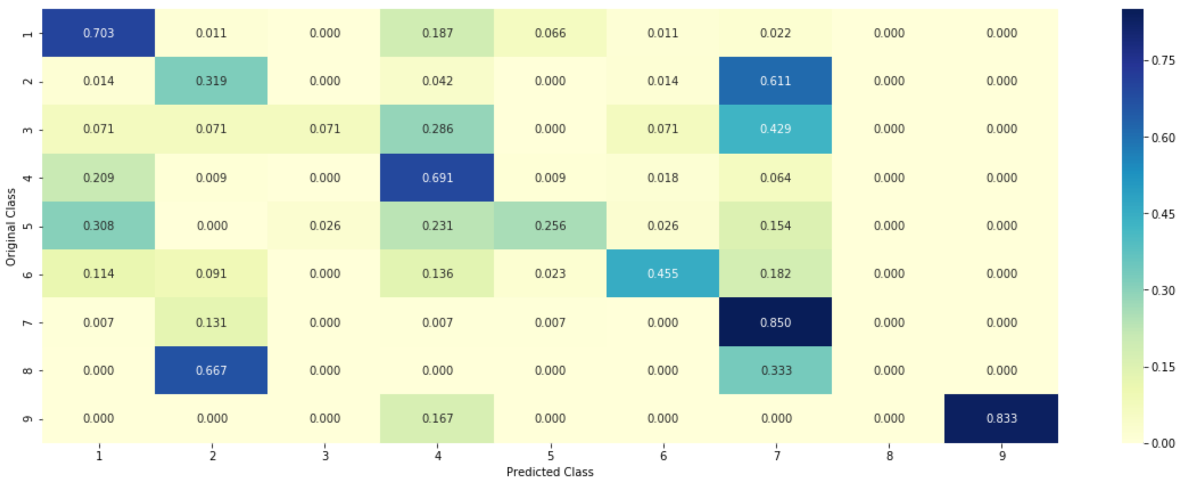
```
Log loss : 1.1861084709067171
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```




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




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

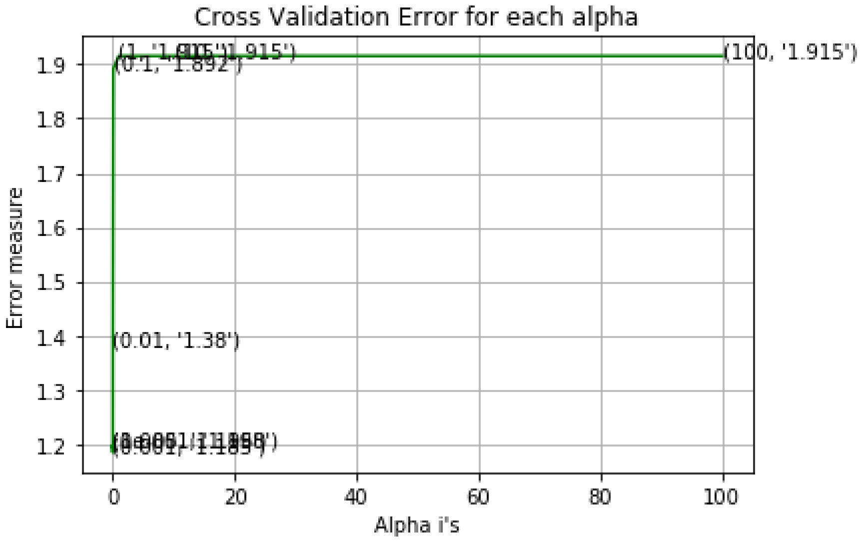


4.4. Linear Support Vector Machines

In [90]:

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

```
for C = 1e-05
Log Loss : 1.195166789723864
for C = 0.0001
Log Loss : 1.1981833906576886
for C = 0.001
Log Loss : 1.1847845392584273
for C = 0.01
Log Loss : 1.3798615200014308
for C = 0.1
Log Loss : 1.8920836975115085
for C = 1
Log Loss : 1.9153611765754375
for C = 10
Log Loss : 1.9153611869315808
for C = 100
Log Loss : 1.9153611857744721
```

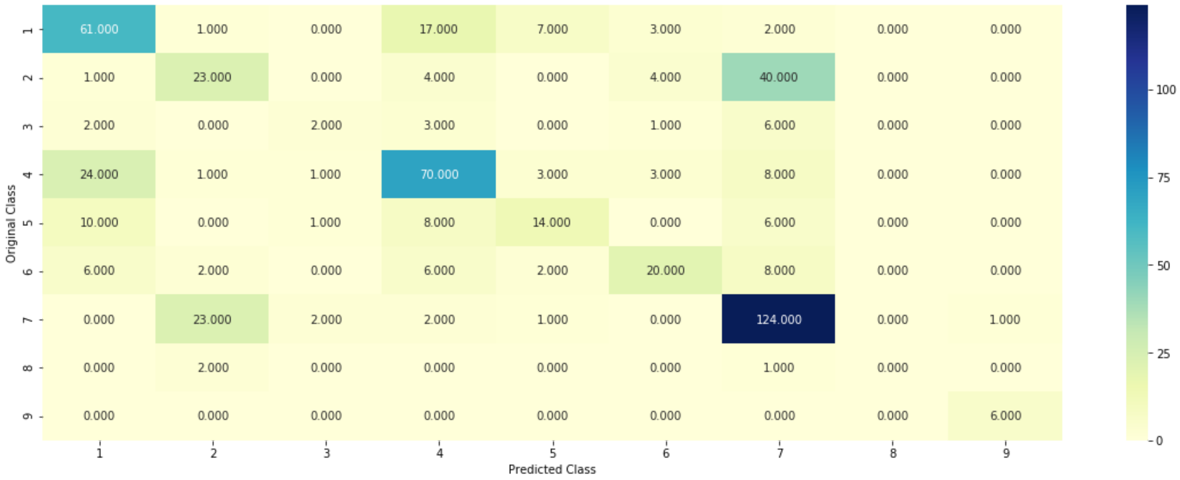



```
For values of best alpha =  0.001 The train log loss is: 0.7812519482232537
For values of best alpha =  0.001 The cross validation log loss is: 1.184784
5392584273
For values of best alpha =  0.001 The test log loss is: 1.1560024417929562
```

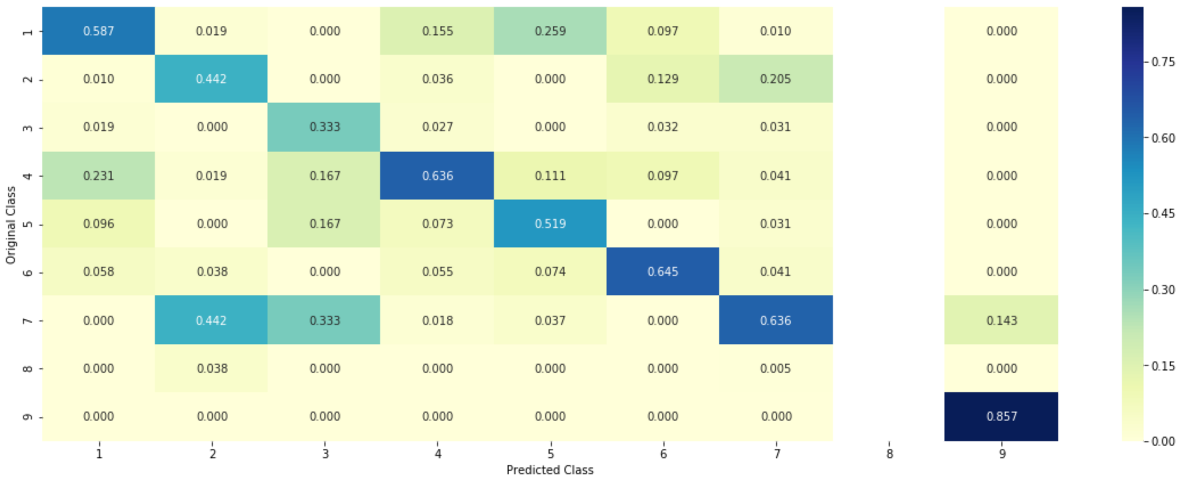
In [91]:

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

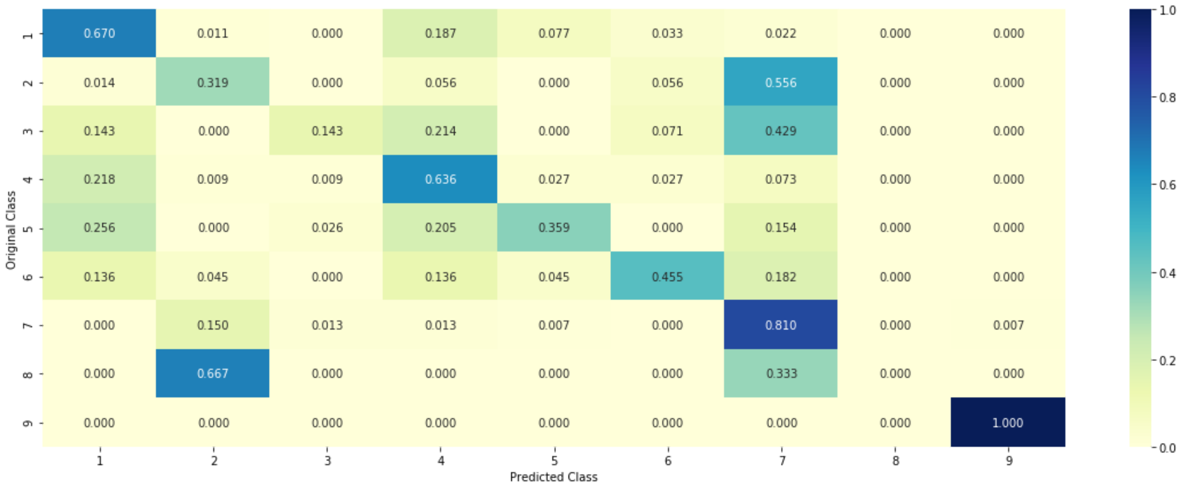
```
Log loss : 1.1847845392584273
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```




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




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



4.5 Random Forest Classifier

In [92]:

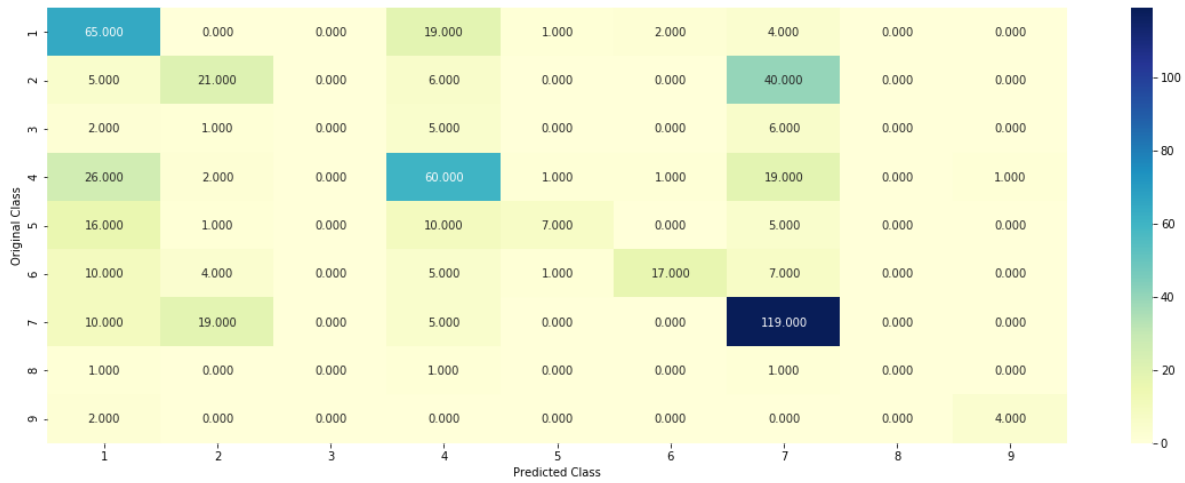
```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_
        clf.fit(train_x_onehotCoding, 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
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_d
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:"
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo
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:",
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.276950896235075
for n_estimators = 100 and max depth =  10
Log Loss : 1.2922881673737192
for n_estimators = 200 and max depth =  5
Log Loss : 1.2720465827942438
for n_estimators = 200 and max depth =  10
Log Loss : 1.2897973082271839
for n_estimators = 500 and max depth =  5
Log Loss : 1.2586339470536112
for n_estimators = 500 and max depth =  10
Log Loss : 1.2789983203102977
for n_estimators = 1000 and max depth =  5
Log Loss : 1.256521250360596
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2752243048198555
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2564246821102634
for n_estimators = 2000 and max depth =  10
Log Loss : 1.269875849032257
For values of best estimator =  2000 The train log loss is: 0.82697953273637
03
For values of best estimator =  2000 The cross validation log loss is: 1.256
4246821102634
For values of best estimator =  2000 The test log loss is: 1.214360015276142
5
```

