In [26]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [0]:

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

```
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

In [28]:

```
data = pd.read_csv('drive/My Drive/Colab Notebooks/Cancer_Problem/train
ing_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[28]:

	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

In [29]:

```
# note the seprator in this file
data_text =pd.read_csv("drive/My Drive/Colab Notebooks/Cancer_Problem/t
raining_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1
)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
```

```
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[29]:

	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...

In [30]:

```
# loading stop words from nltk library
import nltk
nltk.download('stopwords')
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 t
he data
            if not word in stop_words:
                string += word + " "
```

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

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

In [31]:

```
# loading stop words from nltk library
import nltk
nltk.download('stopwords')
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 not a stop word then retain that word from t
he data
            if not word in stop_words:
                string += word + " "

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

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

In [32]:

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

Out[32]:

ID	Gene	Variation	Class	TEXT

	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 [0]:

```
#replacing null value present in text with Gene+Variation
result[result.isnull().any(axis=1)]
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result
['Variation']
```

In [0]:

```
#split data into train ,test and cv before you convert it into any vect
orization
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 o
f output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
ratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, str
atify=y_train, test_size=0.2)
```

In [35]:

```
print("train:",train_df.shape)
print("test:",test_df.shape)
print("cvv:",cv_df.shape)
```

```
train: (2124, 5)
test: (665, 5)
cvv: (532, 5)
```

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

#-------------------------------------------
#one-hot encoding of Variation feature
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [0]:
```
print(train_gene_feature_onehotCoding.shape)
print(test_gene_feature_onehotCoding.shape)
print(cv_gene_feature_onehotCoding.shape)
```

```
(2124, 233)
(665, 233)
(532, 233)
```

In [0]:
```
print(train_variation_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
(2124, 1959)
(665, 1959)
(532, 1959)
```

TASK 1 AND 2

In [0]:
```
#first taking top 1k features then applying it two all models
```

```
#hence task 2 i am doing first and after that task 1
#below code for converting to tfidf
tf_idf_vect = TfidfVectorizer(max_features=1000)
tf_idf_vect.fit(train_df.TEXT)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(train_df.TEXT)
X_test_tf_idf = tf_idf_vect.transform(test_df.TEXT)
X_cv_tf_idf = tf_idf_vect.transform(cv_df.TEXT)
#print("the type of count vectorizer ",type(X_train_tf_idf))
#print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_sha
pe())
#print("the number of unique words including both unigrams and bigrams
 ", X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['000', '05', '10', '1
00', '11', '12', '13', '14', '15', '16']
==================================================
```

In [0]:

```
print(X_train_tf_idf.shape)
print(X_test_tf_idf.shape)
print(X_cv_tf_idf.shape)
```

```
(2124, 1000)
(665, 1000)
(532, 1000)
```

In [0]:

```
print("this shapes are for original data set")
print("train",train_df.shape)
print("test",test_df.shape)
print("cv",cv_df.shape)
print("-"*50)

print("this shapes are for GENE features")
print("train",train_gene_feature_onehotCoding.shape)
print("test",test_gene_feature_onehotCoding.shape)
print("cv",cv_gene_feature_onehotCoding.shape)
```

```
print("-"*50)

print("this shapes are for VARIATION features")
print("train",train_variation_feature_onehotCoding.shape)
print("test",test_variation_feature_onehotCoding.shape)
print("cv",cv_variation_feature_onehotCoding.shape)
print("-"*50)

print("this shapes are for TEXT features")
print("train",X_train_tf_idf.shape)
print("test",X_test_tf_idf.shape)
print("cv",X_cv_tf_idf.shape)
```

```
this shapes are for original data set
train (2124, 5)
test (665, 5)
cv (532, 5)
--------------------------------------------------
this shapes are for GENE features
train (2124, 233)
test (665, 233)
cv (532, 233)
--------------------------------------------------
this shapes are for VARIATION features
train (2124, 1959)
test (665, 1959)
cv (532, 1959)
--------------------------------------------------
this shapes are for TEXT features
train (2124, 1000)
test (665, 1000)
cv (532, 1000)
```

In [0]:

```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))
```

```
X_train = hstack((train_gene_var_onehotCoding, X_train_tf_idf)).tocsr()
y_train = np.array(list(train_df['Class']))

X_test = hstack((test_gene_var_onehotCoding, X_test_tf_idf)).tocsr()
y_test = np.array(list(test_df['Class']))

X_cv = hstack((cv_gene_var_onehotCoding, X_cv_tf_idf)).tocsr()
y_cv = np.array(list(cv_df['Class']))
```

In [0]:

```
print("you can see the the shapes after hstack")
print("that is we have same number of data points and just number of fe
atures incresed :so i can say what we did is correct")
print("(number of data points * number of features) in train data = ",
X_train.shape)
print("(number of data points * number of features) in test data = ", X
_test.shape)
print("(number of data points * number of features) in cross validation
 data =", X_cv.shape)
```

```
you can see the the shapes after hstack
that is we have same number of data points and just number of features
incresed :so i can say what we did is correct
(number of data points * number of features) in train data =  (2124, 31
92)
(number of data points * number of features) in test data =  (665, 319
2)
(number of data points * number of features) in cross validation data =
(532, 3192)
```

In [0]:

```
#Now i am ready with all data preparation
#lets apply all the different algorithms
```

MACHINE LEARNING MODELS

In [0]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
```

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of element
s 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 correspo
nds to rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

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

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

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

```
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

In [0]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

```
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [0]:

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

In [0]:

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

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

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

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

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

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

1.Naive Bayes Model

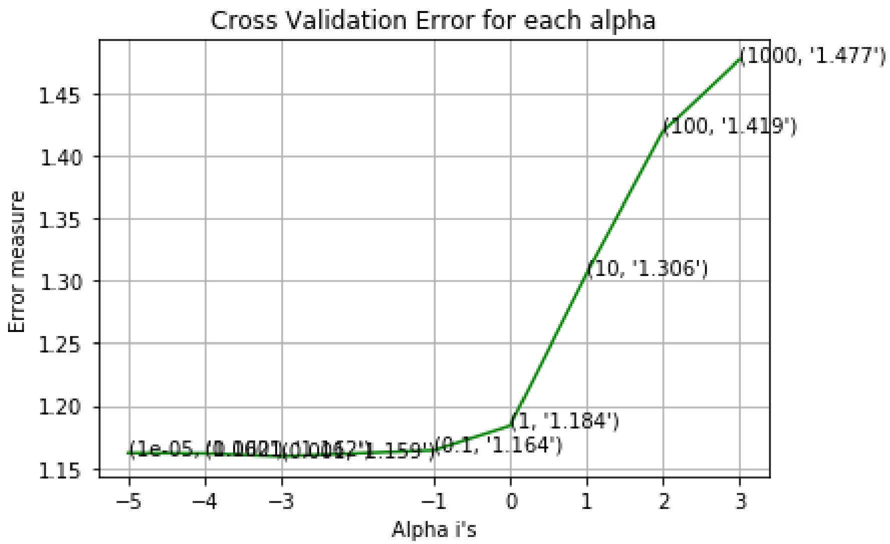
In [0]:

```
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, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(y_cv, 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_a
rray[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, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)


predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.1619491372270754
for alpha = 0.0001
Log Loss : 1.1616556289727744
for alpha = 0.001
Log Loss : 1.1593105839661928
for alpha = 0.1
Log Loss : 1.1641854388827229
for alpha = 1
Log Loss : 1.1838011935360058
for alpha = 10
Log Loss : 1.3055851181673774
for alpha = 100
```

```
Log Loss : 1.419341767406512
for alpha = 1000
Log Loss : 1.4768997553193954
```



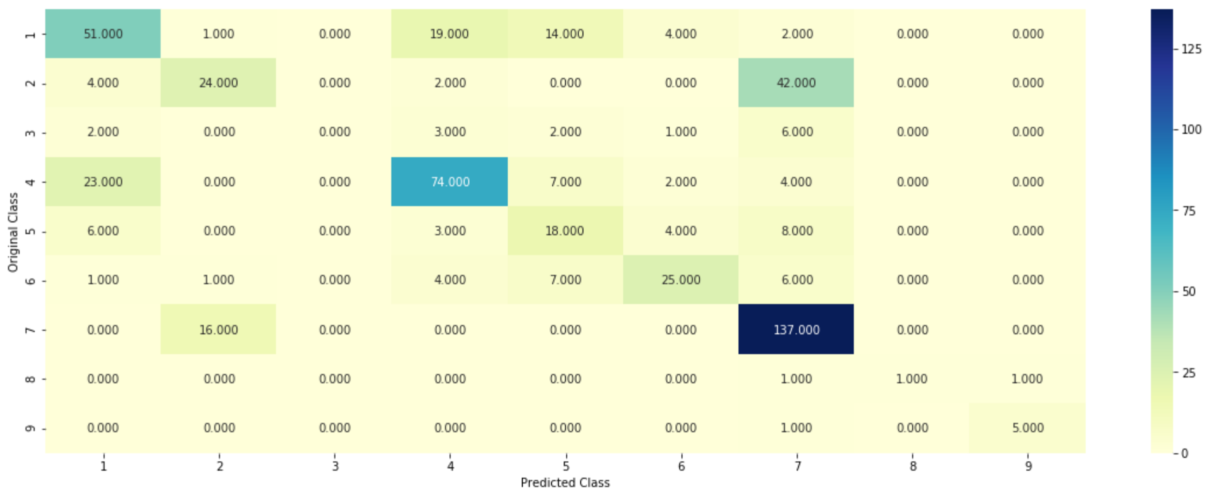

```
For values of best alpha =  0.001 The train log loss is: 0.526374540597
4309
For values of best alpha =  0.001 The cross validation log loss is: 1.1
593105839661928
For values of best alpha =  0.001 The test log loss is: 1.2030073484214
492
```

Testing the model witg best hyperparameter

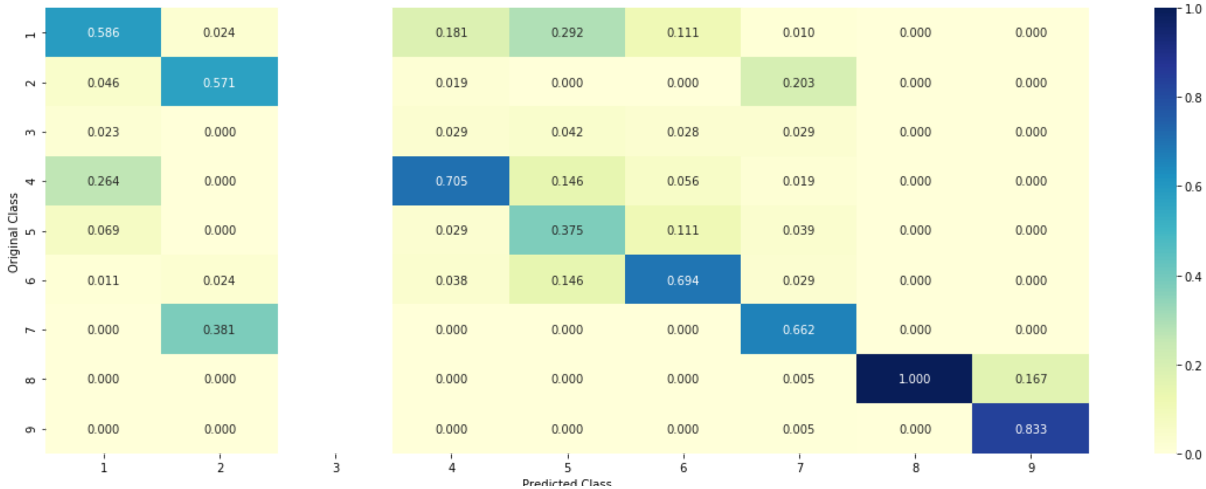
In [0]:
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)
sig_clf_probs = sig_clf.predict_proba(X_cv)
# to avoid rounding error while multiplying probabilites we use log-pro
bability estimates
print("Log Loss :",log_loss(y_cv, sig_clf_probs))
```

```
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(X_cv)- y_cv))/y_cv.shape[0])
plot_confusion_matrix(y_cv, sig_clf.predict(X_cv.toarray()))
```

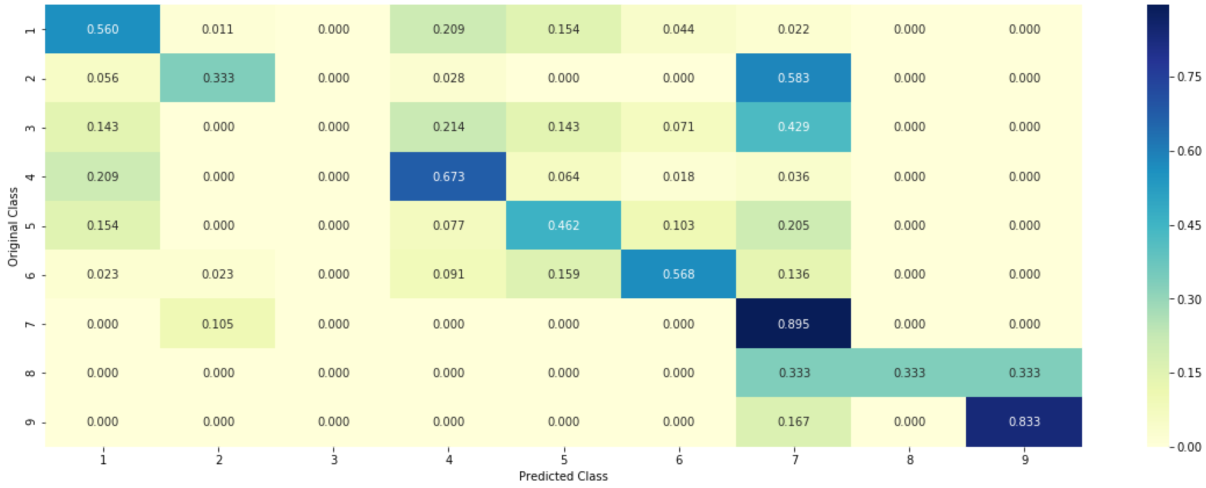
```
Log Loss : 1.1593105839661928
Number of missclassified point : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




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



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



feature importance and correctly classified points

In [0]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
```

```
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0527 0.0443 0.0097 0.0718 0.0347 0.0
315 0.7488 0.003  0.0033]]
Actual Class : 7
--------------------------------------------------
50 Text feature [11] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

feature importance and incorrectly classified points

In [0]:

```
test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1506 0.2247 0.0158 0.3302 0.0522 0.0
397 0.1475 0.0334 0.0059]]
Actual Class : 2
--------------------------------------------------
7 Text feature [1354] present in test data point [True]
47 Text feature [11] present in test data point [True]
58 Text feature [1352] present in test data point [True]
```

```
58 Text feature [1352] present in test data point [True]
88 Text feature [1358] present in test data point [True]
Out of the top  100  features  4 are present in query point
```

K Nearest Neighbour Classification

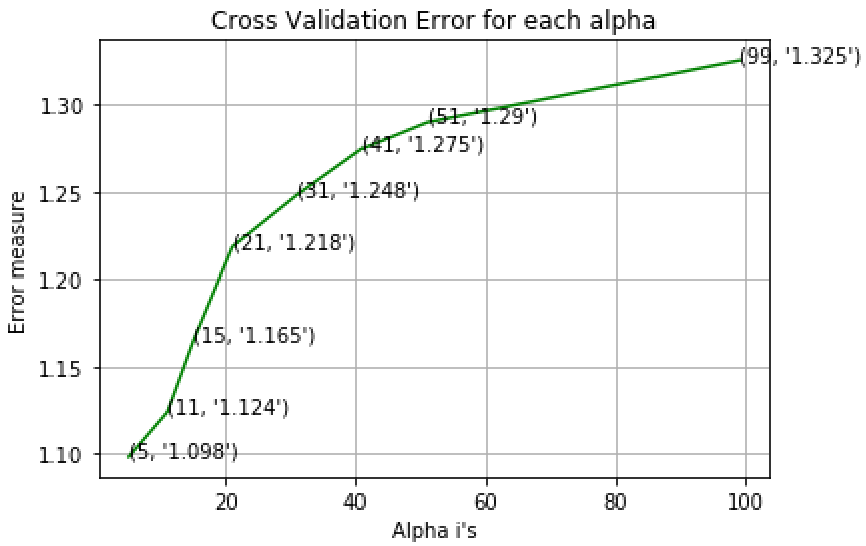
hyper parameter tuning

In [0]:

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
```

```
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0981576771405326
for alpha = 11
Log Loss : 1.1239870473996638
for alpha = 15
Log Loss : 1.164814121116928
for alpha = 21
Log Loss : 1.2182805427686954
for alpha = 31
Log Loss : 1.2481054018231894
for alpha = 41
Log Loss : 1.2746453983938575
for alpha = 51
Log Loss : 1.2896843511105232
for alpha = 99
Log Loss : 1.3251567022923942
```

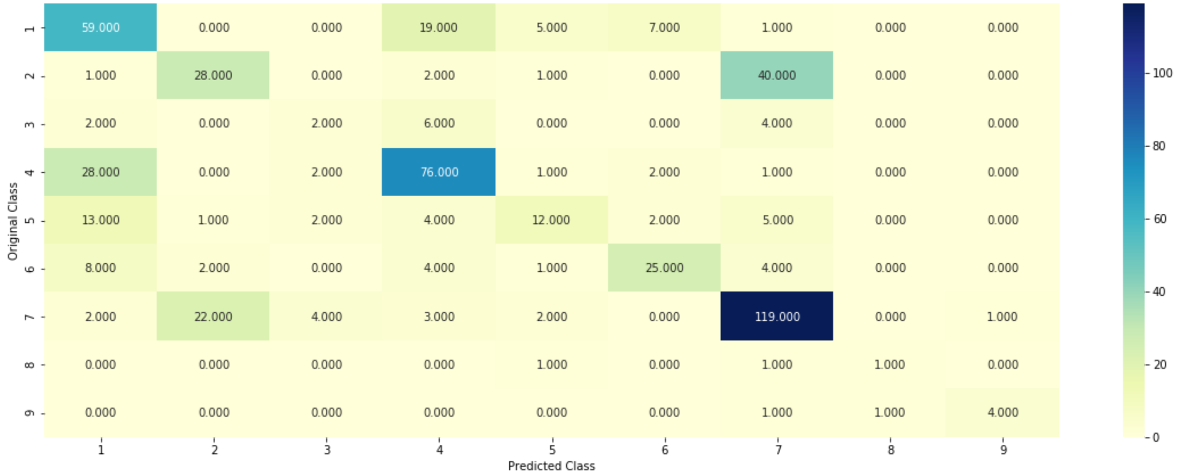



```
For values of best alpha =  5 The train log loss is: 0.8877463288580107
For values of best alpha =  5 The cross validation log loss is: 1.09815
76771405326
For values of best alpha =  5 The test log loss is: 1.1064418792257604
```

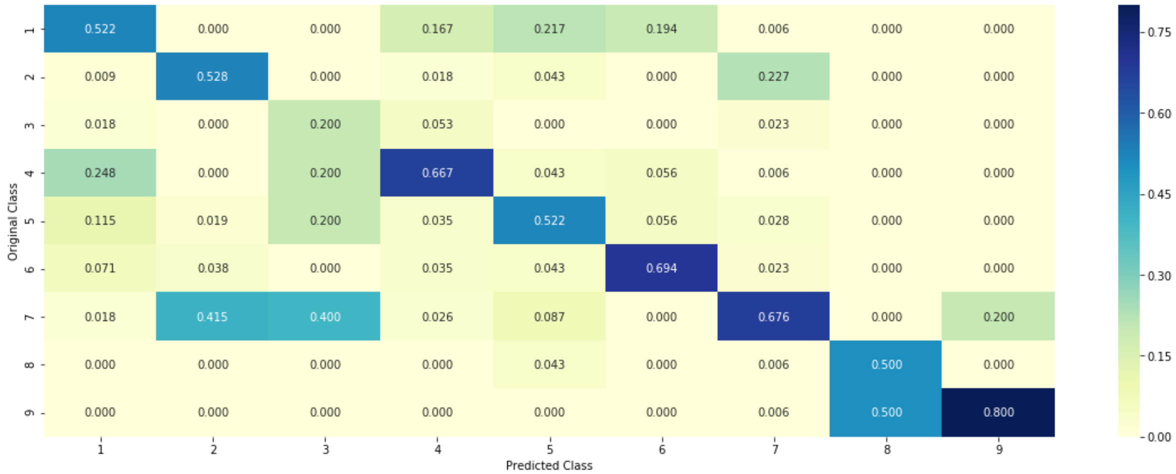
testing the model with best Hyper parameter

In [0]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

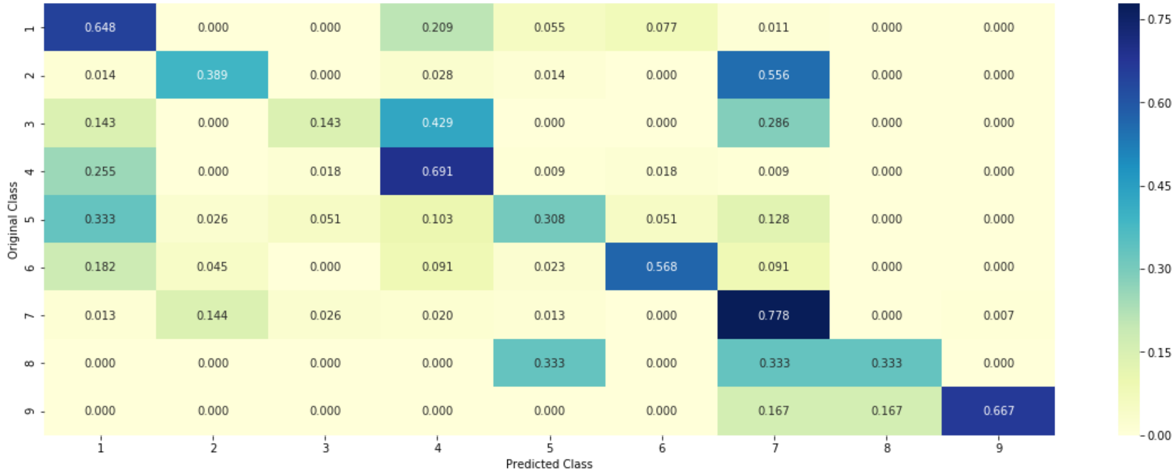
```
Log loss : 1.0981576771405326
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```




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




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



Sample query point 1

In [0]:

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

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

```
Predicted Class : 9
Actual Class : 7
The  5  nearest neighbours of the test points belongs to classes [7 5 7
```

```
7 7]
Fequency of nearest points : Counter({7: 4, 5: 1})
```

Sample query point 2

In [0]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

test_point_index = 100

predicted_cls = sig_clf.predict(X_test[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", y_test[test_point_index])
neighbors = clf.kneighbors(X_test[test_point_index].reshape(1, -1), alp
ha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours of the test points belongs to classes",y_train[neighbors[1][0]])
print("Fequency of nearest points :",Counter(y_train[neighbors[1][0]]))
```

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

Logistic Regression

with class balancing

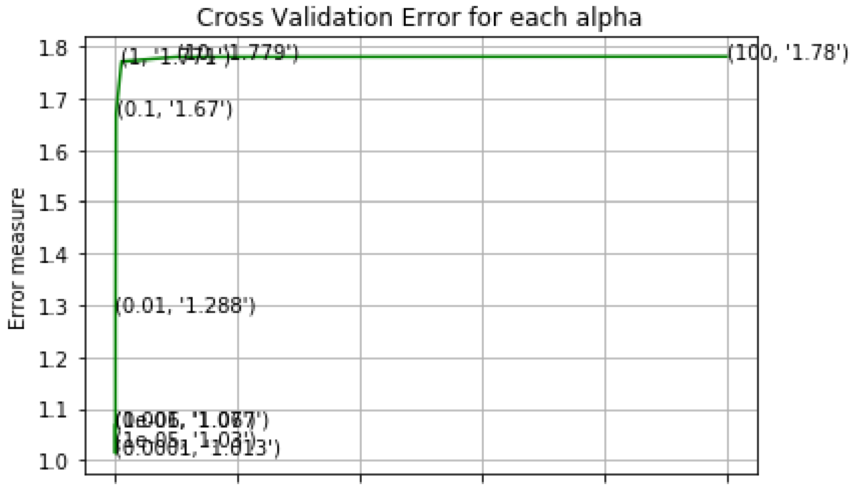
Hyper parameter tuning

In [0]:

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

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

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

```
for alpha = 1e-06
Log Loss : 1.0672994335123587
for alpha = 1e-05
Log Loss : 1.030210850349588
for alpha = 0.0001
Log Loss : 1.0130240298677342
for alpha = 0.001
Log Loss : 1.0699959726640522
for alpha = 0.01
Log Loss : 1.2884456693389315
for alpha = 0.1
Log Loss : 1.6704129905341607
for alpha = 1
Log Loss : 1.7707438335817538
for alpha = 10
Log Loss : 1.779322903914127
for alpha = 100
Log Loss : 1.7801710184789765
```