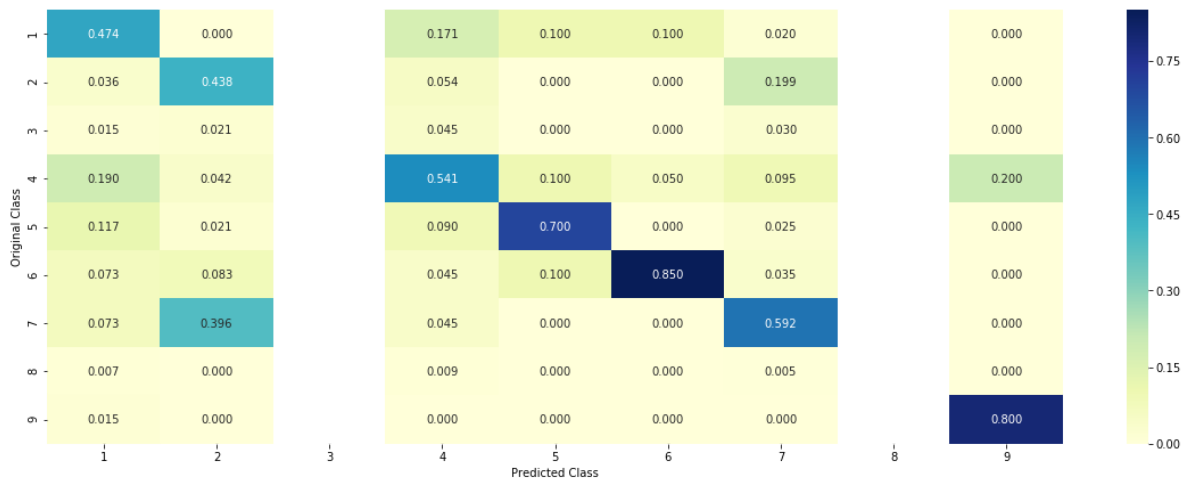
In [93]:

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

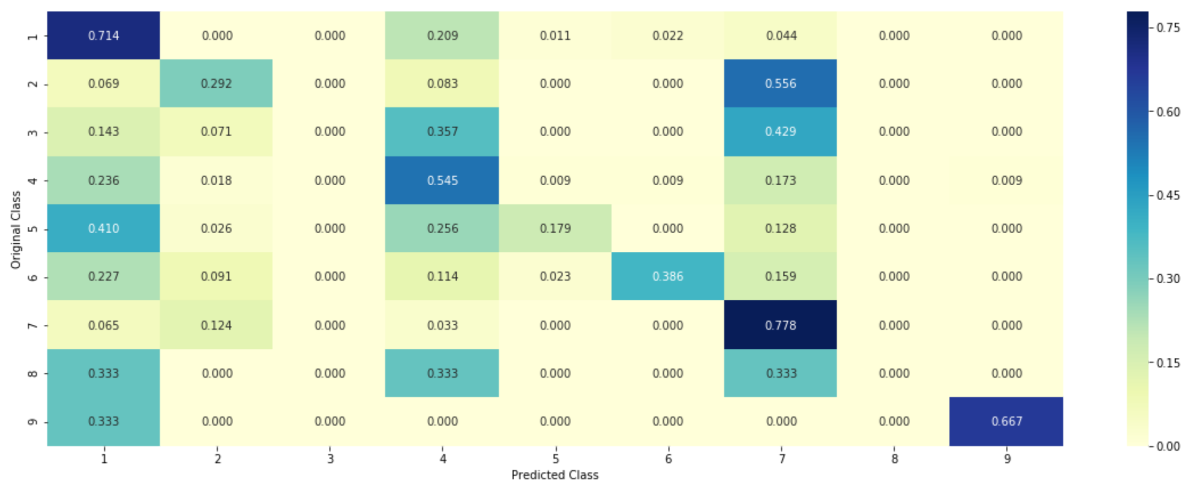
```
Log loss : 1.2564246821102634
Number of mis-classified points : 0.4492481203007519
-------------------- Confusion matrix --------------------
```



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



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



4.7 Stack the models

In [94]:

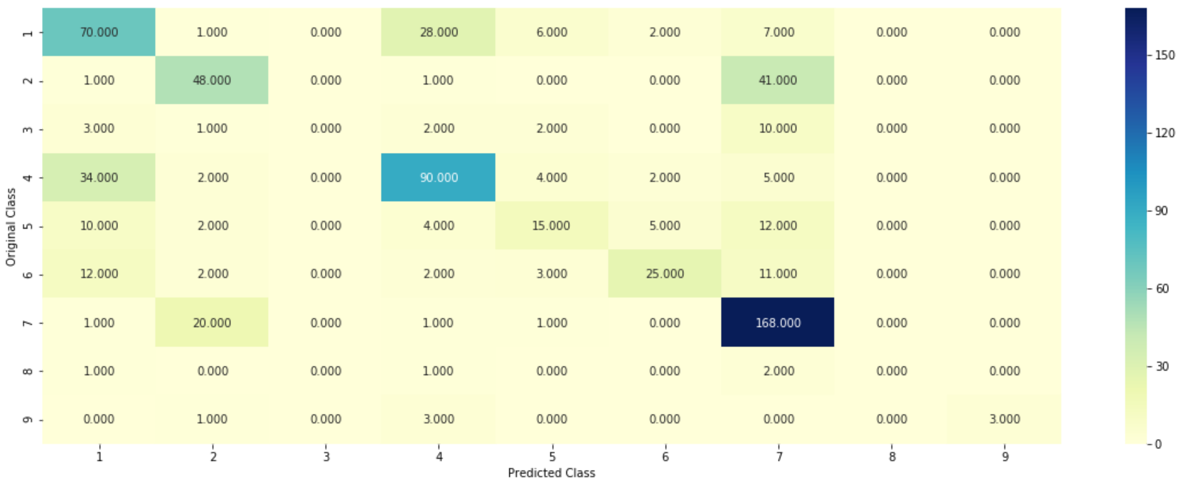
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', randon
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_s
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_
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(
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_onehotC
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=l
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(
    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: 1.14
Support vector machines : Log Loss: 1.92
Naive Bayes : Log Loss: 1.23
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.540
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.219
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.286
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.406
```

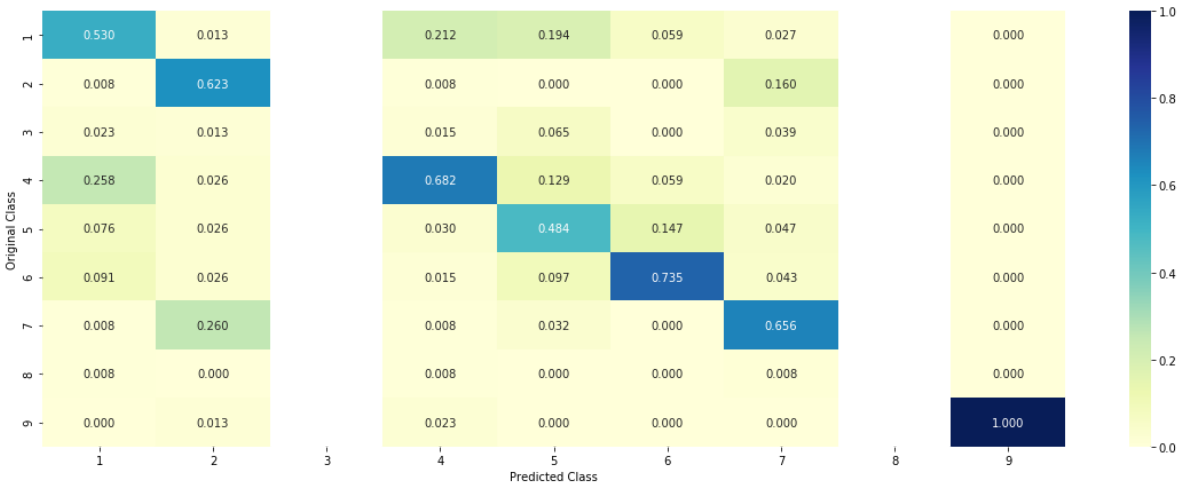
In [95]:

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

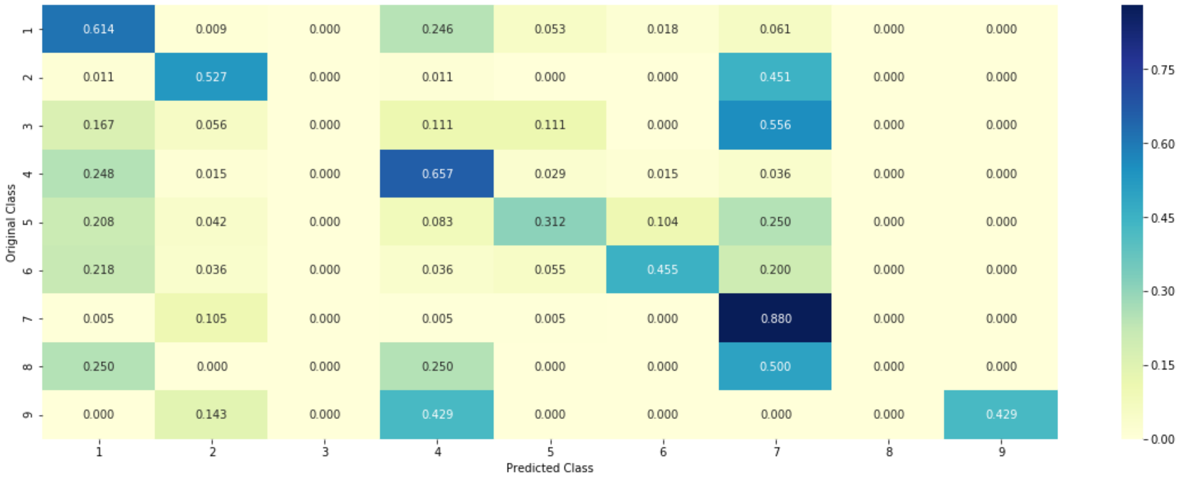
```
Log loss (train) on the stacking classifier : 0.7890907389887343
Log loss (CV) on the stacking classifier : 1.2189844996275194
Log loss (test) on the stacking classifier : 1.1390583577991233
Number of missclassified point : 0.36992481203007 52
-------------------- Confusion matrix --------------------
```




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



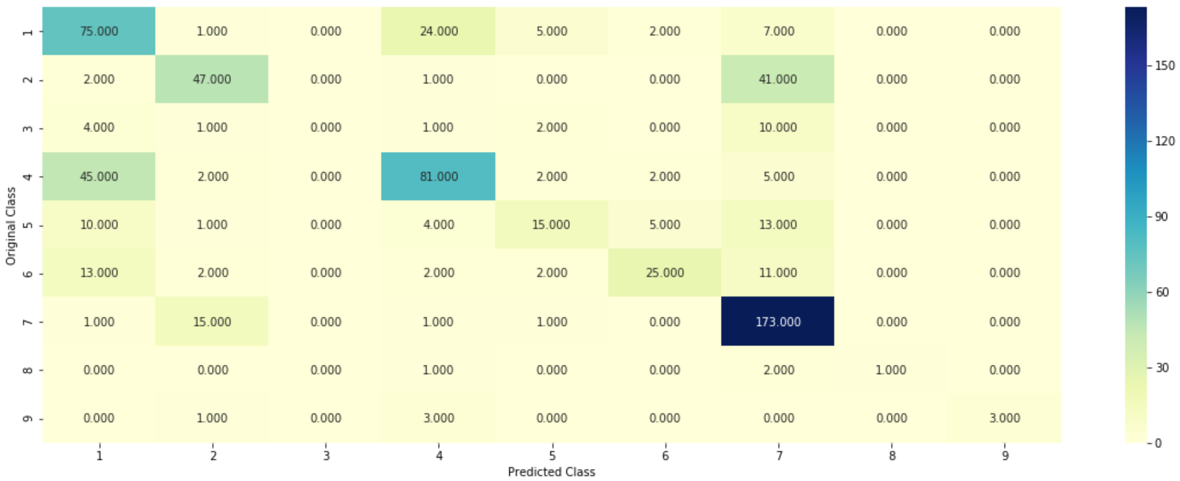

```
-------------------- Recall matrix (Row sum=1) --------------------
```


Original Class
Predicted Class
1
2
3
4
5
6
7
8
9
0.614 0.009 0.000 0.246 0.053 0.018 0.061 0.000 0.000
0.011 0.527 0.000 0.011 0.000 0.000 0.451 0.000 0.000
0.167 0.056 0.000 0.111 0.111 0.000 0.556 0.000 0.000
0.248 0.015 0.000 0.657 0.029 0.015 0.036 0.000 0.000
0.208 0.042 0.000 0.083 0.312 0.104 0.250 0.000 0.000
0.218 0.036 0.000 0.036 0.055 0.455 0.200 0.000 0.000
0.005 0.105 0.000 0.005 0.005 0.000 0.880 0.000 0.000
0.250 0.000 0.000 0.250 0.000 0.000 0.500 0.000 0.000
0.000 0.143 0.000 0.429 0.000 0.000 0.000 0.000 0.429
0.75
0.60
0.45
0.30
0.15
0.00

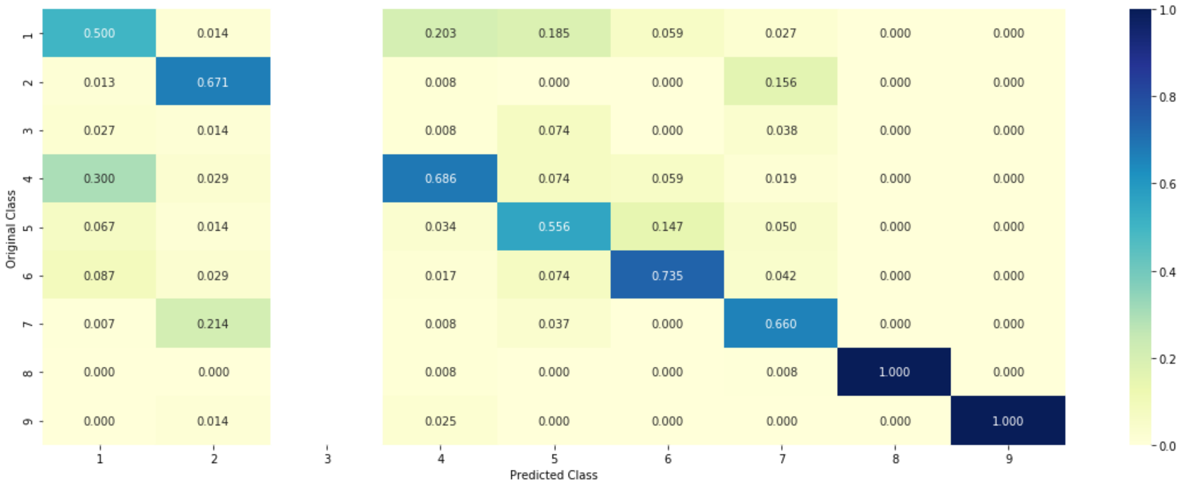
In [96]:

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

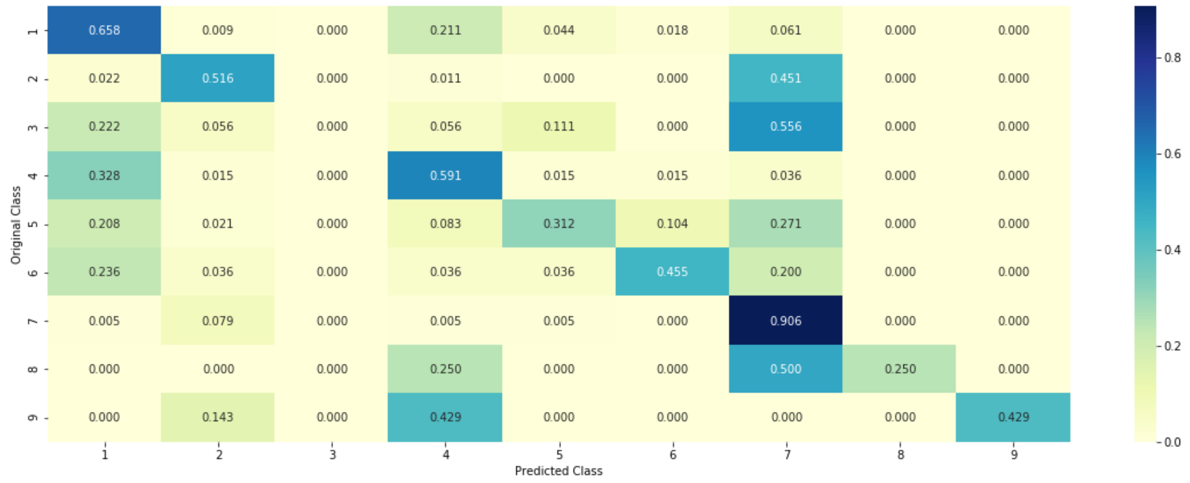
```
Log loss (train) on the VotingClassifier : 0.9362212754580596
Log loss (CV) on the VotingClassifier : 1.257414718324871
Log loss (test) on the VotingClassifier : 1.2065361398159882
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```




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




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



Summary

In [97]:

```
from prettytable import PrettyTable
t=PrettyTable()
t.field_names = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss']
t.add_row(['Naive Bayes', '0.72852', '1.23395', '1.16593'])
t.add_row(['Linear Regression with Class balancing', '0.78984', '1.14215', '1.08068'])
t.add_row(['Linear Regression without Class balancing', '0.56652', '1.18610', '1.10869'])
t.add_row(['Support Vector Machine', '0.78125', '1.18478', '1.15600'])
t.add_row(['Random Forest', '0.82697', '1.25642', '1.21436'])
t.add_row(['Stacking Classifier', '0.78909', '1.21898', '1.13905'])
t.add_row(['Maximum Voting Classifier', '0.93622', '1.25741', '1.20653'])

print(t)
```

```
+-------------------------------------------+----------------+-------------+
---------------+
|                   Model                   | Train Log Loss | CV Log Loss |
Test Log Loss |
+-------------------------------------------+----------------+-------------+
---------------+
|                Naive Bayes                |    0.72852     |   1.23395   |
1.16593     |
|   Linear Regression with Class balancing  |    0.78984     |   1.14215   |
1.08068     |
| Linear Regression without Class balancing |    0.56652     |   1.18610   |
1.10869     |
|           Support Vector Machine          |    0.78125     |   1.18478   |
1.15600     |
|               Random Forest               |    0.82697     |   1.25642   |
1.21436     |
|            Stacking Classifier            |    0.78909     |   1.21898   |
1.13905     |
|         Maximum Voting Classifier         |    0.93622     |   1.25741   |
1.20653     |
+-------------------------------------------+----------------+-------------+
---------------+
```

Part 3 - Apply Logistic Regression with CountVectorizer (unigrams & bigrams)

3.2.1 Univariate Analysis on Gene Feature

In [99]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
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 [100]:

```
gene_vectorizer.get_feature_names()
```

```
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
```

In [101]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 227)
```

3.2.2 Univariate Analysis on Variation Feature

In [102]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variati
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [103]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-ho
t encoding method. The shape of Variation feature: (2124, 2082)
```

3.2.3 Univariate Analysis on Text Feature

In [104]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3, ngram_range=(1, 2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number o
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 occu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

```
Total number of unique words in train data : 759809
```

In [105]:

```
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 [106]:

```
# 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()))
```

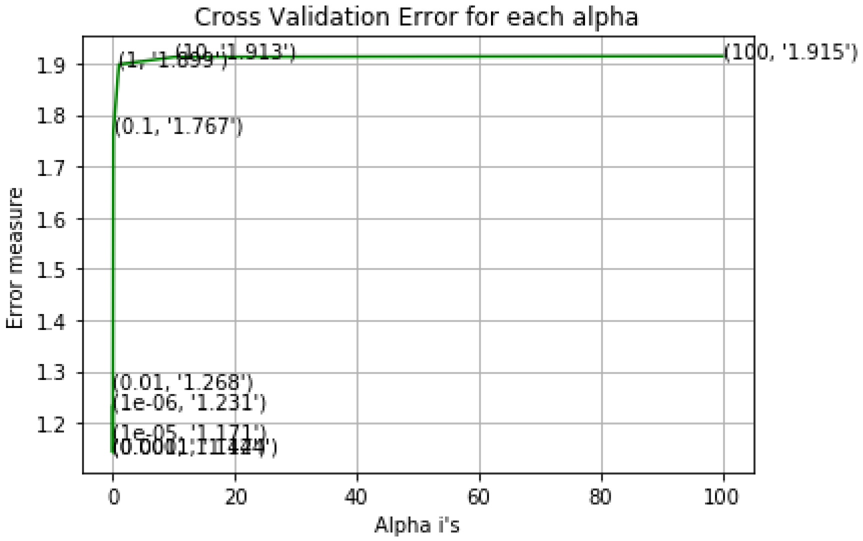
Logistic Regression

With Class balancing

In [107]:

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

```
for alpha = 1e-06
Log Loss : 1.230993092908709
for alpha = 1e-05
Log Loss : 1.1706380135890329
for alpha = 0.0001
Log Loss : 1.143971044997503
for alpha = 0.001
Log Loss : 1.1421580146097203
for alpha = 0.01
Log Loss : 1.2683535032774222
for alpha = 0.1
Log Loss : 1.7674110224663384
for alpha = 1
Log Loss : 1.8985997685648925
for alpha = 10
Log Loss : 1.9130995776121724
for alpha = 100
Log Loss : 1.9148451464968697
```

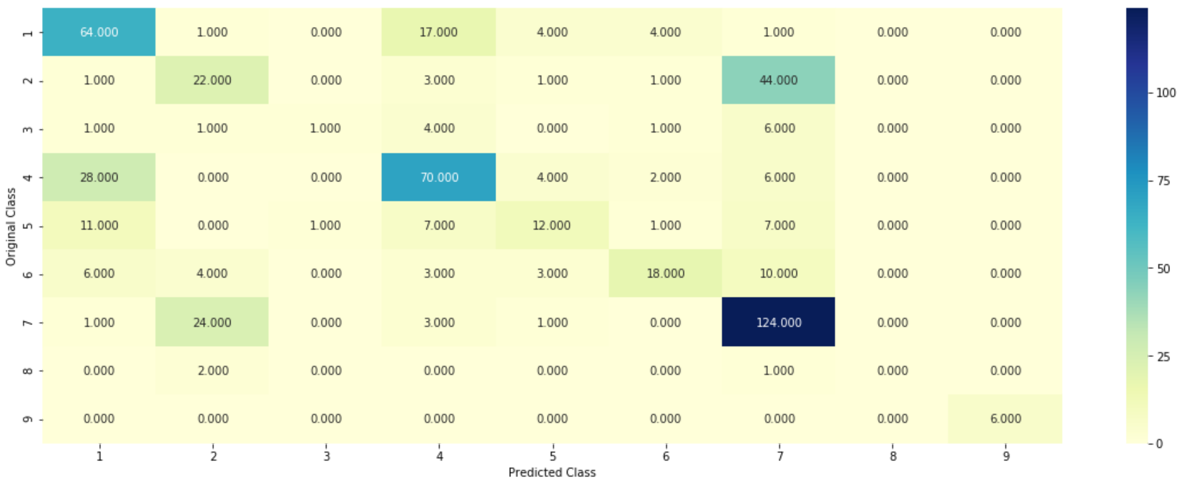