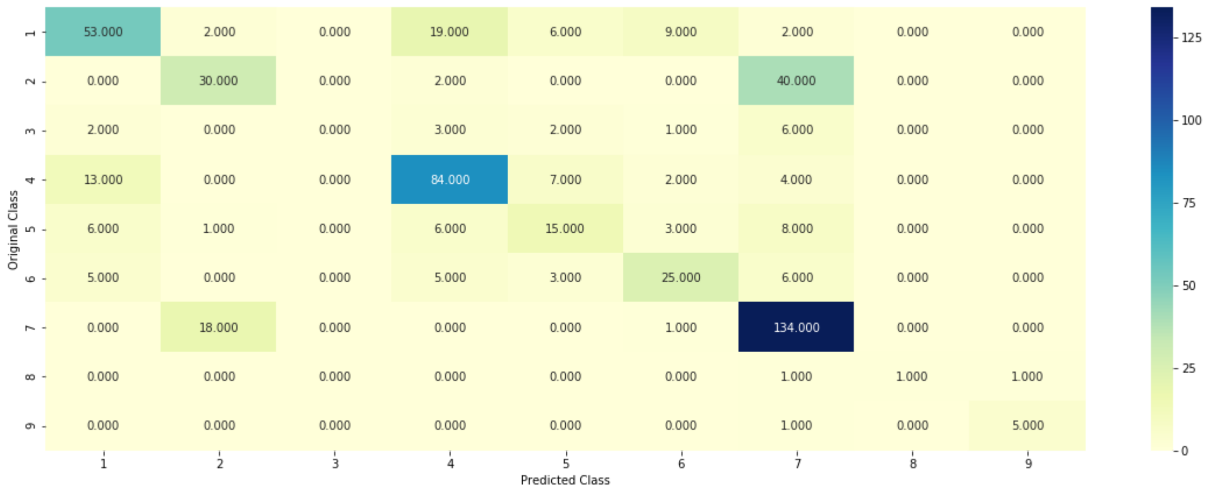



```
For values of best alpha =  0.0001 The train log loss is: 0.47066467549
86259
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0130240298677342
For values of best alpha =  0.0001 The test log loss is: 1.047786509080
4344
```

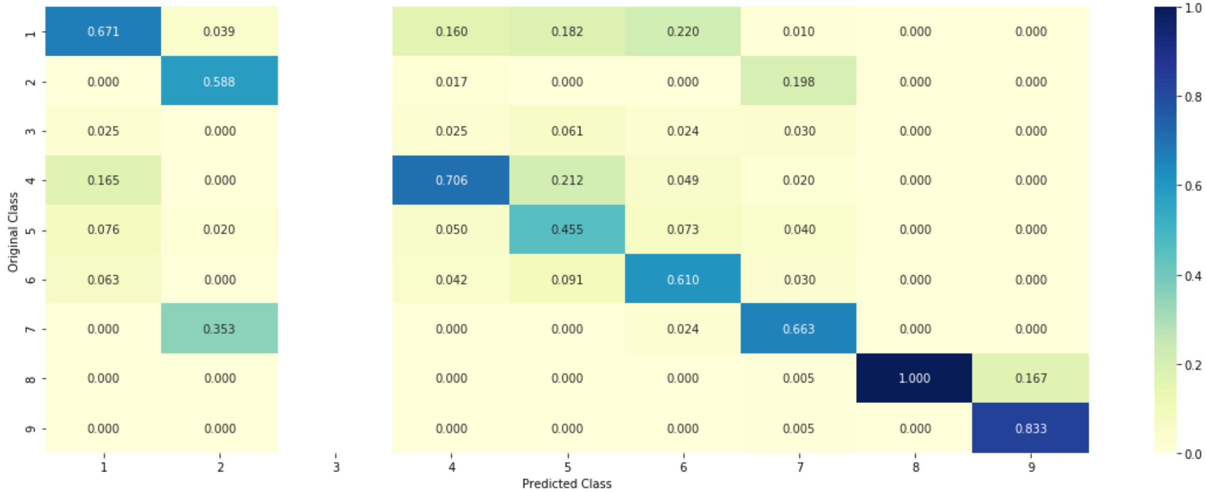
testing with best hyper parameter tuning

In [0]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

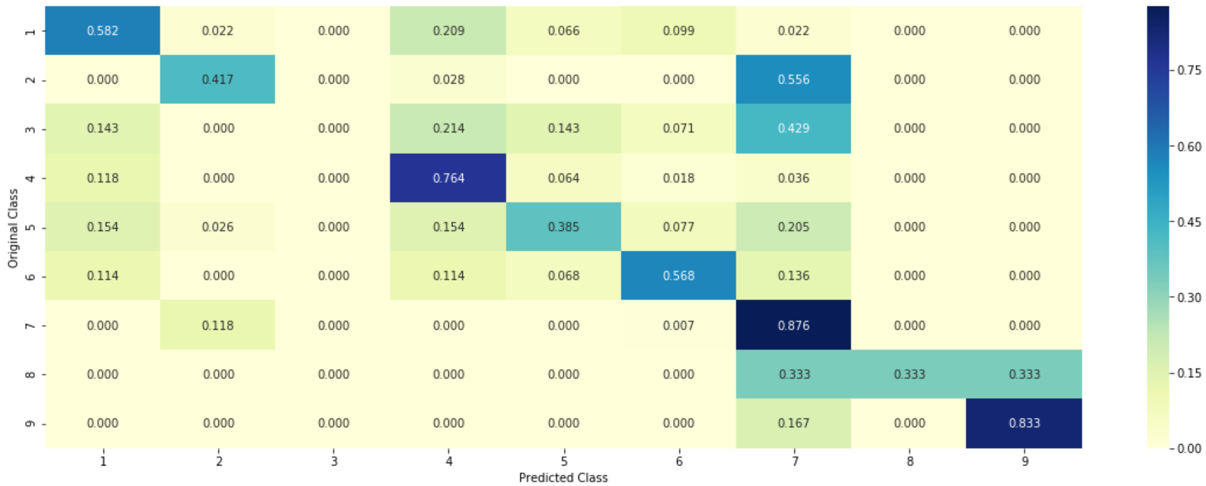
```
Log loss : 1.0130240298677342
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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



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



feature importance

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

Correctly Classified point

In [0]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(X_train,y_train)
test_point_index = 1
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.033  0.0333 0.0254 0.0644 0.0945 0.0
561 0.6832 0.0042 0.0058]]
Actual Class : 7
--------------------------------------------------
106 Text feature [11] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

incorecctly classified point

In [0]:

```
test_point_index = 20
no_feature = 500
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2387 0.0111 0.0074 0.0253 0.25   0.4
517 0.0116 0.0024 0.0018]]
Actual Class : 1
```

```
Actual Class : 1
--------------------------------------------------
282 Text feature [13] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

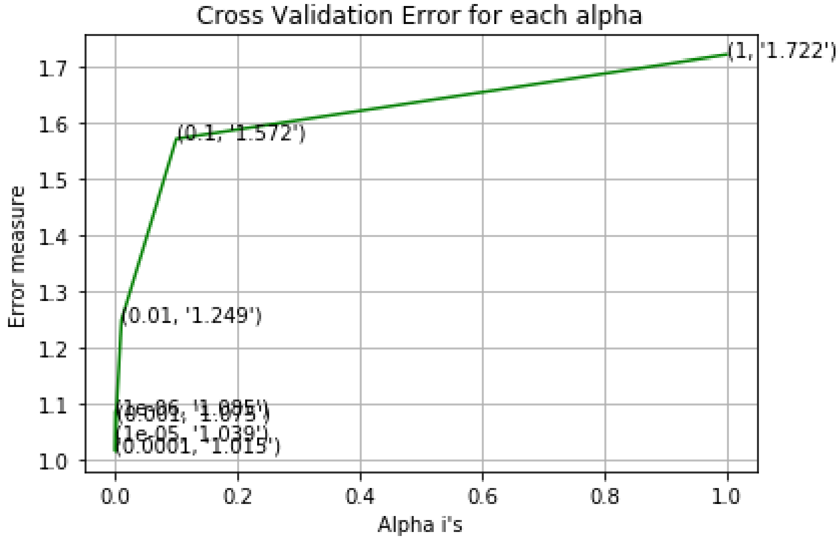
without class balancing

hyper parameter tuning

In [0]:
```
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(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(y_cv, 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(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.0853866589789785
for alpha = 1e-05
Log Loss : 1.0386085830943084
for alpha = 0.0001
Log Loss : 1.0152273435390615
for alpha = 0.001
Log Loss : 1.0750839465279551
for alpha = 0.01
Log Loss : 1.2490884025395046
for alpha = 0.1
Log Loss : 1.5720283141110423
for alpha = 1
Log Loss : 1.7218235299052382
```

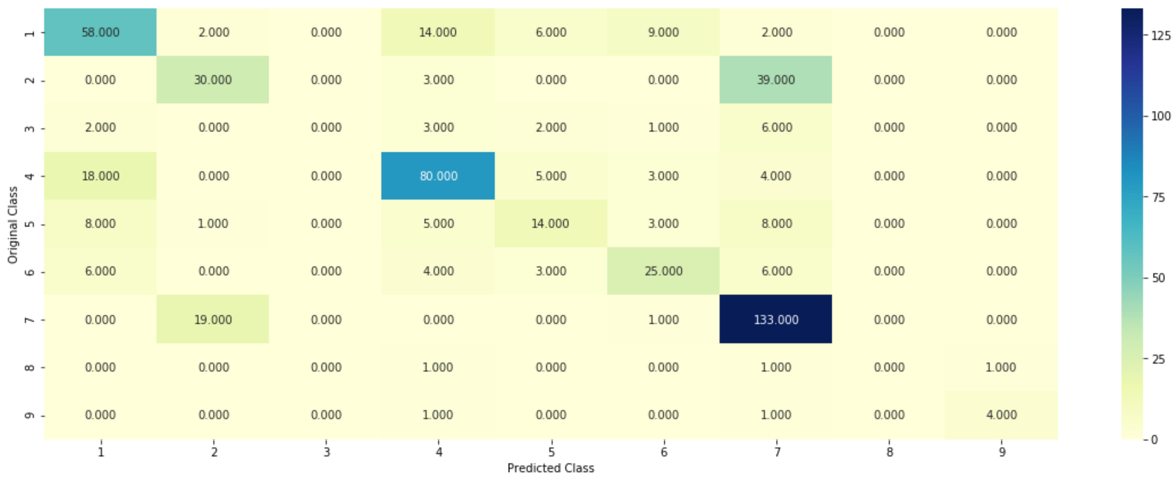



```
For values of best alpha =  0.0001 The train log loss is: 0.457466181
875265
For values of best alpha =  0.0001 The cross validation log loss is:
1.0152273435390615
For values of best alpha =  0.0001 The test log loss is: 1.0461093535
001724
```

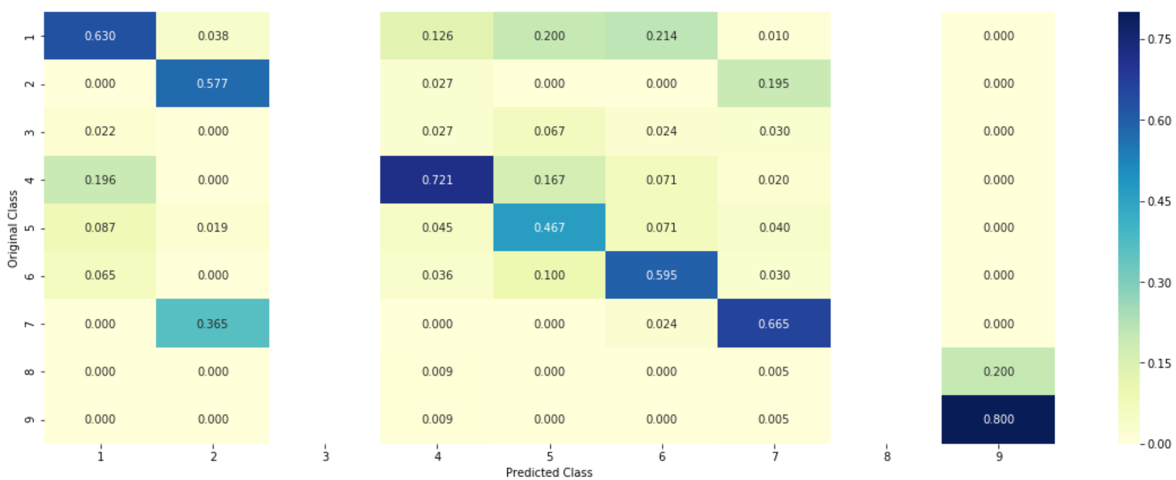
In [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