```
For values of best alpha =  0.001 The train log loss is: 0.7898475305677636
For values of best alpha =  0.001 The cross validation log loss is: 1.142158
0146097203
For values of best alpha =  0.001 The test log loss is: 1.0806870264944455
```

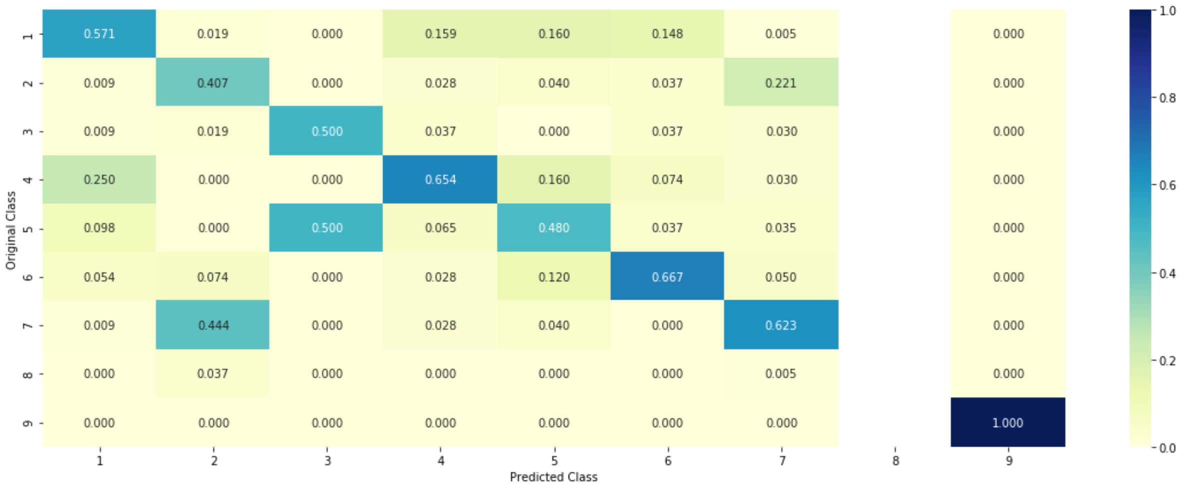
In [108]:

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

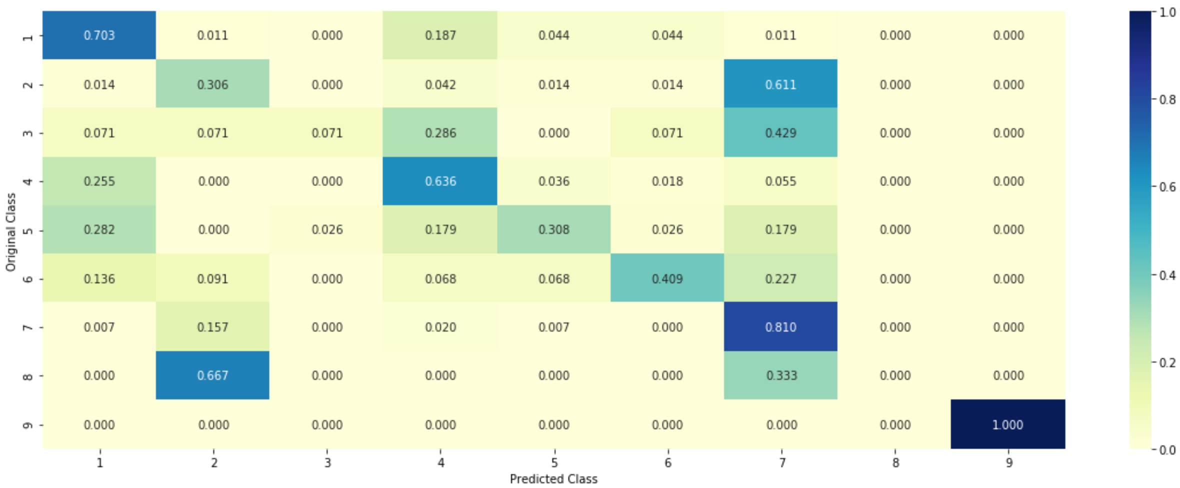
```
Log loss : 1.1421580146097203
Number of mis-classified points : 0.40413533834586447
-------------------- Confusion matrix --------------------
```




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




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

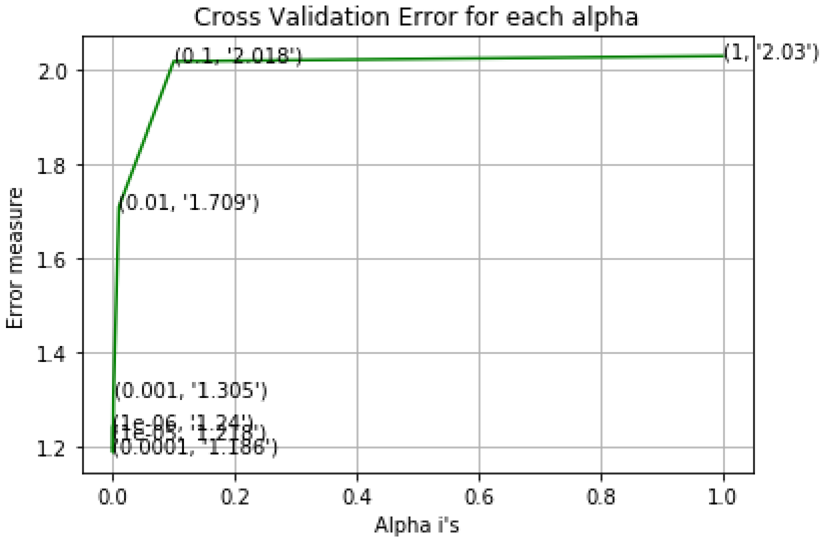


Without Class balancing

In [109]:

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

```
for alpha = 1e-06
Log Loss : 1.2404488470608344
for alpha = 1e-05
Log Loss : 1.2181349796192469
for alpha = 0.0001
Log Loss : 1.1861084709067171
for alpha = 0.001
Log Loss : 1.3049478138782458
for alpha = 0.01
Log Loss : 1.7088649752297245
for alpha = 0.1
Log Loss : 2.018488583320742
for alpha = 1
Log Loss : 2.0298755346633546
```

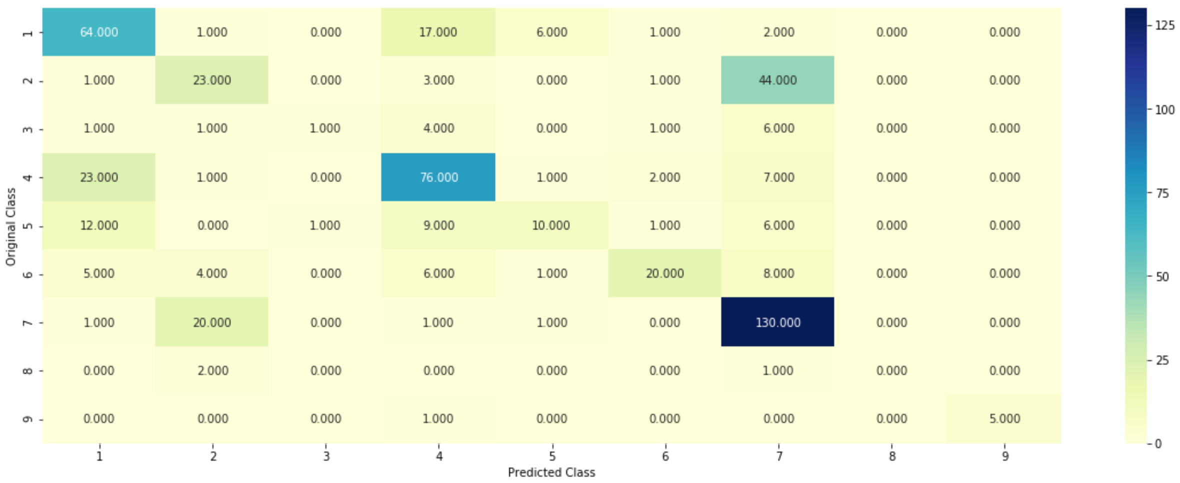



```
For values of best alpha =  0.0001 The train log loss is: 0.5665224692442247
For values of best alpha =  0.0001 The cross validation log loss is: 1.18610
84709067171
For values of best alpha =  0.0001 The test log loss is: 1.1086969230320527
```

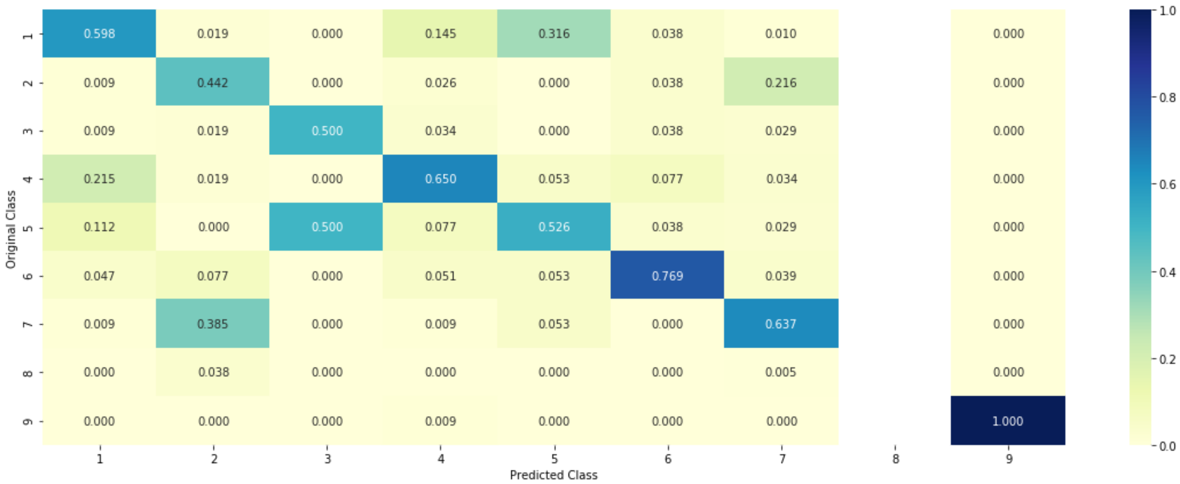
In [110]:

```
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, c
```

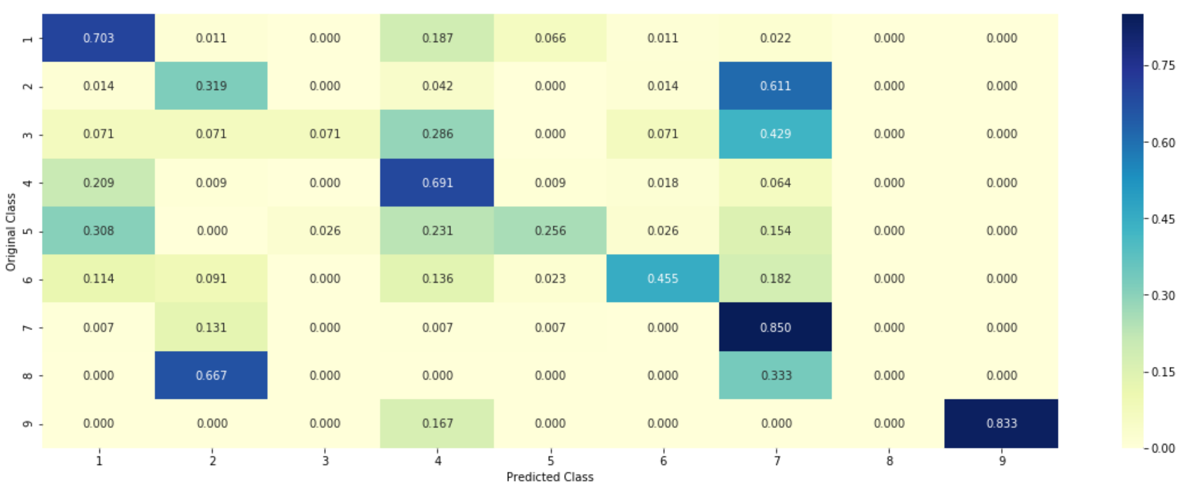
```
Log loss : 1.1861084709067171
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```




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




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



Part 4 - Feature Engineering

Techniques obtained from these kernels/blogs.

1. https://www.kaggle.com/osciiart/redefining-treatment-0-57456-modified (https://www.kaggle.com/osciiart/redefining-treatment-0-57456-modified)
2. https://www.kaggle.com/lalitparihar44/detailed-text-based-feature-engineering (https://www.kaggle.com/lalitparihar44/detailed-text-based-feature-engineering)
3. https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/ (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/)

Gene + Variation Feature

In [92]:

```
result['Gene_Variation'] = result['Gene'] + " " + result["Variation"]
result.head()
```

Out[92]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V

Count of Words Feature

In [93]:

```
result["Word_Count"] = result["TEXT"].apply(lambda x: len(x.split()))
result.head()
```

Out[93]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Word_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	6089
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	5722
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	5722
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	5572
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	6202

Character Count Feature

In [94]:

```
result['Character_Count'] = result['TEXT'].apply(lambda x: len(str(x)))
result.head()
```

Out[94]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Word_Count	Character_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	6089	39765
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	5722	36831
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	5722	36831
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	5572	36308
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	6202	41427

Gene Count Feature

In [95]:

```
result['Gene_Share'] = result.apply(lambda r: sum([1 for w in r['Gene'].split() if w in r['
result.head()
```

Out[95]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Word_Count	Character_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	6089	39765
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	5722	36831
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	5722	36831
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	5572	36308
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	6202	41427

Variation Count Feature

In [96]:

```
result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['Variation'].split(' '
result["Variation_Share"].value_counts()
```

Out[96]:

```
1    1676
0    1572
2      59
3      10
5       2
4       2
Name: Variation_Share, dtype: int64
```

Text Count > 5000 Yes or no feature

In [97]:

```
result["Word_Count_5000"] = result["Word_Count"].apply(lambda x: 1 if x > 5000 else 0)
result.head()
```

Out[97]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Word_Count	Character_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	6089	39765
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	5722	36831
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	5722	36831
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	5572	36308
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	6202	41427

Average Length of Words used in statements

In [98]:

```
result['Avg_length'] = result['Character_Count'] / result['Word_Count']
result.head()
```

Out[98]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Word_Count	Character_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	6089	39765
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	5722	36831
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	5722	36831
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	5572	36308
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	6202	41427

Preprocessing Text

In [99]:

```
# 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 [100]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 317.75504367599933 seconds
```

In [101]:

```
#removing unprocessed "TEXT" from results
result.drop("TEXT", axis=1, inplace=True)
```

In [102]:

```
# Joining Text which is processed :
result = pd.merge(result, data_text,on='ID', how='left')
result.head()
```

Out[102]:

	ID	Gene	Variation	Class	Gene_Variation	Word_Count	Character_Count	Gene_Share
0	0	FAM58A	Truncating Mutations	1	FAM58A Truncating Mutations	6089	39765	1
1	1	CBL	W802*	2	CBL W802*	5722	36831	1
2	2	CBL	Q249E	2	CBL Q249E	5722	36831	1
3	3	CBL	N454D	3	CBL N454D	5572	36308	1
4	4	CBL	L399V	4	CBL L399V	6202	41427	1

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

In [103]:

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

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

In [104]:

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

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

Encoding Gene Feature

In [105]:

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

In [106]:

```
print(train_gene_feature_onehotCoding.shape)
print(test_gene_feature_onehotCoding.shape)
print(cv_gene_feature_onehotCoding.shape)
```

```
(2124, 229)
(665, 229)
(532, 229)
```

Encoding Variation Feature

In [107]:

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

In [108]:

```
print(train_variation_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
(2124, 1977)
(665, 1977)
(532, 1977)
```

Encoding Text Feature

In [110]:

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

In [113]:

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

In [111]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number o
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 occu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

In [114]:

```
# 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 [115]:

```
print(train_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)
```

```
(2124, 5000)
(665, 5000)
(532, 5000)
```

Encoding Gene_Variation Feature

In [116]:

```
# one-hot encoding of gene_and_variation feature.
gene_var_vectorizer = CountVectorizer()
train_gene_var_feature_onehotCoding = gene_var_vectorizer.fit_transform(train_df["Gene_Vari
test_gene_var_feature_onehotCoding = gene_var_vectorizer.transform(test_df["Gene_Variation"
cv_gene_var_feature_onehotCoding = gene_var_vectorizer.transform(cv_df["Gene_Variation"])
```

In [117]:

```
print(train_gene_var_feature_onehotCoding.shape)
print(test_gene_var_feature_onehotCoding.shape)
print(cv_gene_var_feature_onehotCoding.shape)
```

```
(2124, 2171)
(665, 2171)
(532, 2171)
```

Stacking all the features

In [118]:

```
train_1_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_oneh
test_1_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotC
cv_1_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)

train_2_onehotCoding = hstack((train_gene_var_feature_onehotCoding,train_text_feature_onehc
test_2_onehotCoding = hstack((test_gene_var_feature_onehotCoding,test_text_feature_onehotCc
cv_2_onehotCoding = hstack((cv_gene_var_feature_onehotCoding,cv_text_feature_onehotCoding))

train_x_onehotCoding = hstack((train_1_onehotCoding, train_2_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_1_onehotCoding, test_2_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_1_onehotCoding, cv_2_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [119]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 9377)
(number of data points * number of features) in test data =  (665, 9377)
(number of data points * number of features) in cross validation data = (53
2, 9377)
```

In [120]:

```
# extracting the numerical features from the train,test & cv datsets
train_inter = train_df[['Gene_Share', 'Variation_Share', 'Word_Count', 'Character_Count', '
test_inter = test_df[['Gene_Share', 'Variation_Share', 'Word_Count', 'Character_Count', 'Wc
cv_inter = cv_df[['Gene_Share', 'Variation_Share', 'Word_Count', 'Character_Count', 'Word_C
```

In [121]:

```
print(train_inter.shape)
print(test_inter.shape)
print(cv_inter.shape)
```

```
(2124, 6)
(665, 6)
(532, 6)
```

In [122]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
train_inter["Word_Count"] = scaler.fit_transform(train_inter["Word_Count"].values.reshape(-
test_inter["Word_Count"] = scaler.fit_transform(test_inter["Word_Count"].values.reshape(-1,
cv_inter["Word_Count"] = scaler.fit_transform(cv_inter["Word_Count"].values.reshape(-1,1))

train_inter["Character_Count"] = scaler.fit_transform(train_inter["Character_Count"].values
test_inter["Character_Count"] = scaler.fit_transform(test_inter["Character_Count"].values.r
cv_inter["Character_Count"] = scaler.fit_transform(cv_inter["Character_Count"].values.resha

train_inter["Avg_length"] = scaler.fit_transform(train_inter["Avg_length"].values.reshape(-
test_inter["Avg_length"] = scaler.fit_transform(test_inter["Avg_length"].values.reshape(-1,
cv_inter["Avg_length"] = scaler.fit_transform(cv_inter["Avg_length"].values.reshape(-1,1))
```

In [123]:

```
x_train_final = hstack((train_x_onehotCoding, train_inter)).tocsr()
x_test_final = hstack((test_x_onehotCoding, test_inter)).tocsr()
x_cv_final = hstack((cv_x_onehotCoding, cv_inter)).tocsr()
```

In [124]:

```
print(x_train_final.shape)
print(x_test_final.shape)
print(x_cv_final.shape)
```

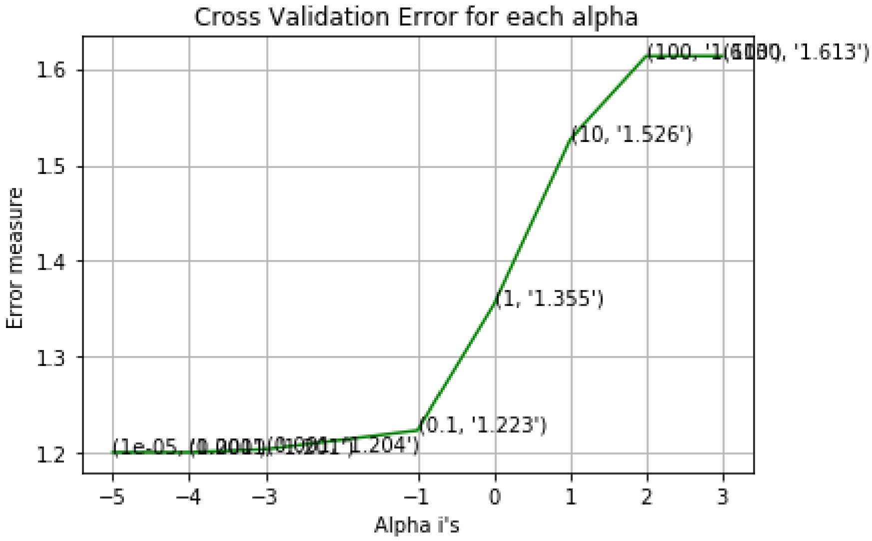
```
(2124, 9383)
(665, 9383)
(532, 9383)
```

Naive Bayes

In [125]:

```
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(x_train_final, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_final, train_y)
    sig_clf_probs = sig_clf.predict_proba(x_cv_final)
    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 estimat
    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(x_train_final, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_final, train_y)


predict_y = sig_clf.predict_proba(x_train_final)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
```

```
for alpha = 1e-05
Log Loss : 1.2005942881445657
for alpha = 0.0001
Log Loss : 1.2005536684714955
for alpha = 0.001
Log Loss : 1.20379856337728
for alpha = 0.1
Log Loss : 1.2234438452675278
for alpha = 1
Log Loss : 1.3546696333205988
for alpha = 10
Log Loss : 1.5263585050089978
for alpha = 100
Log Loss : 1.613260995326949
for alpha = 1000
Log Loss : 1.613353691234811
```



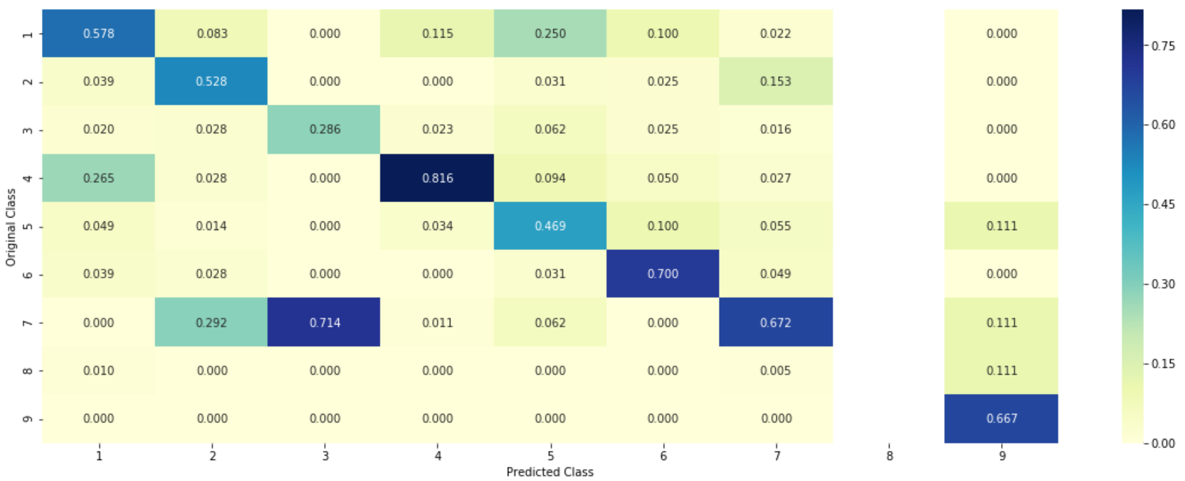

```
For values of best alpha =  0.0001 The train log loss is: 0.6488219378182999
For values of best alpha =  0.0001 The cross validation log loss is: 1.20055
36684714955
For values of best alpha =  0.0001 The test log loss is: 1.275970020412448
```

In [126]:

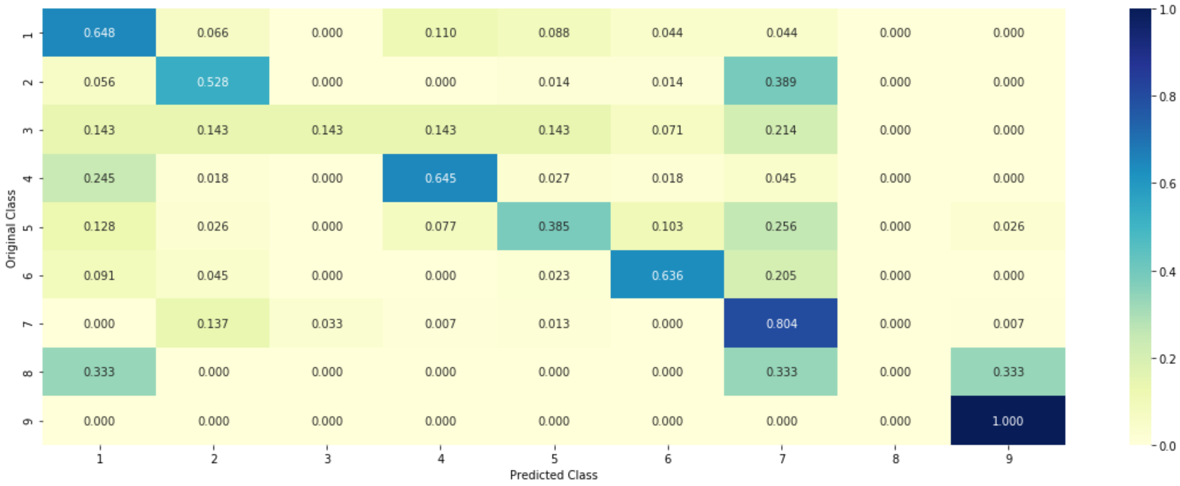
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(x_train_final, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_final, train_y)
sig_clf_probs = sig_clf.predict_proba(x_cv_final)
# to avoid rounding error while multiplying probabilites we use log-probability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

```
Log Loss : 1.2005536684714955
Number of missclassified point : 0.35714285714285715
-------------------- Confusion matrix --------------------
```

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



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



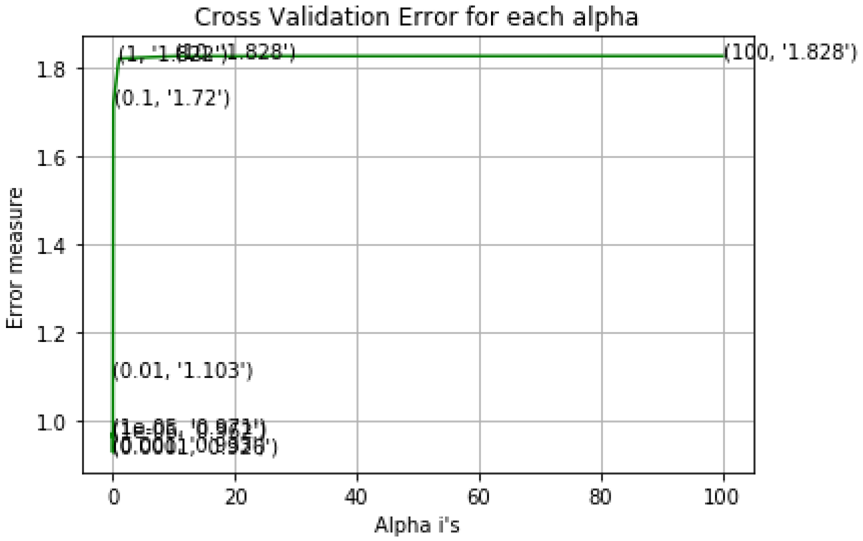
Logistic Regression

With Class balancing

In [127]:

```
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_
    clf.fit(x_train_final, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_final, train_y)
    sig_clf_probs = sig_clf.predict_proba(x_cv_final)
    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 estimat
    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='l
clf.fit(x_train_final, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_final, train_y)

predict_y = sig_clf.predict_proba(x_train_final)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
```

```
for alpha = 1e-06
Log Loss : 0.961851274796587
for alpha = 1e-05
Log Loss : 0.9714010834840572
for alpha = 0.0001
Log Loss : 0.926033615143531
for alpha = 0.001
Log Loss : 0.9331951360732824
for alpha = 0.01
Log Loss : 1.1026621960362821
for alpha = 0.1
Log Loss : 1.7196569930366807
for alpha = 1
Log Loss : 1.8224340770238423
for alpha = 10
Log Loss : 1.8278524364050628
for alpha = 100
Log Loss : 1.8282927241422788
```