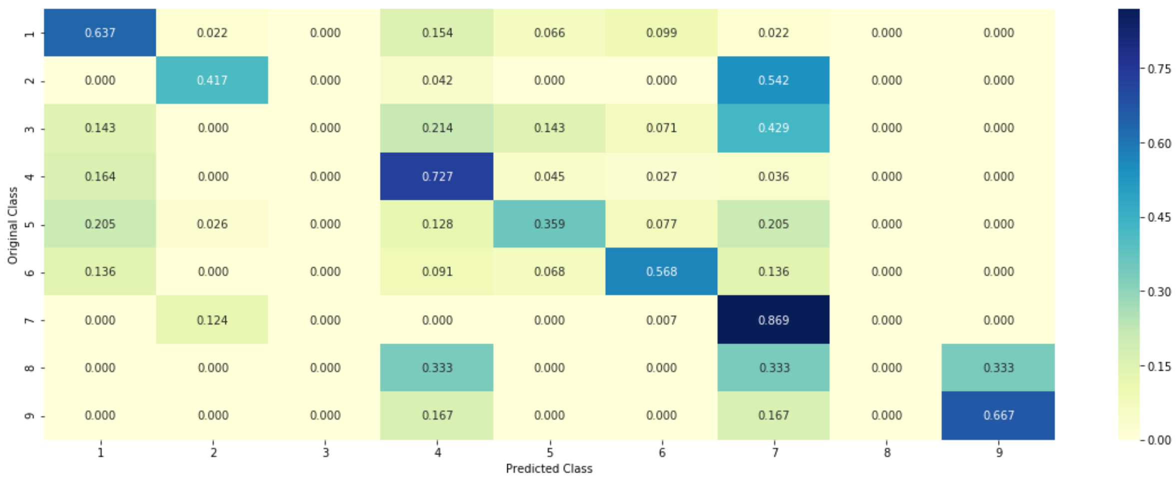
```
Log loss : 1.0152273435390615
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




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




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



Feature Importance, Correctly Classified point

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0299 0.0318 0.0204 0.0591 0.0906 0.0566 0.7014 0.0051 0.0052]]
```

```
Actual Class : 7
--------------------------------------------------
136 Text feature [11] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

Feature Importance, Inorrectly Classified point

In [0]:
```
test_point_index = 20
no_feature = 500
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.231  0.0119 0.0075 0.0237 0.2404 0.4
679 0.0124 0.0028 0.0025]]
Actual Class : 1
--------------------------------------------------
290 Text feature [13] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

Linear Support Vector

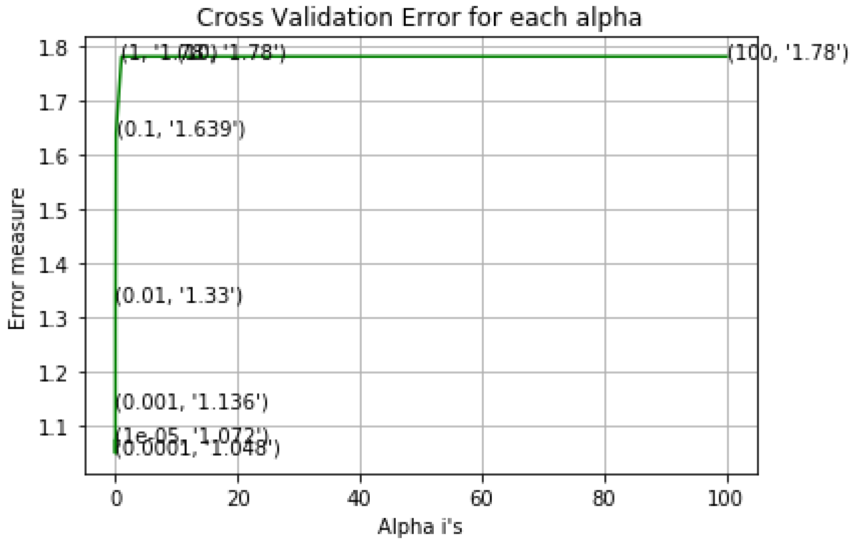
Hyper Parameter tuning

In [0]:
```
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='bal
anced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2'
, loss='hinge', random_state=42)
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(y_cv, 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='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='hinge', random_state=42)
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

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

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

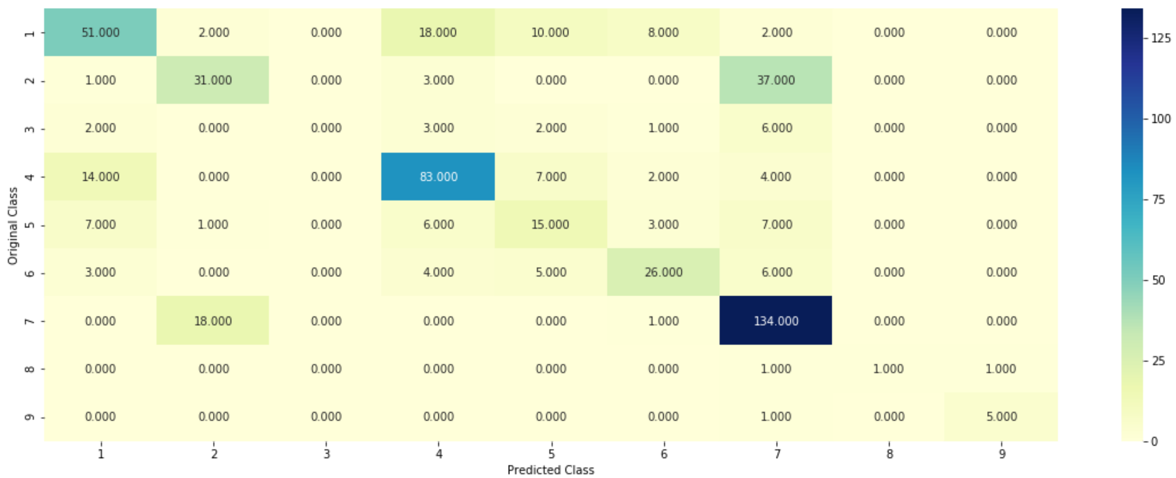
```
for C = 1e-05
Log Loss : 1.071810622580935
for C = 0.0001
Log Loss : 1.0481267510026882
for C = 0.001
Log Loss : 1.13588290461685
for C = 0.01
Log Loss : 1.3299685386088684
for C = 0.1
Log Loss : 1.6393064038911151
for C = 1
Log Loss : 1.7803085187376204
for C = 10
Log Loss : 1.7803088189527154
for C = 100
Log Loss : 1.7803087662378128
```



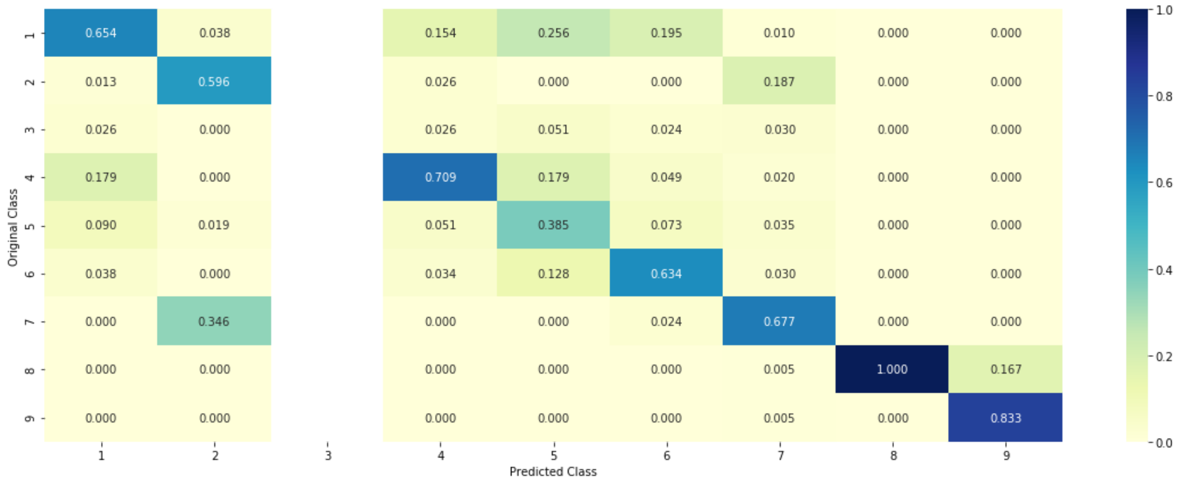
```
For values of best alpha =  0.0001 The train log loss is: 0.40206095672
49059
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0481267510026882

For values of best alpha =  0.0001 The test log loss is: 1.087933637167
0878
```

In [0]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(X_train,y_train,X_cv,y_cv, clf)
```

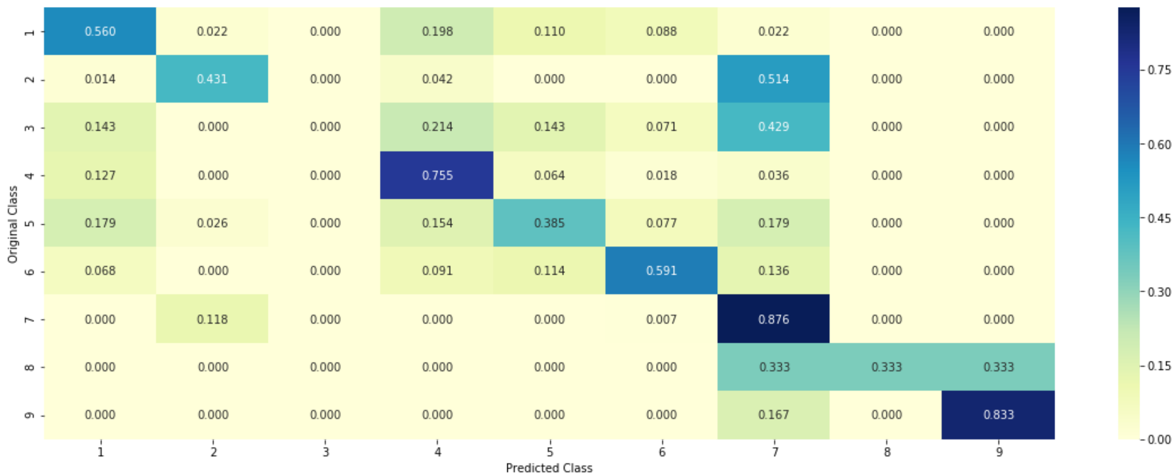
```
Log loss : 1.0481267510026882
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




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




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



Feature importance

for correctly classified point

In [0]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(X_train,y_train)
test_point_index = 2
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1198 0.0804 0.0109 0.6242 0.027  0.0
083 0.1206 0.0037 0.005 ]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

for incorrectly classified point

In [0]:
```
test_point_index = 20
no_feature = 500
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2255 0.0426 0.0125 0.0373 0.248  0.3
616 0.0667 0.0032 0.0028]]
Actual Class : 1
--------------------------------------------------
260 Text feature [13] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

Random Forest Classifier

Hyper parametere tuning

In [0]:

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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
```

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.0910844457443267
for n_estimators = 100 and max depth =  10
Log Loss : 1.0816796184383708
for n_estimators = 200 and max depth =  5
Log Loss : 1.0816144160343475
for n_estimators = 200 and max depth =  10
Log Loss : 1.0800360501857105
for n_estimators = 500 and max depth =  5
Log Loss : 1.076737581111195
for n_estimators = 500 and max depth =  10
Log Loss : 1.0782182529166053
```

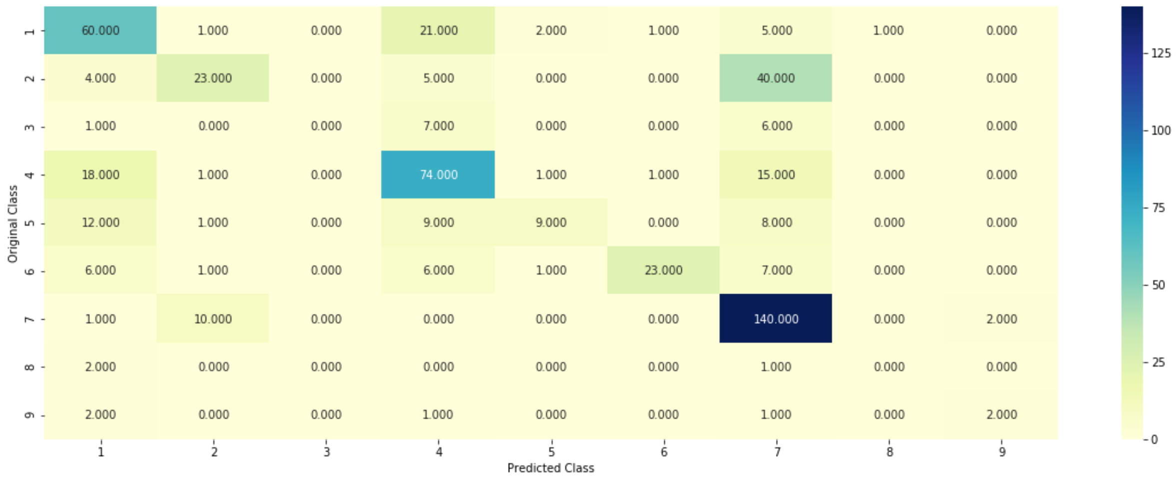
```
for n_estimators = 1000 and max depth =  5
Log Loss : 1.0755992791384972
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0783088807669003

for n_estimators = 2000 and max depth =  5
Log Loss : 1.077581078578767
for n_estimators = 2000 and max depth =  10
Log Loss : 1.077738272135842
For values of best estimator =  1000 The train log loss is: 0.850480844
4067675
For values of best estimator =  1000 The cross validation log loss is:
1.0755992791384972
For values of best estimator =  1000 The test log loss is: 1.0979099148
628966
```

Testing model with best hyper parameter

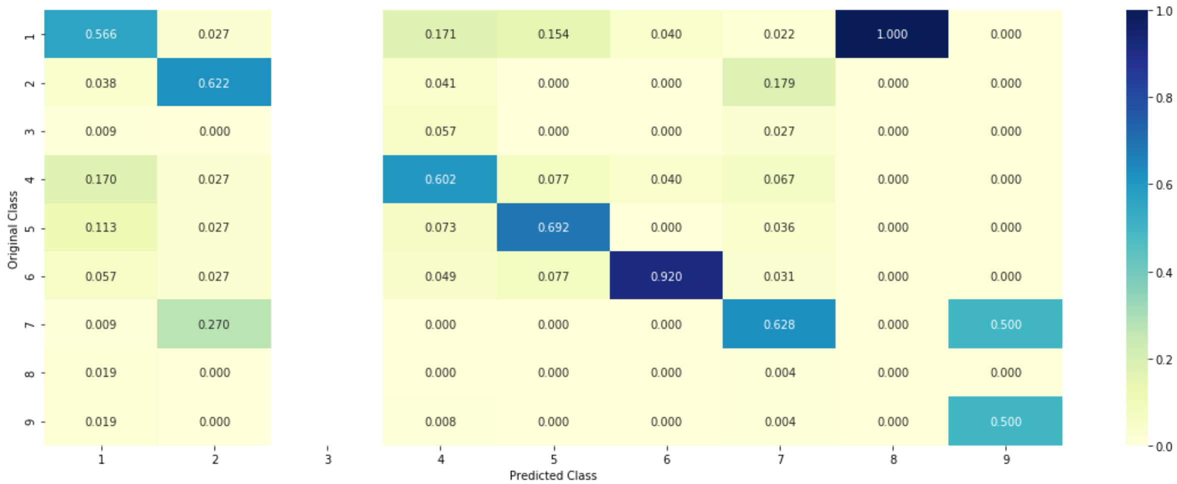
In [0]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
predict_and_plot_confusion_matrix(X_train, y_train,X_cv,y_cv, clf)
```

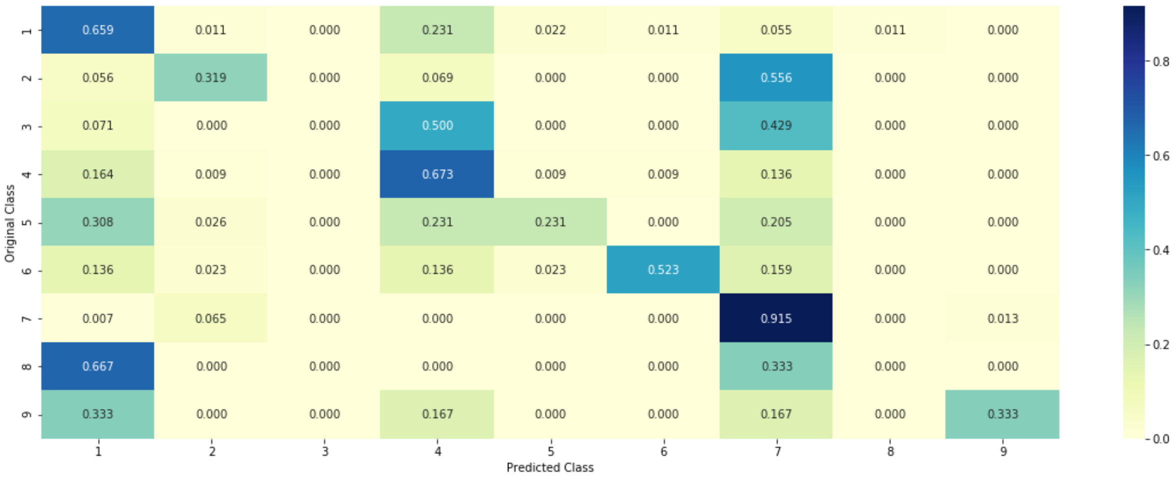
```
Log loss : 1.0755992791384972
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```



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



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



Feature importance

correctly classified point:

In [0]:
```
# test_point_index = 1
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train,y_train)

test_point_index = 100
no_feature = 200
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
```

```
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0807 0.0072 0.0135 0.061  0.7297 0.0
899 0.014  0.0024 0.0016]]
Actual Class : 5
--------------------------------------------------
17 Text feature [109] present in test data point [True]
36 Text feature [13] present in test data point [True]
39 Text feature [1150] present in test data point [True]
102 Text feature [1151] present in test data point [True]
137 Text feature [1147] present in test data point [True]
Out of the top  200  features  5 are present in query point
```

for incorrectly classified points

In [0]:

```
test_point_index = 10
no_feature = 200
predicted_cls = sig_clf.predict(X_test[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
X_test[test_point_index]),4))
print("Actuall Class :", y_test[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 8
Predicted Class Probabilities: [[0.109  0.2749 0.0163 0.0706 0.0416 0.0
292 0.1773 0.2768 0.0043]]
Actuall Class : 2
--------------------------------------------------
36 Text feature [13] present in test data point [True]
```

```
36 Text feature [13] present in test data point [True]
123 Text feature [1350] present in test data point [True]
131 Text feature [107] present in test data point [True]
152 Text feature [1352] present in test data point [True]
Out of the top  200  features  4 are present in query point
```

Stacking models

Hyper parameter tuning

In [0]:

```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(X_train ,y_train)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(X_train ,y_train)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(X_train ,y_train)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(y_cv, sig_cl
f1.predict_proba(X_cv))))
sig_clf2.fit(X_train ,y_train)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(y_cv, sig
_clf2.predict_proba(X_cv))))
sig_clf3.fit(X_train ,y_train)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(y_cv, sig_clf3.predic
t_proba(X_cv))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
```

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

```
Logistic Regression :  Log Loss: 1.07
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.16
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.044
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.542
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.140
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.257
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.610
```

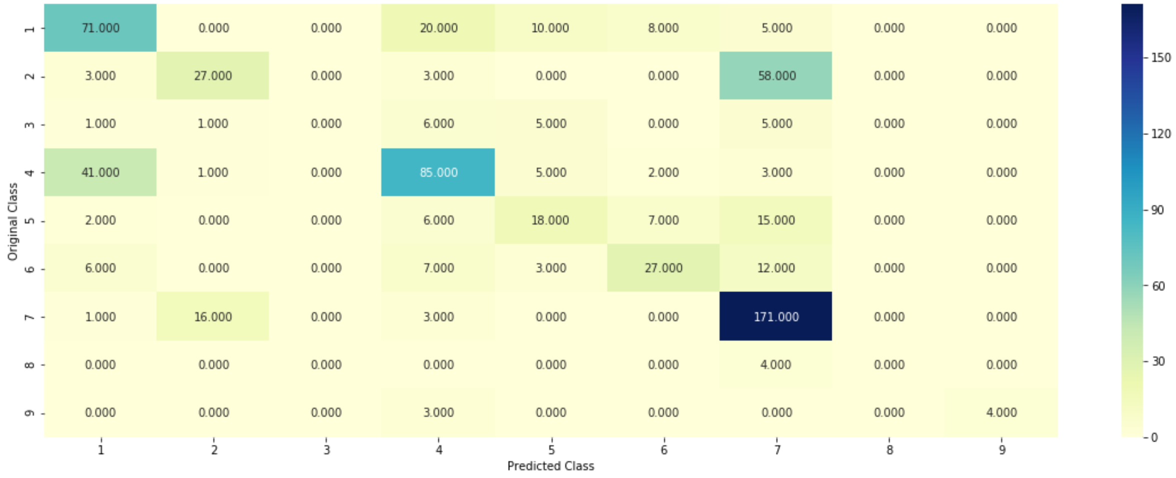
testing the model with best hyper parameter

In [0]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(X_train,y_train)

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

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

log_error = log_loss(y_test, sclf.predict_proba(X_test))
print("Log loss (test) on the stacking classifier :",log_error)
```

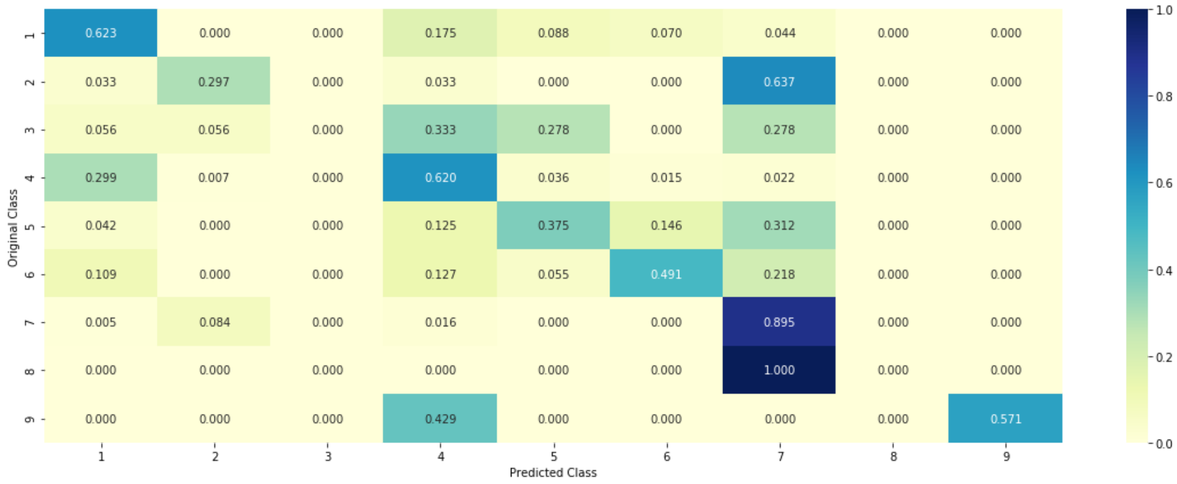
```
print("Number of missclassified point :", np.count_nonzero((sclf.predic
t(X_test)- y_test))/y_test.shape[0])
plot_confusion_matrix(test_y=y_test, predict_y=sclf.predict(X_test))
```

```
Log loss (train) on the stacking classifier : 0.5464534152548693
Log loss (CV) on the stacking classifier : 1.1398838453779343
Log loss (test) on the stacking classifier : 1.1864890805111423
Number of missclassified point : 0.39398496240601505
-------------------- Confusion matrix --------------------
```



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

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

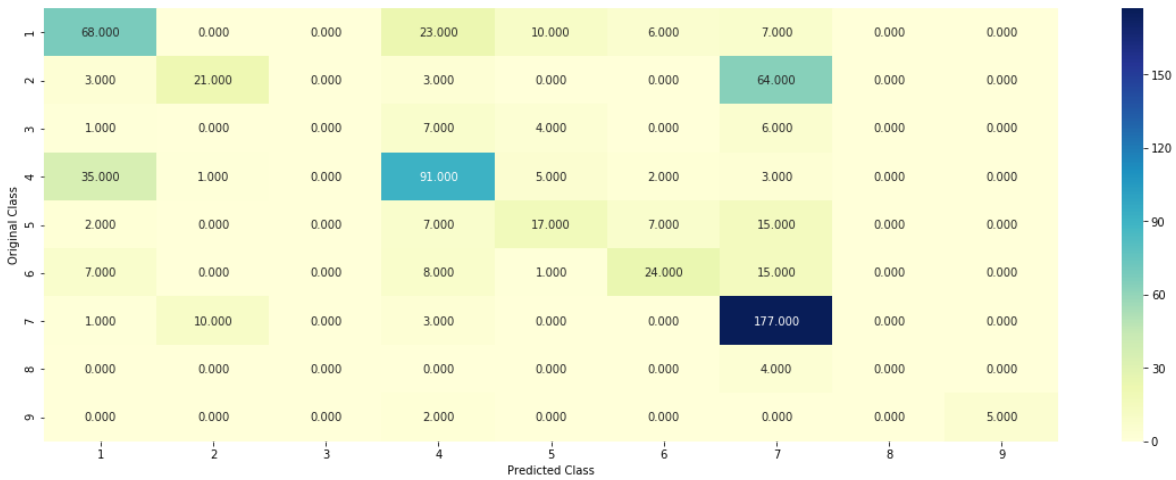


maximum voting classifier

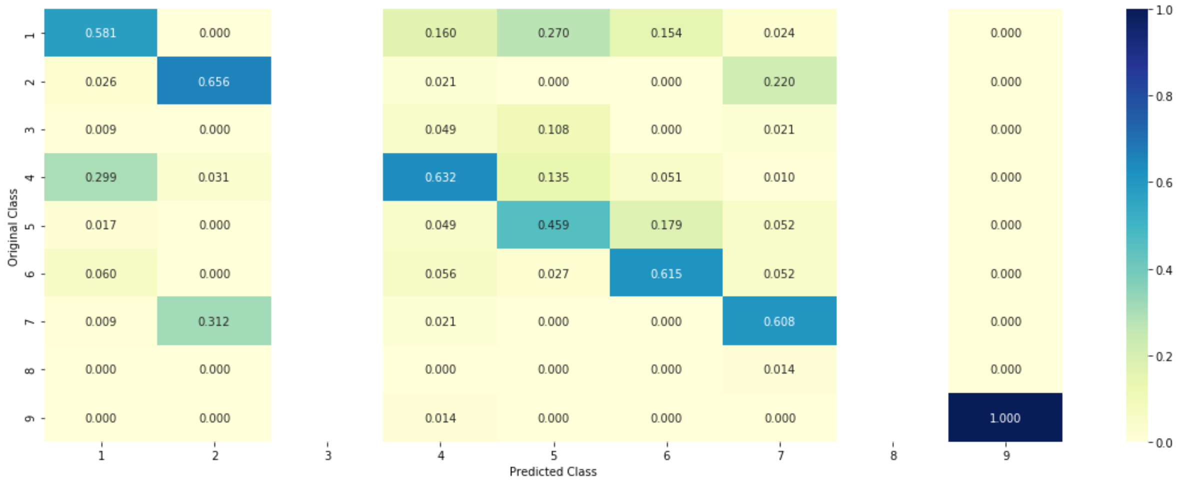
In [0]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensembl
e.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
```

```
), ('rf', sig_clf3)], voting='soft')
vclf.fit(X_train,y_train)
print("Log loss (train) on the VotingClassifier :", log_loss(y_train, v
clf.predict_proba(X_train)))
print("Log loss (CV) on the VotingClassifier :", log_loss(y_cv, vclf.pr
edict_proba(X_cv)))
print("Log loss (test) on the VotingClassifier :", log_loss(y_test, vcl
f.predict_proba(X_test)))
print("Number of missclassified point :", np.count_nonzero((vclf.predic
t(X_test)- y_test))/y_test.shape[0])
plot_confusion_matrix(test_y=y_test, predict_y=vclf.predict(X_test))
```