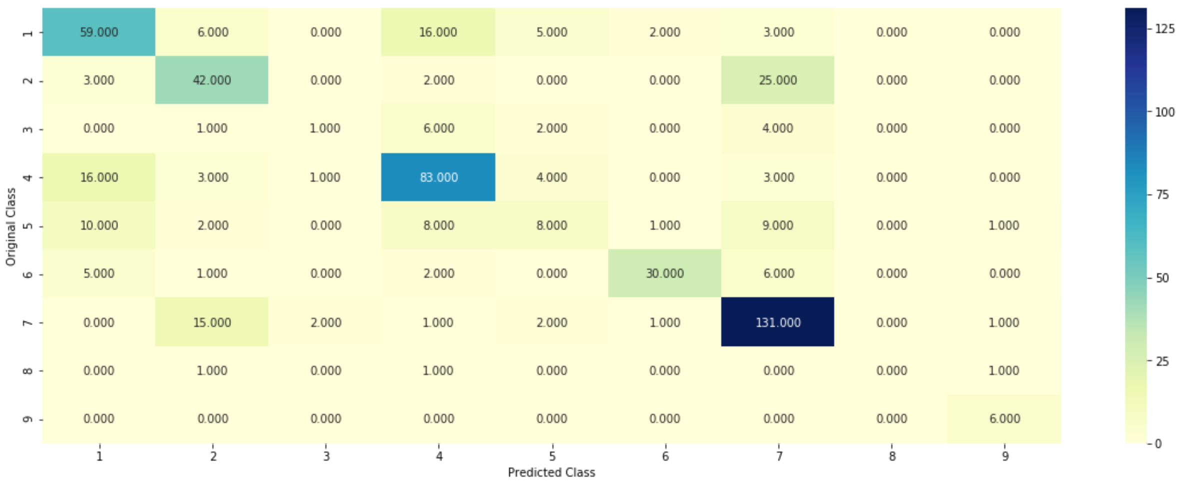



```
For values of best alpha =  0.0001 The train log loss is: 0.4205141466653583
For values of best alpha =  0.0001 The cross validation log loss is: 0.92603
3615143531
For values of best alpha =  0.0001 The test log loss is: 0.9695918612372099
```

In [128]:

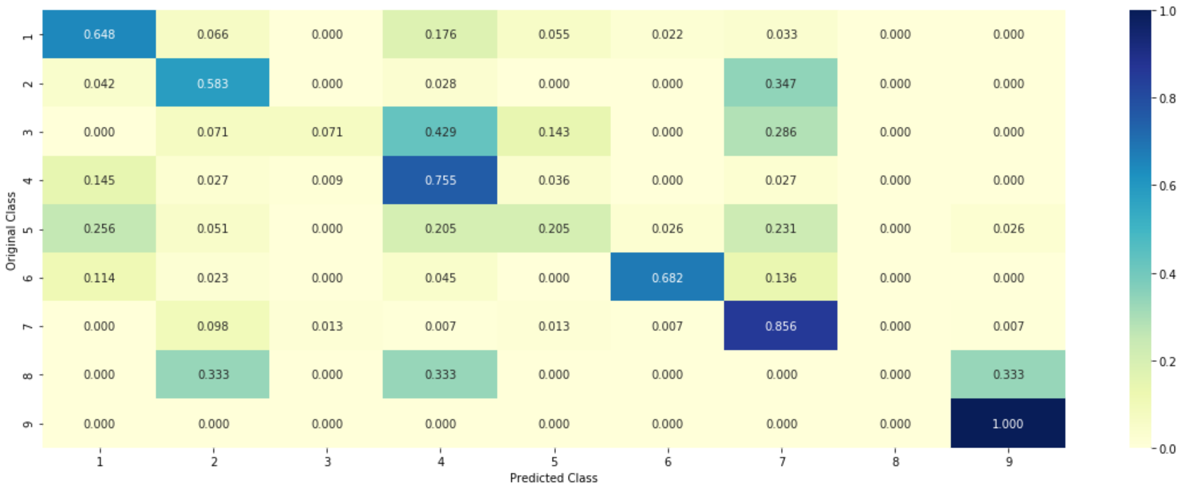
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(x_train_final, train_y, x_cv_final, cv_y, clf)
```

```
Log loss : 0.926033615143531
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```




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


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

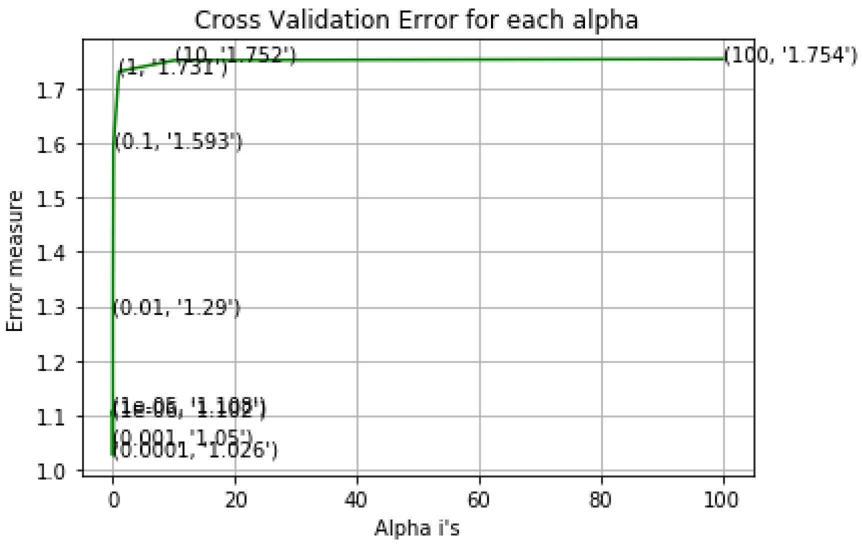


Without Class balancing

In [129]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15)
    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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
```

```
for alpha = 1e-06
Log Loss : 1.1019964220986735
for alpha = 1e-05
Log Loss : 1.1079838578204324
for alpha = 0.0001
Log Loss : 1.026390275169807
for alpha = 0.001
Log Loss : 1.0499965519543142
for alpha = 0.01
Log Loss : 1.2895384556612974
for alpha = 0.1
Log Loss : 1.5931125028778481
for alpha = 1
Log Loss : 1.7312296425557345
for alpha = 10
Log Loss : 1.751796397375641
for alpha = 100
Log Loss : 1.7541608606606152
```

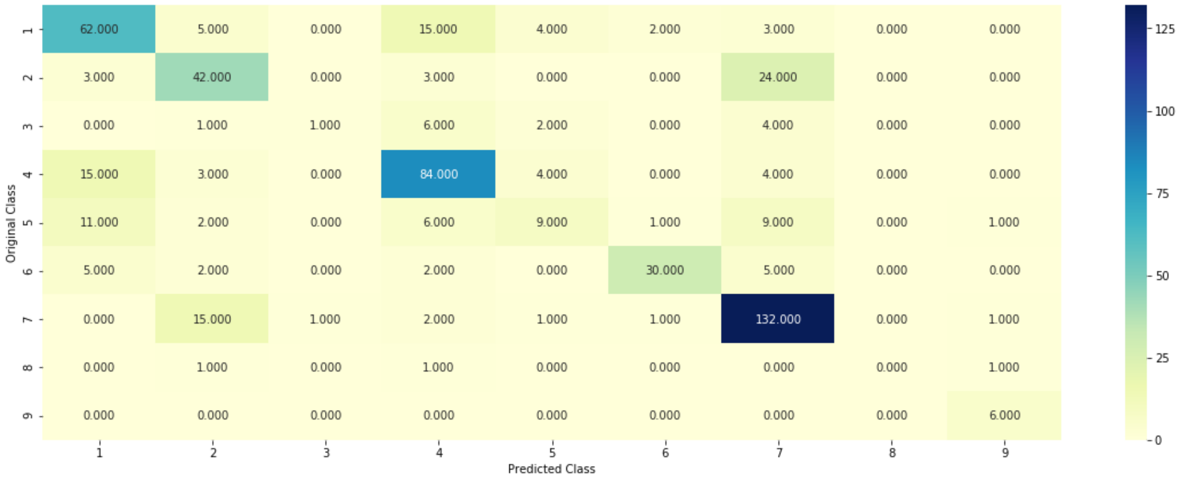



```
For values of best alpha =  0.0001 The train log loss is: 0.4096339530367641
For values of best alpha =  0.0001 The cross validation log loss is: 1.02639
0275169807
For values of best alpha =  0.0001 The test log loss is: 1.0476440640833955
```

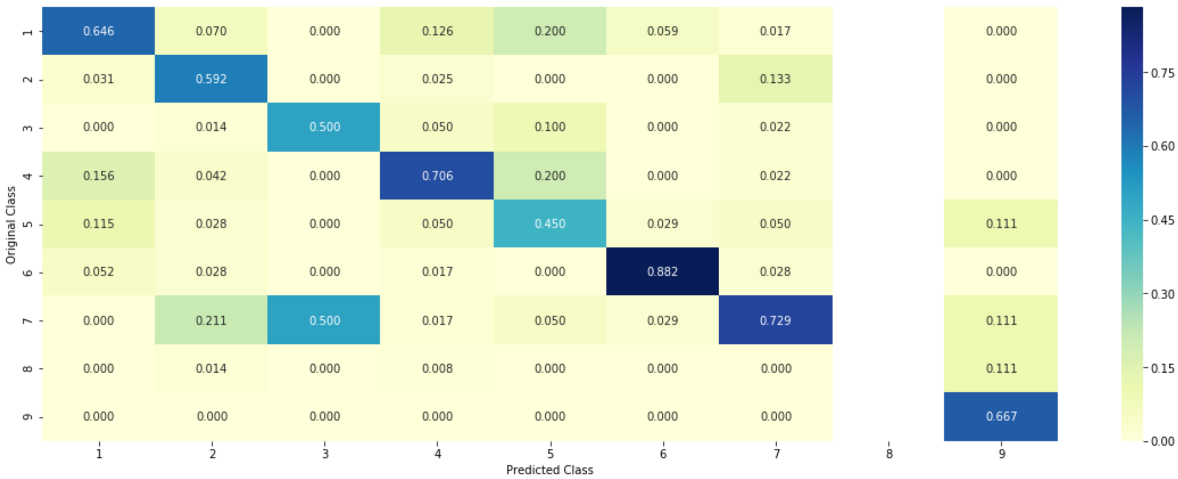
In [130]:

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

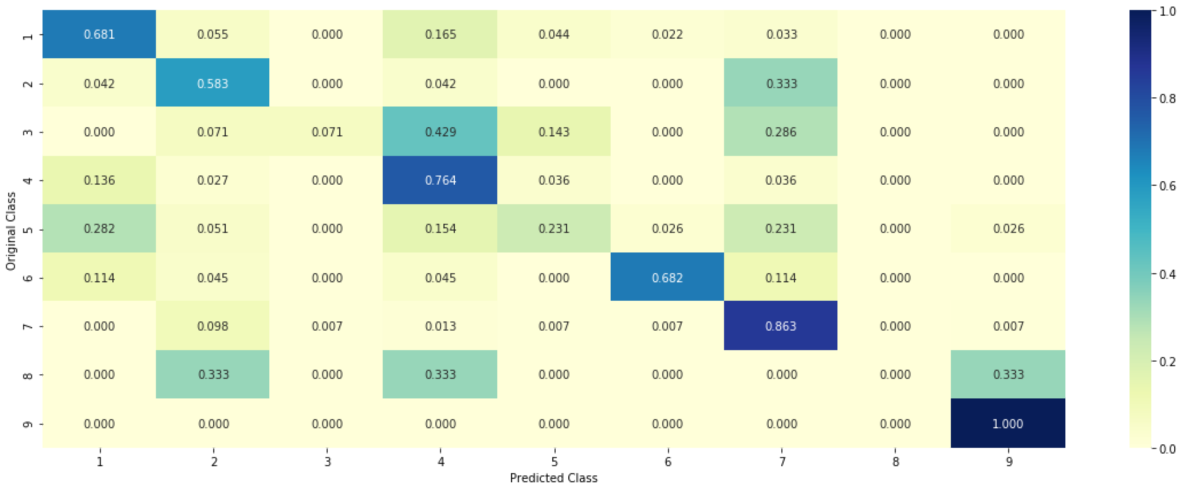
```
Log loss : 0.9397411015276902
Number of mis-classified points : 0.31203007518796994
-------------------- Confusion matrix --------------------
```




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




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

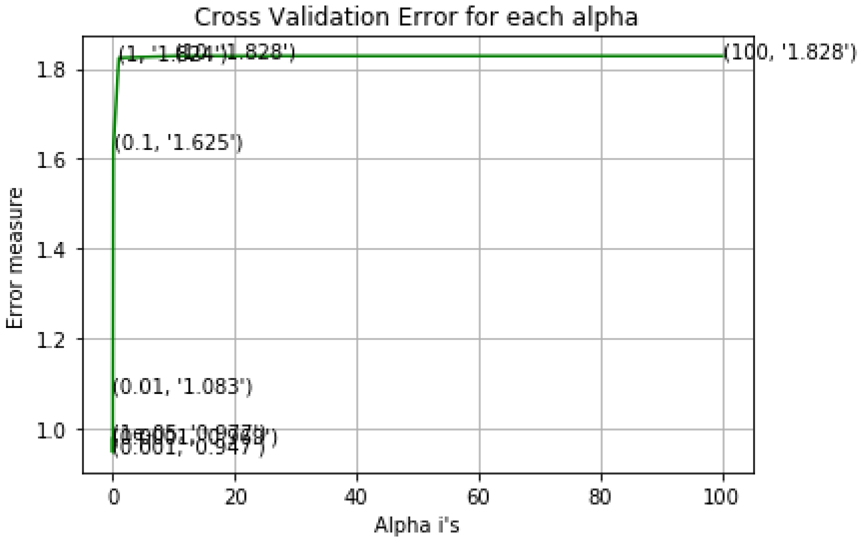


Linear Support Vector Machines

In [131]:

```
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', ranc
    clf.fit(x_train_final, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_final, train_y)
    sig_clf_probs = sig_clf.predict_proba(x_cv_final)
    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='h
clf.fit(x_train_final, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_final, train_y)

predict_y = sig_clf.predict_proba(x_train_final)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
```

```
for C = 1e-05
Log Loss : 0.9774871096008211
for C = 0.0001
Log Loss : 0.96883106269444
for C = 0.001
Log Loss : 0.9469936254962772
for C = 0.01
Log Loss : 1.0832525099212287
for C = 0.1
Log Loss : 1.625425333855736
for C = 1
Log Loss : 1.824049673447288
for C = 10
Log Loss : 1.828346847051547
for C = 100
Log Loss : 1.8283466770040466
```

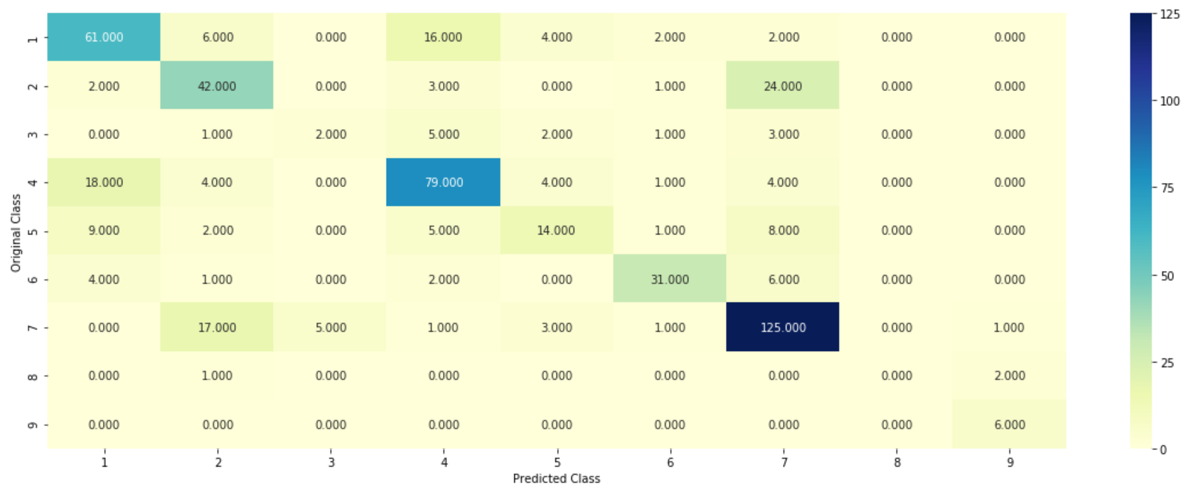



```
For values of best alpha =  0.001 The train log loss is: 0.471648467856095554
For values of best alpha =  0.001 The cross validation log loss is: 0.946993
6254962772
For values of best alpha =  0.001 The test log loss is: 1.0044844554029406
```

In [132]:

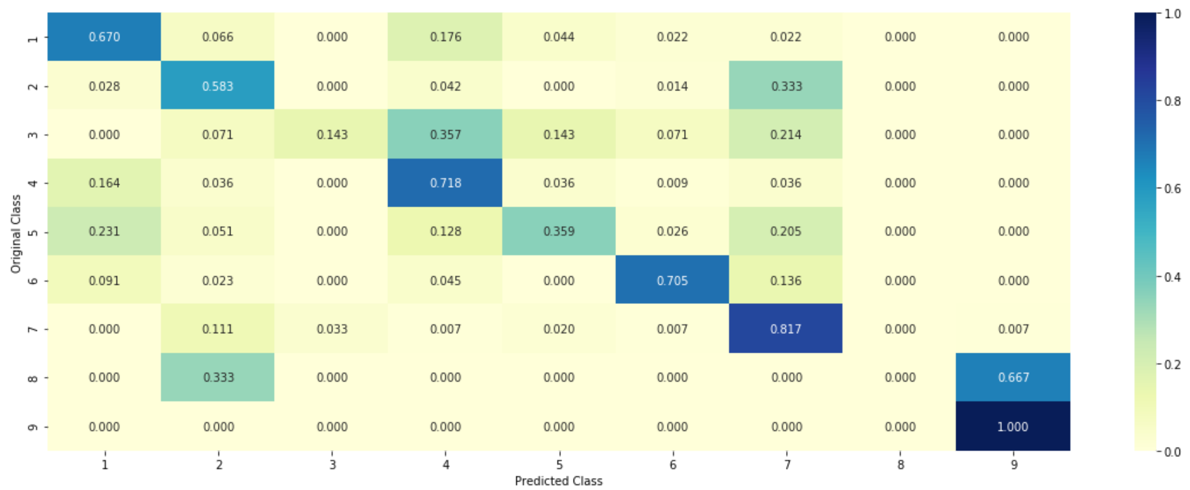
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42,cl
predict_and_plot_confusion_matrix(x_train_final, train_y, x_cv_final, cv_y, clf)
```

```
Log loss : 0.9469936254962772
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```




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


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



Random Forest Classifier

In [133]:

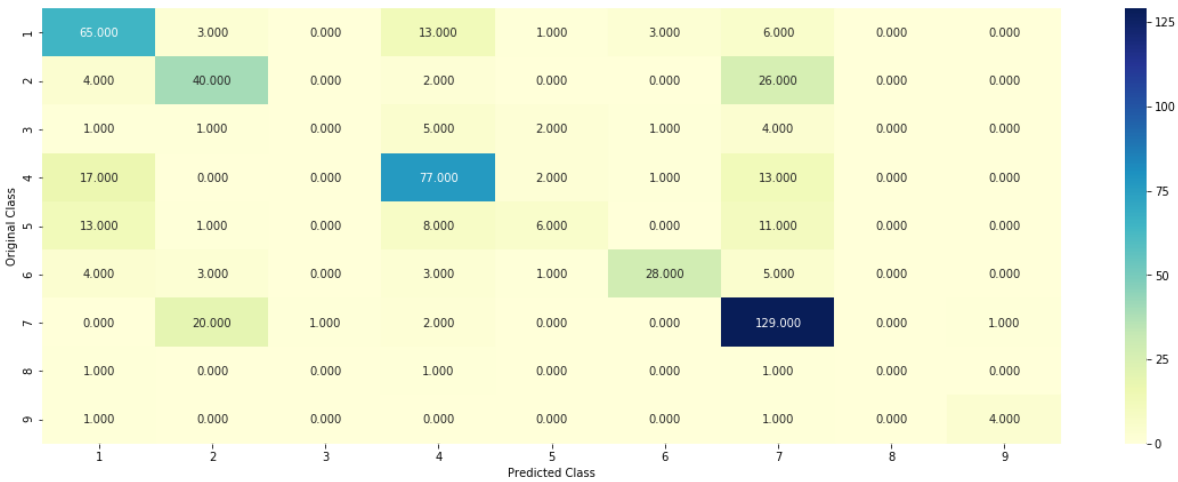
```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_
        clf.fit(x_train_final,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

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_d
clf.fit(x_train_final, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_final, train_y)

predict_y = sig_clf.predict_proba(x_train_final)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:"
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.208188396593792
for n_estimators = 100 and max depth =  10
Log Loss : 1.174784917106713
for n_estimators = 200 and max depth =  5
Log Loss : 1.1973293679896628
for n_estimators = 200 and max depth =  10
Log Loss : 1.1652378363614866
for n_estimators = 500 and max depth =  5
Log Loss : 1.1822785797828403
for n_estimators = 500 and max depth =  10
Log Loss : 1.1558184959356097
for n_estimators = 1000 and max depth =  5
Log Loss : 1.183269623572702
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1556095978751395
for n_estimators = 2000 and max depth =  5
Log Loss : 1.181280376583699
for n_estimators = 2000 and max depth =  10
Log Loss : 1.152925110275245
For values of best estimator =  2000 The train log loss is: 0.60583195277344
25
For values of best estimator =  2000 The cross validation log loss is: 1.152
925110275245
For values of best estimator =  2000 The test log loss is: 1.166387977203047
4
```

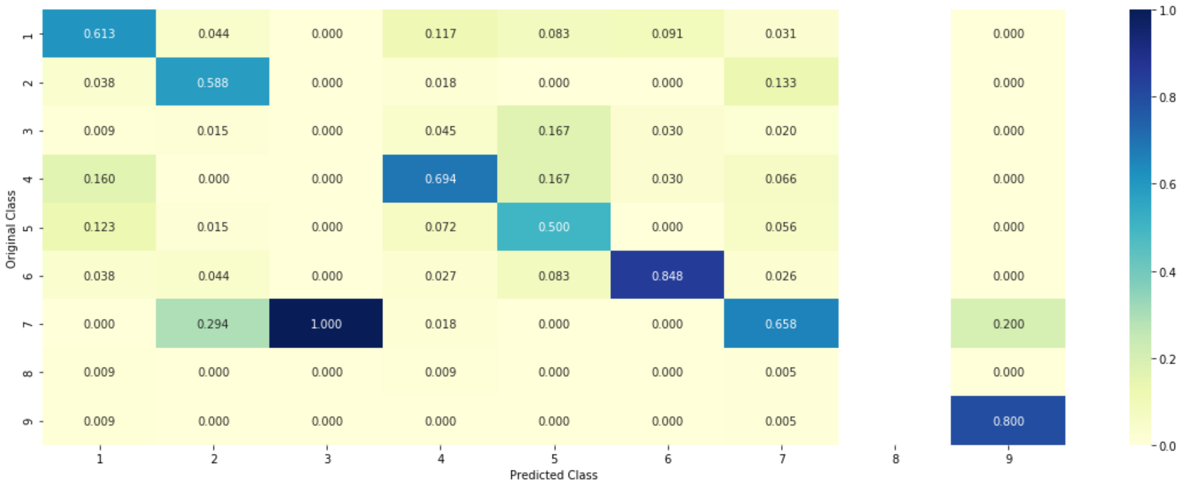
In [134]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c
predict_and_plot_confusion_matrix(x_train_final, train_y,x_cv_final,cv_y, clf)
```

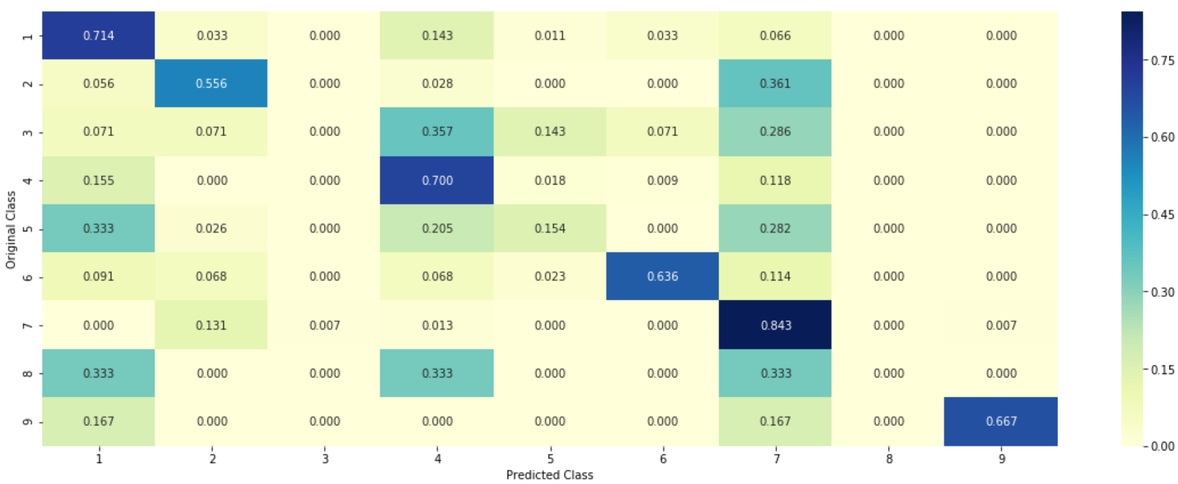
```
Log loss : 1.152925110275245
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```




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




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



Stack the models

In [135]:

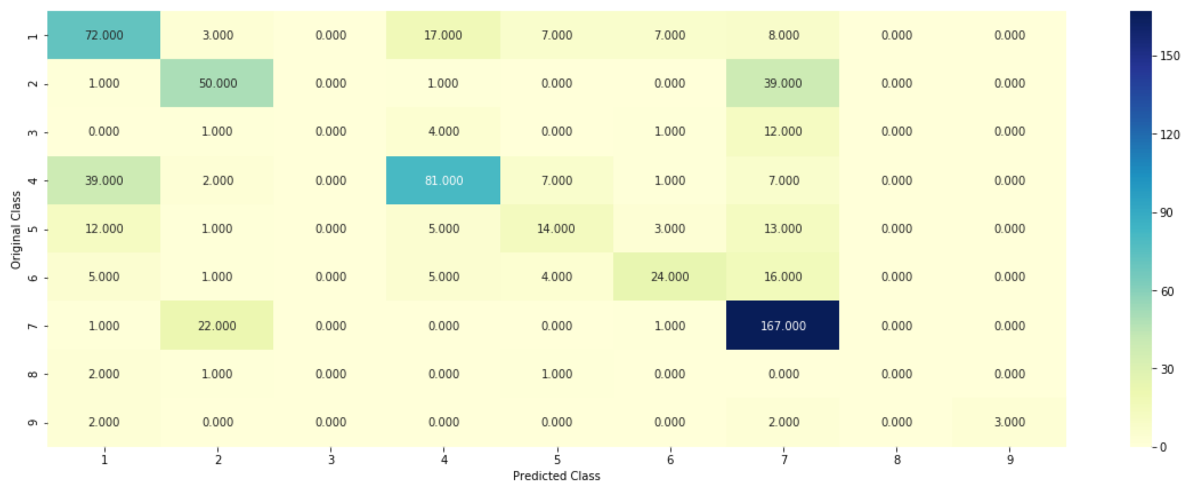
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random
clf1.fit(x_train_final, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

sig_clf1.fit(x_train_final, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(x_c
sig_clf2.fit(x_train_final, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(
sig_clf3.fit(x_train_final, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(x_cv_final))
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=l
    sclf.fit(x_train_final, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(
    log_error =log_loss(cv_y, sclf.predict_proba(x_cv_final))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 0.94
Support vector machines : Log Loss: 1.82
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.029
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.468
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.064
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.226
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.699
```

In [136]:

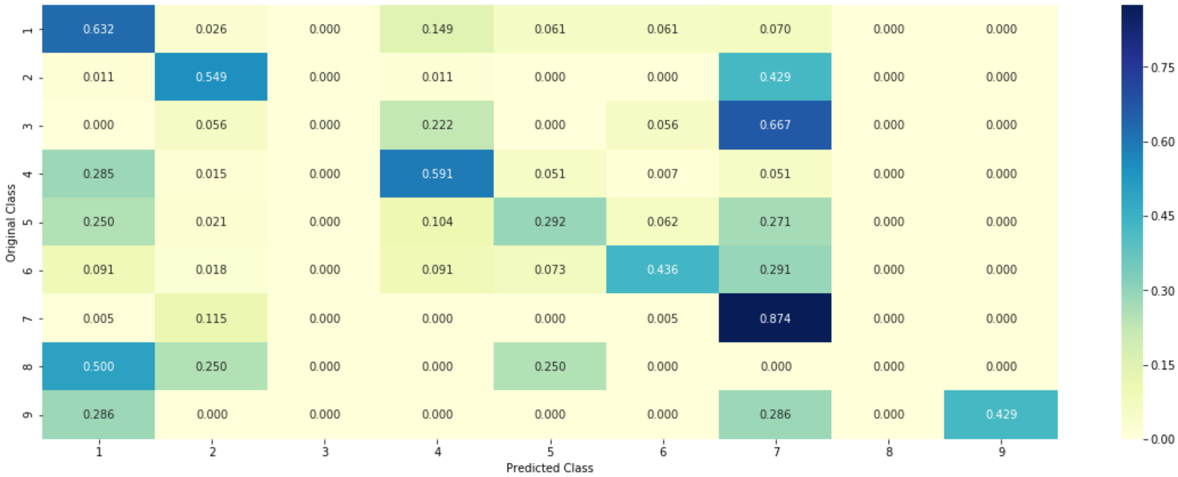
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, u
sclf.fit(x_train_final, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(x_test_final)- tes
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(x_test_final))
```

```
Log loss (train) on the stacking classifier : 0.5579384461991181
Log loss (CV) on the stacking classifier : 1.0643178096270367
Log loss (test) on the stacking classifier : 1.1420987383563035
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```




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


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

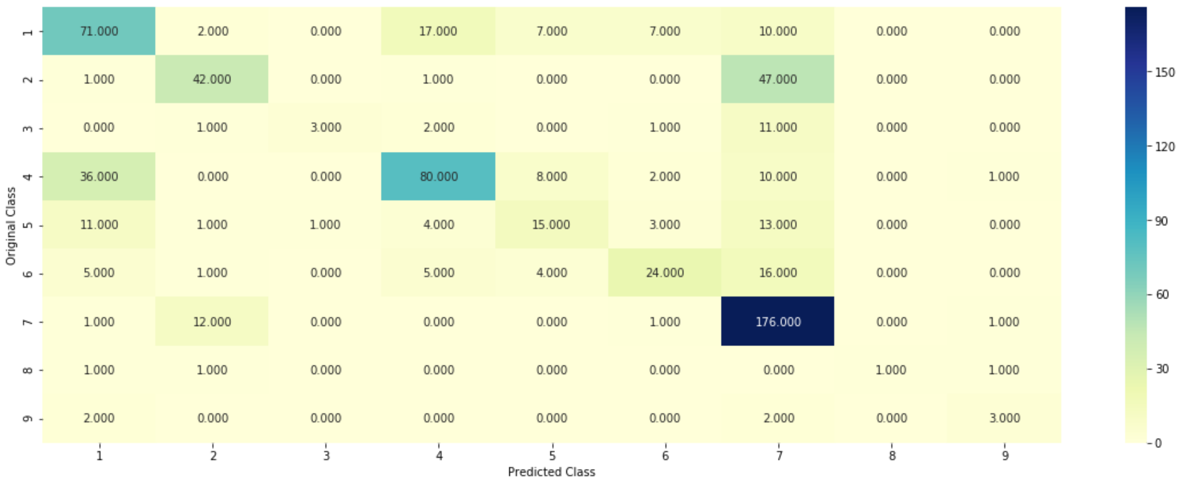


Maximum Voting classifier

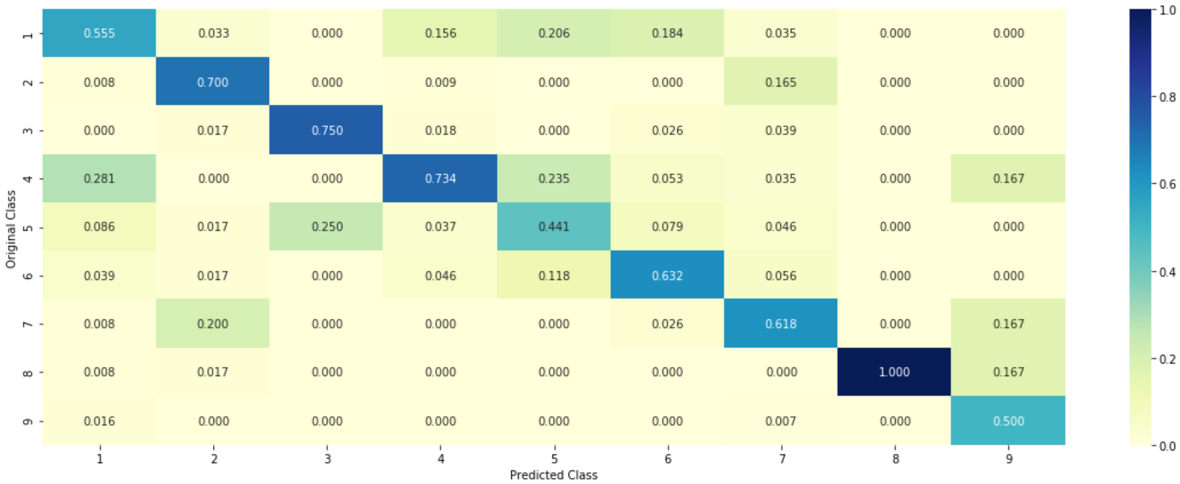
In [137]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.h
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)],
vclf.fit(x_train_final, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(x_
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(x_cv_fir
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(x_te
print("Number of missclassified point :", np.count_nonzero((vclf.predict(x_test_final)- tes
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(x_test_final))
```

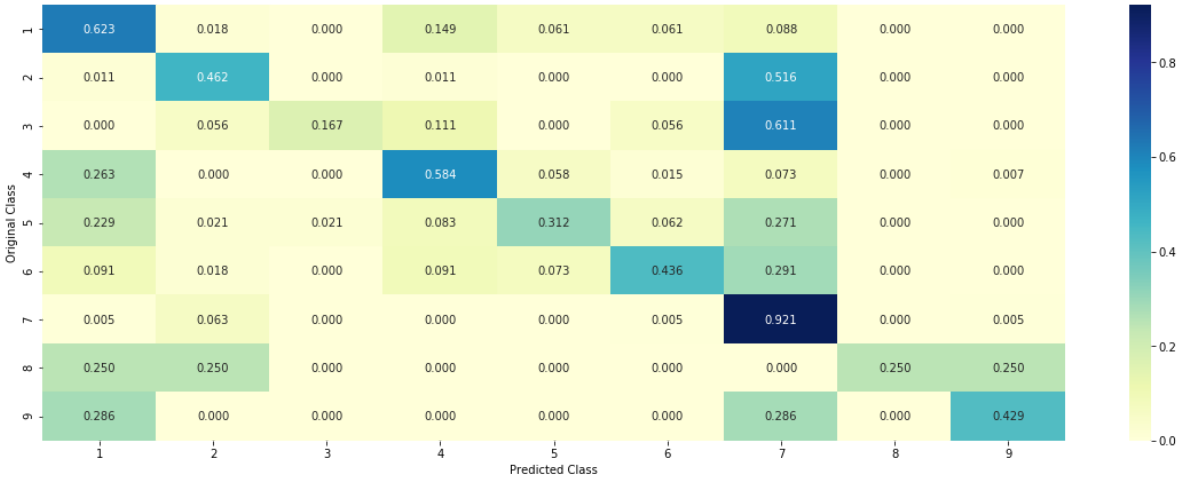
```
Log loss (train) on the VotingClassifier : 0.8452457182089783
Log loss (CV) on the VotingClassifier : 1.1423000038712066
Log loss (test) on the VotingClassifier : 1.195862466038733
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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




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



Summary

In [138]:

```
from prettytable import PrettyTable
t=PrettyTable()
t.field_names = ['Model', 'Train Log Loss', 'CV Log Loss', 'Test Log Loss']
t.add_row(['Naive Bayes', '0.64882', '1.20055', '1.27957'])
t.add_row(['Linear Regression with Class balancing', '0.42051', '0.92603', '0.96959'])
t.add_row(['Linear Regression without Class balancing', '0.40963', '1.02639', '1.04764'])
t.add_row(['Support Vector Machine', '0.47164', '0.94699', '1.00448'])
t.add_row(['Random Forest', '0.60583', '1.15292', '1.16638'])
t.add_row(['Stacking Classifier', '0.55793', '1.06431', '1.14209'])
t.add_row(['Maximum Voting Classifier', '0.84524', '1.14230', '1.19586'])

print(t)
```

```
+-------------------------------------------+----------------+-------------+
---------------+
|                   Model                   | Train Log Loss | CV Log Loss |
Test Log Loss |
+-------------------------------------------+----------------+-------------+
---------------+
|                Naive Bayes                |    0.64882     |   1.20055   |
1.27957    |
|   Linear Regression with Class balancing  |    0.42051     |   0.92603   |
0.96959    |
| Linear Regression without Class balancing |    0.40963     |   1.02639   |
1.04764    |
|           Support Vector Machine          |    0.47164     |   0.94699   |
1.00448    |
|               Random Forest               |    0.60583     |   1.15292   |
1.16638    |
|            Stacking Classifier            |    0.55793     |   1.06431   |
1.14209    |
|         Maximum Voting Classifier         |    0.84524     |   1.14230   |
1.19586    |
+-------------------------------------------+----------------+-------------+
---------------+
```

Conclusion:- Hence we can observe that Logistic Regression with Class balancing is the best model with Test Log Loss of 0.96959(<1).