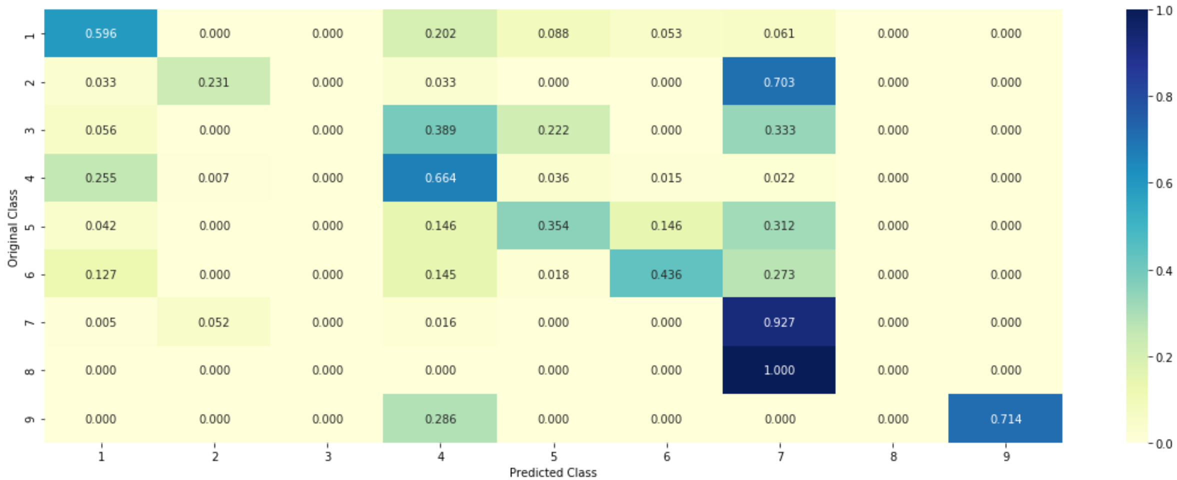
```
Log loss (train) on the VotingClassifier : 0.8659969824969254
Log loss (CV) on the VotingClassifier : 1.2306534898244248
Log loss (test) on the VotingClassifier : 1.256289931650356
Number of missclassified point : 0.39398496240601505
-------------------- Confusion matrix --------------------
```




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




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



In [0]:

```
from prettytable import PrettyTable
print('RESULTS OF ALL ALGORITHMS USING TFIDF VECTORIZER')
x = PrettyTable()
x.field_names = ["Algorithm", "Log Loss of Train", 'Log Loss of CV',"Lo
g Loss of Test","% of miss classification"]
x.add_row(["Naive bayes", 0.52,1.15,1.20,'37.03%'])
```

```
x.add_row(["KNN",0.88,1.09, 1.10,'38.72%'])
x.add_row(["LR with weight Balancing",0.47,1.01,1.04,'34.77%'])
x.add_row(["LR without weight balancing",0.45,1.01, 1.04,'35.33%'])
x.add_row(["Linear SVM",0.40,1.04, 1.08,'34.96%'])
x.add_row(["RF",0.85,1.07, 1.09,'37.77%'])
x.add_row(["stacking classifier",0.54,1.13, 1.18,'39.39%'])
x.add_row(["Maximum voting classifier",0.86,1.23, 1.25,'39.39%'])
print(x)
```

```
RESULTS OF ALL ALGORITHMS USING TFIDF VECTORIZER
+-----------------------------+-------------------+-----------------+---
---------------+-------------------------+
|          Algorithm          | Log Loss of Train | Log Loss of CV | Lo
g Loss of Test | % of miss classification |
+-----------------------------+-------------------+-----------------+---
---------------+-------------------------+
|         Naive bayes         |        0.52       |       1.15      |
    1.2        |          37.03%         |
|             KNN             |        0.88       |       1.09      |
    1.1        |          38.72%         |
|   LR with weight Balancing  |        0.47       |       1.01      |
    1.04       |          34.77%         |
| LR without weight balancing |        0.45       |       1.01      |
    1.04       |          35.33%         |
|          Linear SVM         |        0.4        |       1.04      |
    1.08       |          34.96%         |
|              RF             |        0.85       |       1.07      |
    1.09       |          37.77%         |
|     stacking classifier     |        0.54       |       1.13      |
    1.18       |          39.39%         |
|  Maximum voting classifier  |        0.86       |       1.23      |
    1.25       |          39.39%         |
+-----------------------------+-------------------+-----------------+---
---------------+-------------------------+
```

In [0]:

```
#based on the above results the test log loss for logistic regression i
s low that is 1.01 and % of misclassification also less that is
#34.77% so our best model till now is Lr with weight balancing
```

TASK 3

In [0]:
```
print("train:",train_df.shape)
print("test:",test_df.shape)
print("cvv:",cv_df.shape)
```

```
train: (2124, 5)
test: (665, 5)
cvv: (532, 5)
```

applying unigram

In [0]:
```
#applying countvectorizer with unigram to gene data
#by defualt it is unigram so no need to pass any parameter
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

#-------------------------------------------
#applying countvectorizer with unigram to variation data
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])

#applying countvectorizer with unigram to text data
text_vectorizer = CountVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEX
```

```
T'])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
```

In [0]:
```
#hstacking all unigram features
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

X_train_uni = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
y_train = np.array(list(train_df['Class']))

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

X_cv_uni = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
y_cv = np.array(list(cv_df['Class']))
```

In [0]:
```
print("shapes of three train features after applyihng unigrams on them:")
print(train_gene_feature_onehotCoding.shape)
print(train_variation_feature_onehotCoding.shape)
print(train_text_feature_onehotCoding.shape)
```

```
shapes of three train features after applyihng unigrams on them:
(2124, 233)
(2124, 1959)
(2124, 130909)
```

In [0]:
```
print("confirmation that all features joined or not")
print(X_train_uni.shape)
```

```
confirmation that all features joined or not
```

```
(2124, 133101)
```

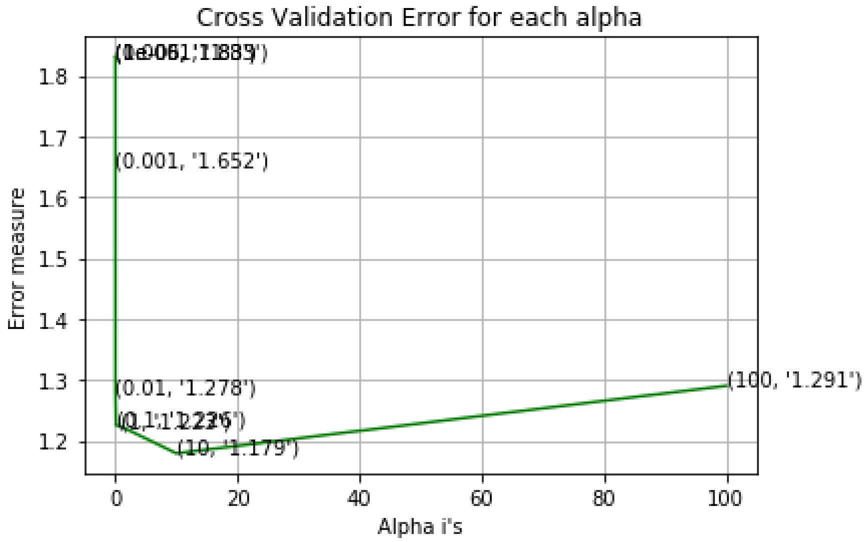
Logistic regression with weight balancing and on unigram data

hyper parameter tuning

In [0]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(X_train_uni,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_uni,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv_uni)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(y_cv, 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], p
```

```
enalty='l2', loss='log', random_state=42)
clf.fit(X_train_uni,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_uni,y_train)

predict_y = sig_clf.predict_proba(X_train_uni)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(X_cv_uni)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(X_test_uni)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.8304997567764278
for alpha = 1e-05
Log Loss : 1.8304997567764278
for alpha = 0.0001
Log Loss : 1.8304997567764278
for alpha = 0.001
Log Loss : 1.6519998099980078
for alpha = 0.01
Log Loss : 1.2780637468358704
for alpha = 0.1
Log Loss : 1.2255930227994671
for alpha = 1
Log Loss : 1.2232367519812335
for alpha = 10
Log Loss : 1.1792382959054575
for alpha = 100
Log Loss : 1.2905295395893512
```



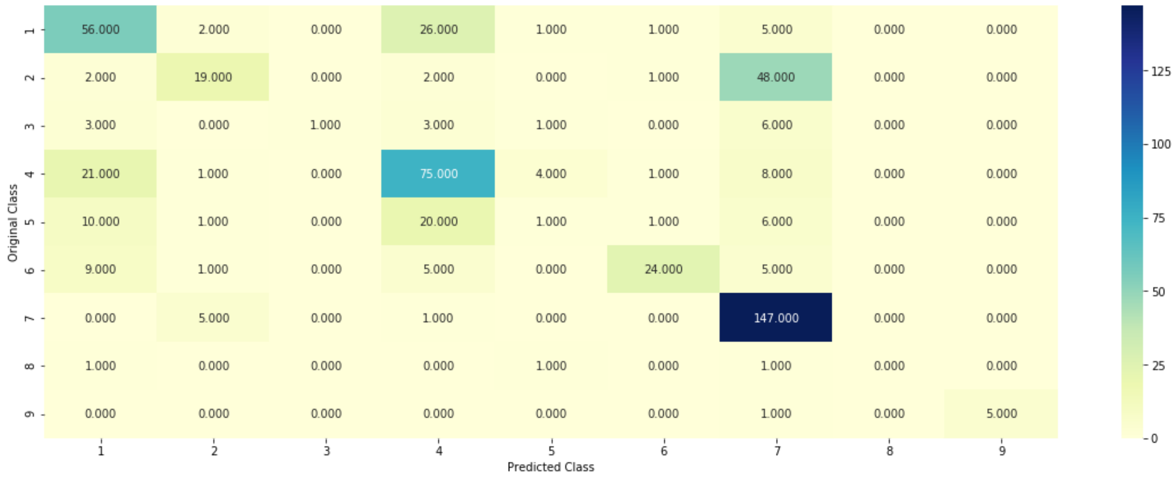

```
For values of best alpha =  10 The train log loss is: 0.936616970884157
6
For values of best alpha =  10 The cross validation log loss is: 1.1792
382959054575
For values of best alpha =  10 The test log loss is: 1.2432810700250843
```

Testing the model with best hyper parameter

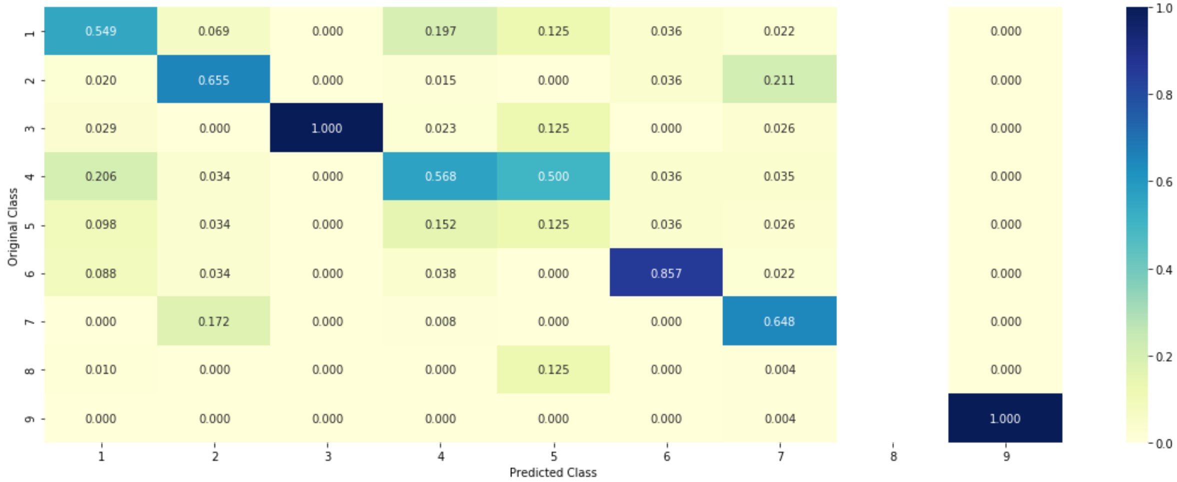
In [0]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(X_train_uni,y_train, X_cv_uni,y_cv, c
lf)
```

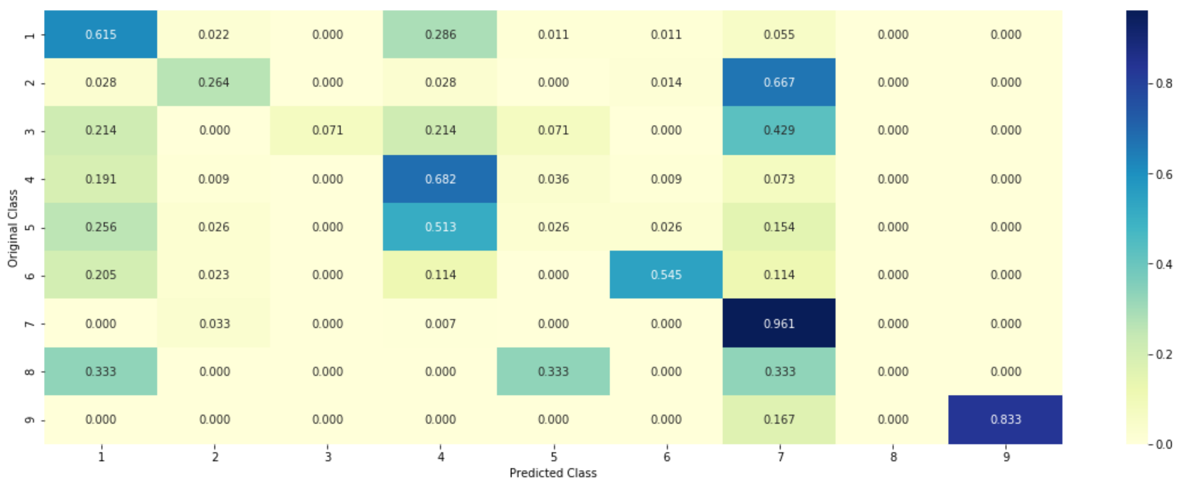
```
Log loss : 1.1792382959054575
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```



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



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



applying bigram

In [0]:
```
#applying countvectorizer with bigram to text data
text_vectorizer_bi = CountVectorizer(ngram_range=(2, 2))
train_text_feature_onehotCoding_bi = text_vectorizer_bi.fit_transform(t
rain_df['TEXT'])
test_text_feature_onehotCoding_bi = text_vectorizer_bi.transform(test_d
f['TEXT'])
cv_text_feature_onehotCoding_bi = text_vectorizer_bi.transform(cv_df['T
EXT'])
```

In [0]:
```
#hstacking all bigram features
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))
```

```
X_train_bi = hstack((train_gene_var_onehotCoding, train_text_feature_on
ehotCoding_bi)).tocsr()
y_train = np.array(list(train_df['Class']))

X_test_bi = hstack((test_gene_var_onehotCoding,test_text_feature_onehot
Coding_bi)).tocsr()
y_test = np.array(list(test_df['Class']))

X_cv_bi = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCodin
g_bi)).tocsr()
y_cv = np.array(list(cv_df['Class']))
```

In [0]:
```
print(train_gene_feature_onehotCoding_bi.shape)
print(train_variation_feature_onehotCoding_bi.shape)
print(train_text_feature_onehotCoding_bi.shape)
```

```
(2124, 227)
(2124, 2043)
(2124, 1803605)
```

In [0]:
```
print(X_train_bi.shape)
print(X_test_bi.shape)
print(X_cv_bi.shape)
```

```
(2124, 1805797)
(665, 1805797)
(532, 1805797)
```

Logistic regression with weight balancing and on bigram data data

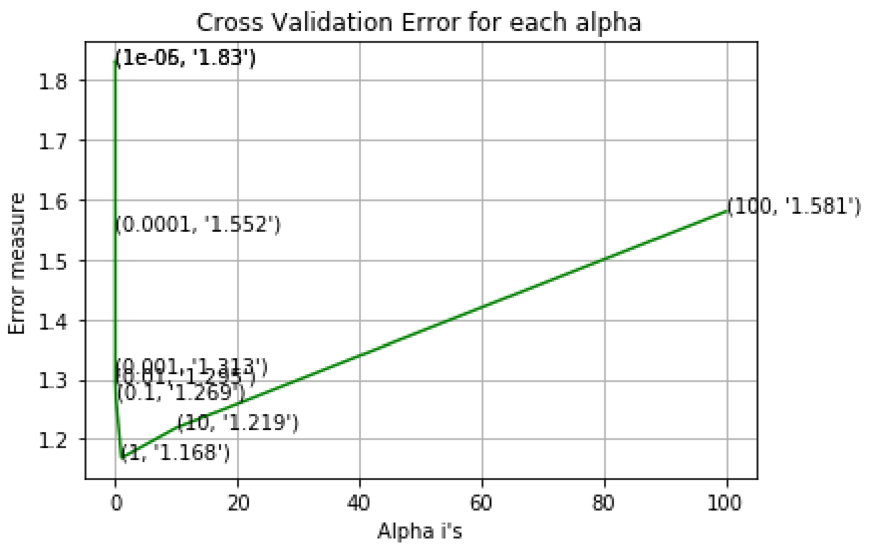
hyper parameter tuning

In [0]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
```

```
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(X_train_bi,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_bi,y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv_bi)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(y_cv, 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], p
enalty='l2', loss='log', random_state=42)
clf.fit(X_train_bi,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_bi,y_train)

predict_y = sig_clf.predict_proba(X_train_bi)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(X_cv_bi)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
```

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

```
for alpha = 1e-06
Log Loss : 1.8304997567764278
for alpha = 1e-05
Log Loss : 1.8304997567764278
for alpha = 0.0001
Log Loss : 1.5521794301441487
for alpha = 0.001
Log Loss : 1.3131690297390557
for alpha = 0.01
Log Loss : 1.294741934169925
for alpha = 0.1
Log Loss : 1.2685354151040977
for alpha = 1
Log Loss : 1.1681194880399801
for alpha = 10
Log Loss : 1.2187007719349652
for alpha = 100
Log Loss : 1.5806374580906342
```



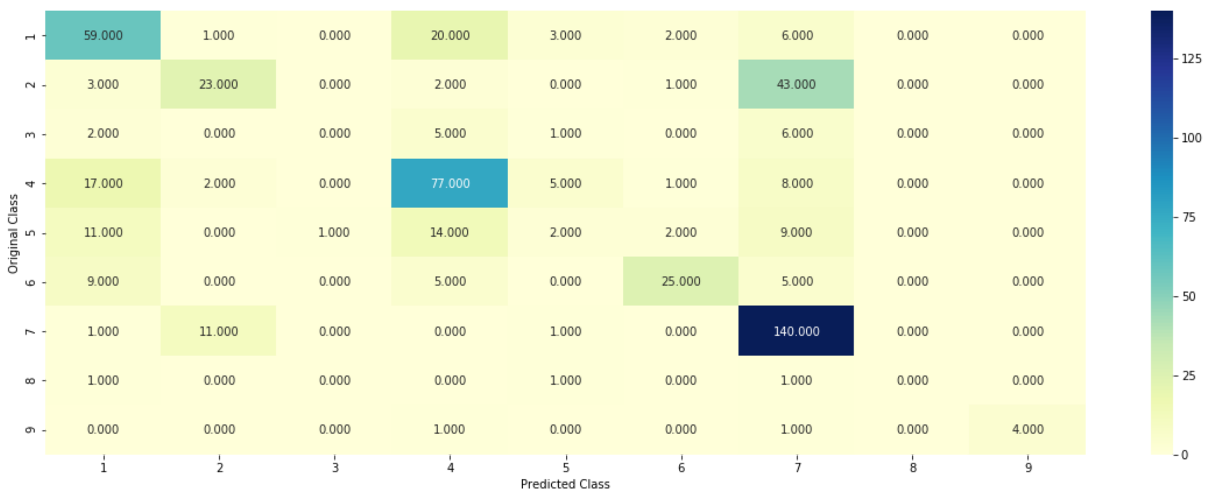
```
For values of best alpha =  1 The train log loss is: 0.80218709369438
09
For values of best alpha =  1 The cross validation log loss is: 1.168
1194880399801
For values of best alpha =  1 The test log loss is: 1.209578126168612
3
```

testing the model with best hyper parameter

In [0]:

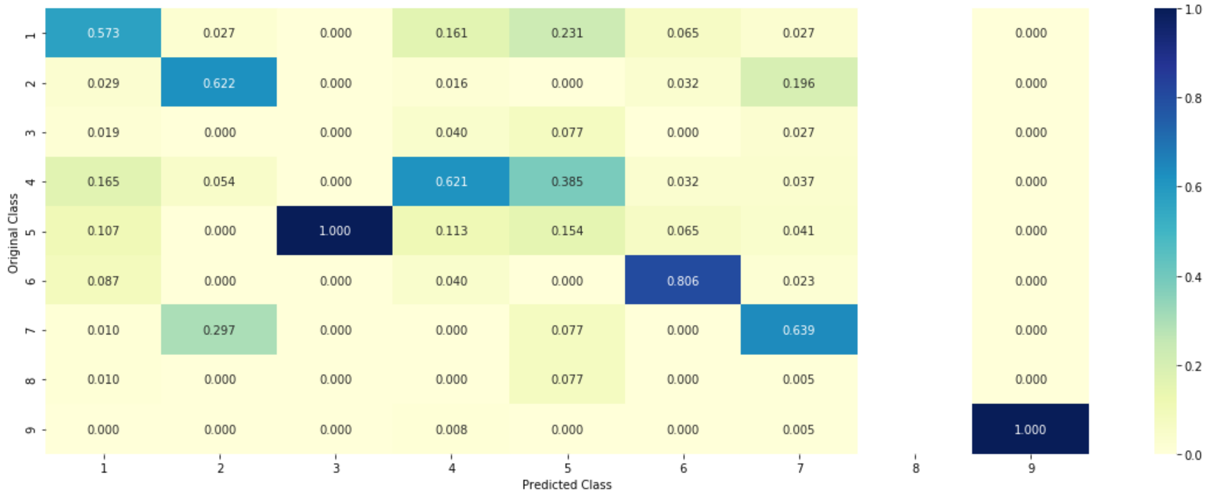
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(X_train_bi,y_train, X_cv_bi,y_cv, clf
)
```

```
Log loss : 1.1681194880399801
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```

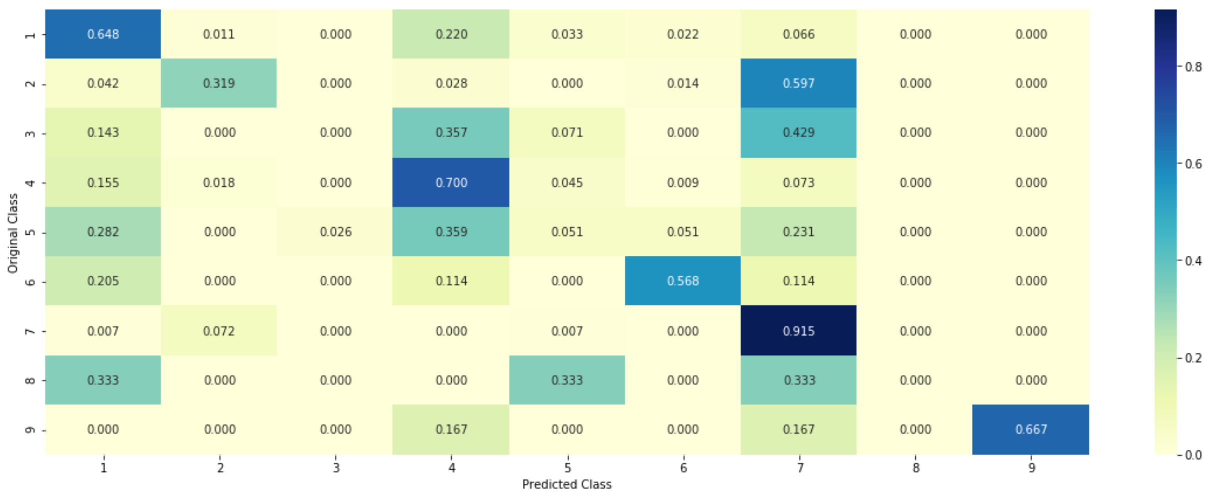


	1	2	3	4	5	6	7	8	9
1	59.000	1.000	0.000	20.000	3.000	2.000	6.000	0.000	0.000
2	3.000	23.000	0.000	2.000	0.000	1.000	43.000	0.000	0.000
3	2.000	0.000	0.000	5.000	1.000	0.000	6.000	0.000	0.000
4	17.000	2.000	0.000	77.000	5.000	1.000	8.000	0.000	0.000
5	11.000	0.000	1.000	14.000	2.000	2.000	9.000	0.000	0.000
6	9.000	0.000	0.000	5.000	0.000	25.000	5.000	0.000	0.000
7	1.000	11.000	0.000	0.000	1.000	0.000	140.000	0.000	0.000
8	1.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

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



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



In [0]:
```
from prettytable import PrettyTable
print('RESULTS OF TASK 3:for unigrams and bigrams')
x = PrettyTable()
x.field_names = ["Algorithm","n-gram used","Log Loss of Train", 'Log Lo
ss of CV',"Log Loss of Test","% of miss classification"]
x.add_row(["LR with weight Balancing","unigram",0.93,1.17,1.24,'38.34%'
])
x.add_row(["LR with weight Balancing","bigram",0.80,1.16,1.20,'37.96%'
])
print(x)
```

```
RESULTS OF TASK 3:for unigrams and bigrams
+--------------------------+-------------+-------------------+---------
-------+------------------+-------------------------+
|        Algorithm         | n-gram used | Log Loss of Train | Log Loss
of CV | Log Loss of Test | % of miss classification |
+--------------------------+-------------+-------------------+---------
-------+------------------+-------------------------+
| LR with weight Balancing |   unigram   |        0.93       |      1.1
7      |       1.24       |          38.34%          |
| LR with weight Balancing |    bigram   |        0.8        |      1.1
6      |       1.2        |          37.96%          |
+--------------------------+-------------+-------------------+---------
-------+------------------+-------------------------+
```

TASK 4

feature engineering

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
```

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

#-------------------------------------------
#one-hot encoding of Variation feature
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [46]:

```
#first taking top 1k features then applying it two all models
#below code for converting to tfidf with top 1k features and also ngram
=4
tf_idf_vect = TfidfVectorizer(ngram_range=(4,4),max_features=2000)
tf_idf_vect.fit(train_df.TEXT)
print("some sample features(unique words in the corpus)",tf_idf_vect.ge
t_feature_names()[0:10])
print('='*50)

X_train_tf_idf = tf_idf_vect.transform(train_df.TEXT)
X_test_tf_idf = tf_idf_vect.transform(test_df.TEXT)
X_cv_tf_idf = tf_idf_vect.transform(cv_df.TEXT)
#print("the type of count vectorizer ",type(X_train_tf_idf))
#print("the shape of out text TFIDF vectorizer ",X_train_tf_idf.get_sha
pe())
#print("the number of unique words including both unigrams and bigrams
 ", X_train_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['000 005 copyright 20
17', '005 copyright 2017 american', '0094 october 2013cancer discover
y', '10 1158 0008 5472', '10 1158 1078 0432', '10 1158 2159 8290', '10
fetal bovine serum', '10 fetal calf serum', '10 µg ml of', '100 in favo
r of']
==================================================
```

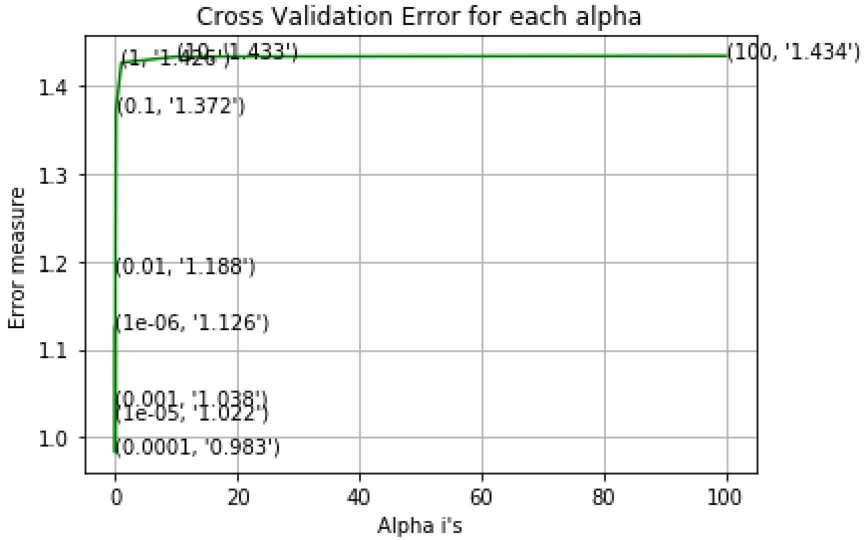
In [0]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))

X_train = hstack((train_gene_var_onehotCoding, X_train_tf_idf)).tocsr()
y_train = np.array(list(train_df['Class']))

X_test = hstack((test_gene_var_onehotCoding, X_test_tf_idf)).tocsr()
y_test = np.array(list(test_df['Class']))

X_cv = hstack((cv_gene_var_onehotCoding, X_cv_tf_idf)).tocsr()
y_cv = np.array(list(cv_df['Class']))
```

In [48]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(y_cv, 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], p
enalty='l2', loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.125688235581597
for alpha = 1e-05
Log Loss : 1.0224810059894618
for alpha = 0.0001
Log Loss : 0.9830168791725015
for alpha = 0.001
Log Loss : 1.037518890025571
for alpha = 0.01
Log Loss : 1.18840608403734
for alpha = 0.1
Log Loss : 1.371789581973373
for alpha = 1
Log Loss : 1.4263448870606337
for alpha = 10
Log Loss : 1.4331588741690648
```

```
for alpha = 100
Log Loss : 1.43397996299367
```

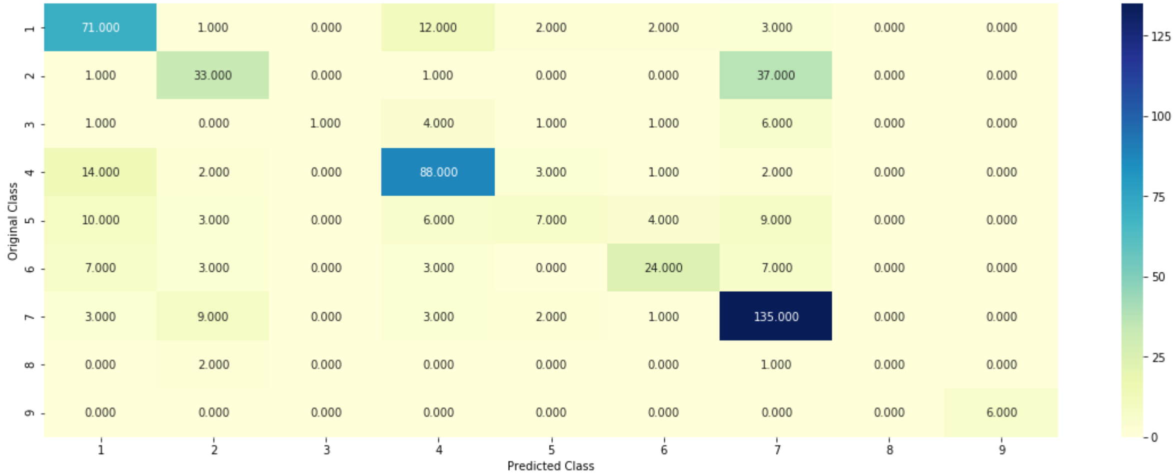



```
For values of best alpha =  0.0001 The train log loss is: 0.43603510541
85871
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9830168791725015
For values of best alpha =  0.0001 The test log loss is: 0.990456844211
3068
```

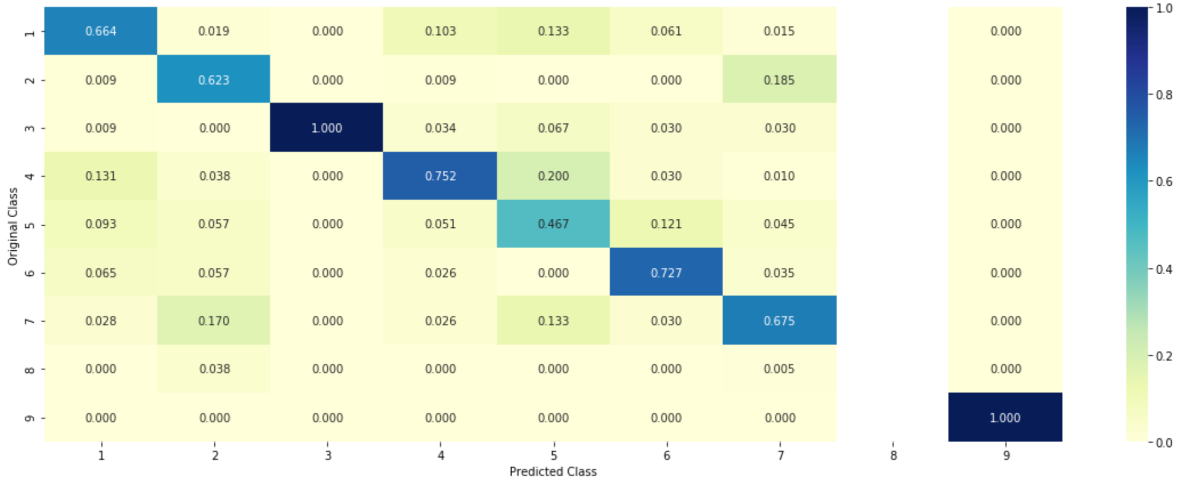
In [49]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(X_train, y_train, X_cv, y_cv, clf)
```

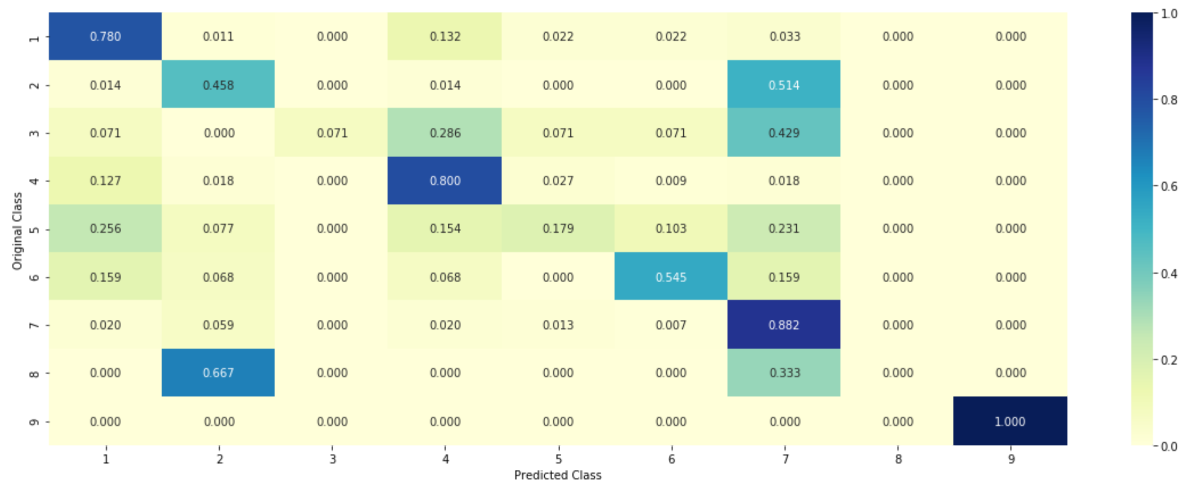
```
Log loss : 0.9830168791725015
Number of mis-classified points : 0.31390977443609025
-------------------- Confusion matrix --------------------
```



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



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



In [50]:

```
from prettytable import PrettyTable
print('RESULTS OF TASK 4:Feature engineering')
print("Here we used tfidf vectorizer with top 2000 features using Selec
tKBest")
x = PrettyTable()
x.field_names = ["Algorithm","n-gram used","Log Loss of Train", 'Log Lo
ss of CV',"Log Loss of Test","% of miss classification"]
x.add_row(["LR with weight Balancing","4-gram",0.45,0.9830,0.9904,'31.3
9%'])
print(x)
```

```
RESULTS OF TASK 4:Feature engineering
Here we used tfidf vectorizer with top 2000 features using SelectKBest
+--------------------------+-------------+-------------------+---------
-------+------------------+-------------------------+
|        Algorithm         | n-gram used | Log Loss of Train | Log Loss
of CV | Log Loss of Test | % of miss classification |
+--------------------------+-------------+-------------------+---------
-------+------------------+-------------------------+
| LR with weight Balancing |    4-gram   |        0.45       |     0.98
3      |       0.9904     |          31.39%          |
```

```
+------------------------+------------+-----------------+---------
-------+----------------+-----------------------+
```

In [0]: