In [4]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Reading Gene and Variation Data

In [5]:

```
training_var=pd.read_csv("Training/training_variants")
```

In [6]:

```
training_var.head(5)
```

Out[6]:

	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

Reading Text Data

In [7]:

```
training_text=pd.read_csv("Training/training_text",error_bad_lines=False,sep="\|\|",names=["Id","Te
xt"],engine="python",skiprows=1)
```

In [8]:

```
training_text.head(5)
```

Out[8]:

	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...

Preprocessing of text

In [9]:

```
import re
```

In [ ]:

In [10]:

```
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [11]:

```
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(r"won't", "will not", total_text)
```

```
        total_text = re.sub(r"won't", "will not", total_text)
        total_text = re.sub(r"can\'t", "can not", total_text)

    # general
        total_text = re.sub(r"n\'t", " not", total_text)
        total_text = re.sub(r"\'re", " are", total_text)
        total_text = re.sub(r"\'s", " is", total_text)
        total_text = re.sub(r"\'d", " would", total_text)
        total_text = re.sub(r"\'ll", " will", total_text)
        total_text = re.sub(r"\'t", " not", total_text)
        total_text = re.sub(r"\'ve", " have", total_text)
        total_text = re.sub(r"\'m", " am", total_text)
        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.replace('\\r', ' ')
        total_text = total_text.replace('\\"', ' ')
        total_text = total_text.replace('\\n', ' ')
        string=' '.join(e.lower() for e in total_text.split() if e.lower() not in stopwords)
        training_text[column][index] = string
```

In [12]:

```
for index, row in training_text.iterrows():
    if type(row['Text']) is str:
        nlp_preprocessing(row['Text'], index, 'Text')
    else:
        print("there is no text description for id:",index)
```

```
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
```

In [ ]:

```

```

In [13]:

```
training_var.columns=["Id","Gene","Variation","Class"]
```

In [14]:

```
train_data = pd.merge(training_var, training_text,on='Id', how='left')
train_data.head()
```

Out[14]:

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

In [15]:

```
train_data.shape
```

Out[15]:

```
(3321, 5)
```

In [16]:

```
train_data.loc[train_data['Text'].isnull(),'Text'] = train_data['Gene'] +' '+train_data['Variation'
]
```

Splitting data into train, test and cross validation

In [17]:

```
from sklearn.model_selection import train_test_split
y=train_data["Class"].values
x=train_data
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,test_size=0.2,stratify=y)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,train_size=0.8,test_size=0.2,stratify=y
_train)
```

In [18]:

```
print("shape of train data ")
print(x_train.shape)
print(y_train.shape)
print("shape of test data ")
print(x_test.shape)
print(y_test.shape)
print("shape of crossvalidation data ")
print(x_cv.shape)
print(y_cv.shape)
```

```
shape of train data
(2124, 5)
(2124,)
shape of test data
(665, 5)
(665,)
shape of crossvalidation data
(532, 5)
(532,)
```

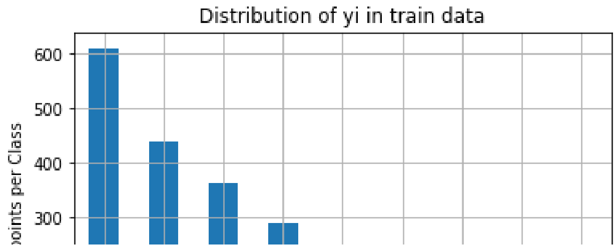
In [19]:

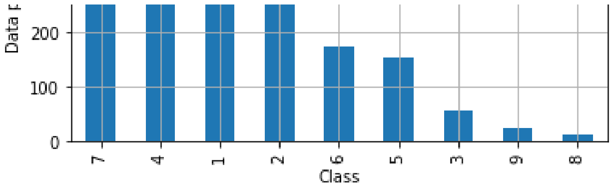
```
train_class_dist = x_train["Class"].value_counts()
test_class_dist = x_test["Class"].value_counts()
cv_class_dist= x_cv["Class"].value_counts()
```

Distribution of y_is in Train, Test and Cross Validation datasets

In [20]:

```
train_class_dist.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
sorted_yi = np.argsort(-train_class_dist.values)
for i in sorted_yi:
    print('Number of data points in class', train_class_dist.index[i], ':',train_class_dist.values[
i], '(', np.round((train_class_dist.values[i]/x_train.shape[0]*100), 3), '%)')
```

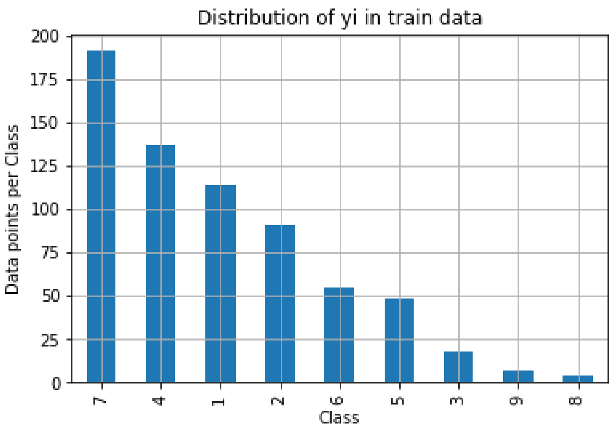





```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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

In [21]:

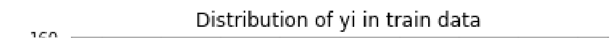
```
test_class_dist.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
sorted_yi = np.argsort(-test_class_dist.values)
for i in sorted_yi:
    print('Number of data points in class', test_class_dist.index[i], ':',test_class_dist.values[i]
, '(', np.round((test_class_dist.values[i]/x_test.shape[0]*100), 3), '%)')
```

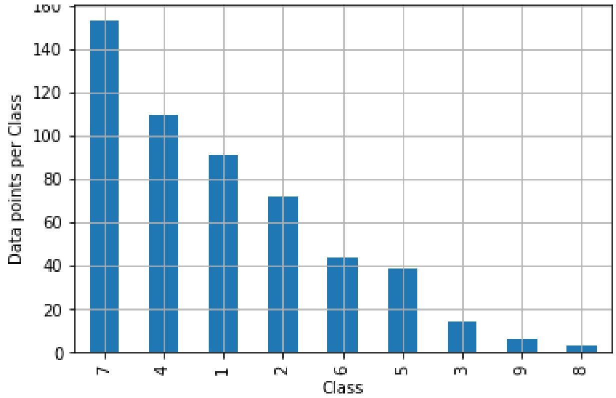



```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218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

In [22]:

```
cv_class_dist.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
sorted_yi = np.argsort(-cv_class_dist.values)
for i in sorted_yi:
    print('Number of data points in class', cv_class_dist.index[i], ':',cv_class_dist.values[i], '(
', np.round((cv_class_dist.values[i]/x_cv.shape[0]*100), 3), '%)')
```






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

In [23]:

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

    A =(C.T/C.sum(axis=1)).T

    B =(C/C.sum(axis=0))

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

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

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

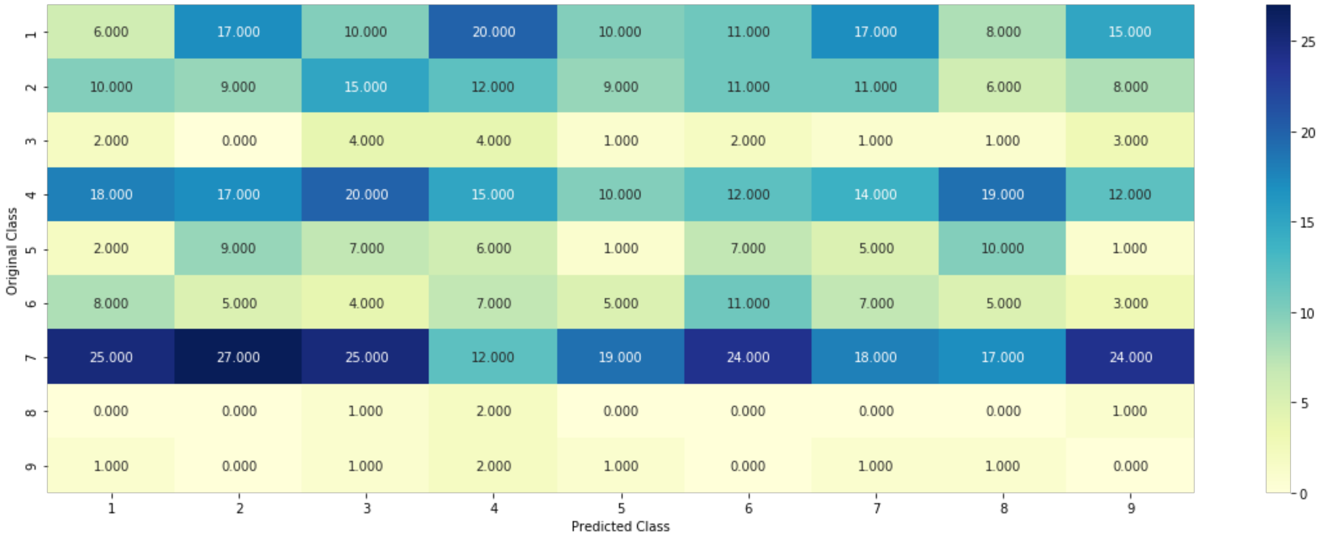
Prediction using a 'Random' Model

In [24]:

```
from sklearn.metrics import log_loss

test_data_len = x_test.shape[0]
cv_data_len = x_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
```

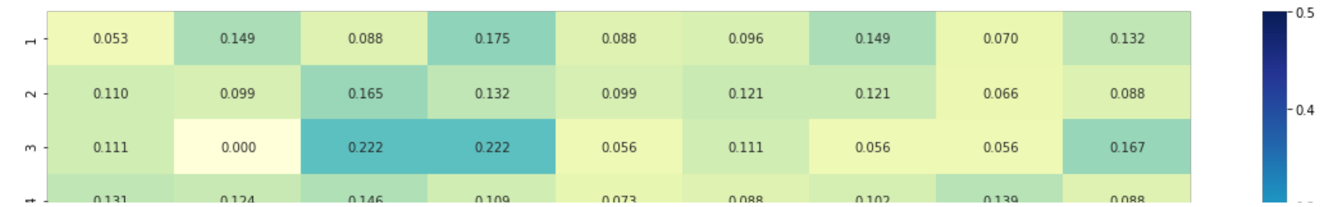
```
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


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

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

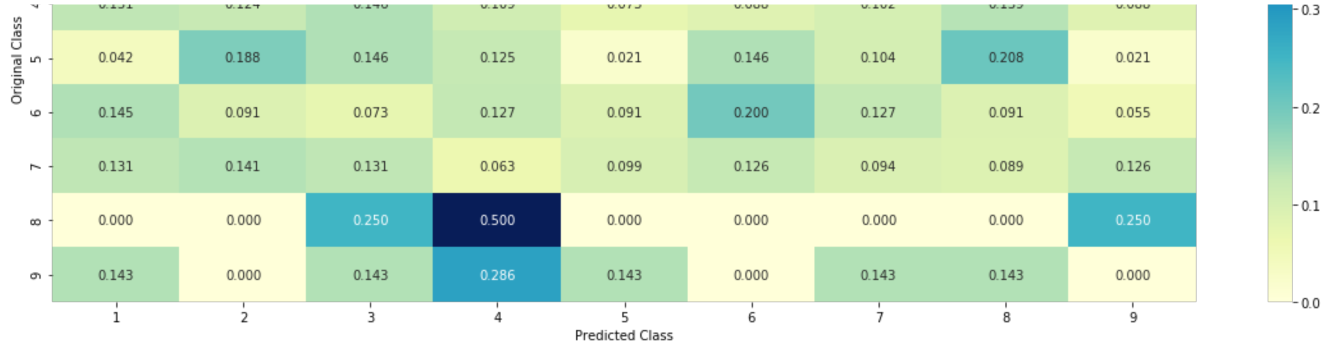
```
Log loss on Cross Validation Data using Random Model 2.4960308982309485
Log loss on Test Data using Random Model 2.5109878212850374
-------------------- Confusion matrix --------------------
```




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


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





In [25]:

```
def get_fea_dict(alpha, feature ):
    fea_dict=dict()
    value_count=x_train[feature].value_counts()
    for i , denominator in value_count.items():
        vec=[]
        for k in range(1,10):
            cnt=x_train[(x_train["Class"]==k) & (x_train[feature]==i)]
            vec.append((cnt.shape[0]+alpha*10)/(denominator+alpha*90))
        fea_dict[i]=vec
    return fea_dict

def get_fea(alpha,feature, df):
    fea_dict=get_fea_dict(alpha,feature)
    value_count=x_train[feature].value_counts()
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(fea_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
```

Univariate Analysis

Gene

In [26]:

```
unique_genes = x_train['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 : 229
BRCA1     179
TP53      111
PTEN       87
EGFR       86
BRCA2      85
KIT        64
ALK        53
BRAF       53
ERBB2      42
PIK3CA     38
Name: Gene, dtype: int64
```

In [27]:

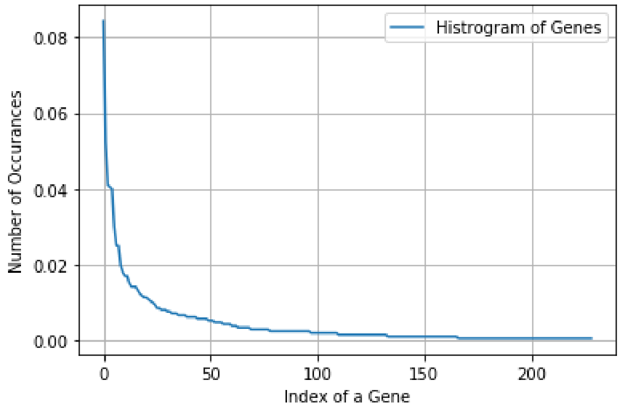
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
```

```
d they are distibuted as follows",)
```

Ans: There are 229 different categories of genes in the train data, and they are distibuted as follows

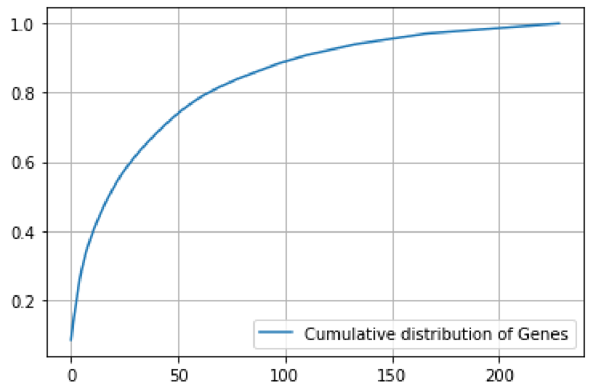
In [28]:

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



In [29]:

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



In [30]:

```
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_fea(alpha, "Gene", x_train))
# test gene feature
test_gene_feature_responseCoding = np.array(get_fea(alpha, "Gene", x_test))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_fea(alpha, "Gene", x_cv))
```

In [31]:

```
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
```

```
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

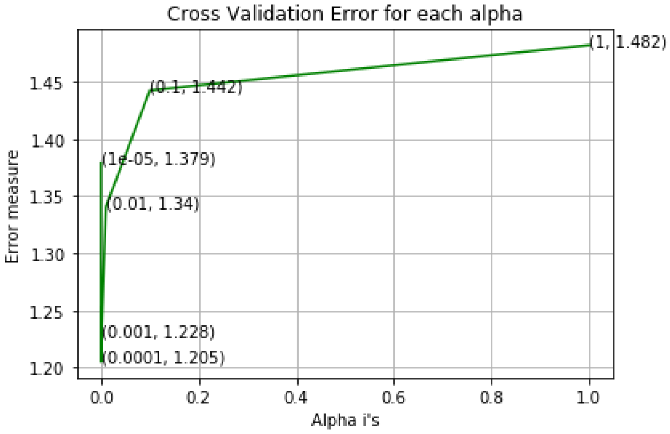
In [32]:

```
from sklearn import linear_model
from sklearn import calibration
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
cv_log_error_array=[]
for i in alpha:
    clf = linear_model.SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.3790533101496905
For values of alpha =  0.0001 The log loss is: 1.2049953666125688
For values of alpha =  0.001 The log loss is: 1.2283606860896765
For values of alpha =  0.01 The log loss is: 1.3404303580066426
For values of alpha =  0.1 The log loss is: 1.442292736203793
For values of alpha =  1 The log loss is: 1.4818232972490393
```

In [33]:

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



In [34]:

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

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

```
For values of best alpha =  0.0001 The train log loss is: 1.0331810250604543
For values of best alpha =  0.0001 The cross validation log loss is: 1.2049953666125688
For values of best alpha =  0.0001 The test log loss is: 1.2471468497903841
```

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

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

In [48]:

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

test_coverage=x_test[x_test['Gene'].isin(list(set(x_train['Gene'])))].shape[0]
cv_coverage=x_cv[x_cv['Gene'].isin(list(set(x_train['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',x_test.shape[0], ":",(test_coverage/x_test.sha
pe[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',x_cv.shape[0],":" ,(cv_coverage/x_cv.sha
pe[0])*100)
```

```
Q. How many data points in Test and CV datasets are covered by the  234  genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 515 out of  532 : 96.80451127819549
```

Variation

In [49]:

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

```
Number of Unique Variations : 1935
Truncating Mutations     58
Amplification            46
Deletion                 45
Fusions                  24
G12V                      3
Y64A                      2
TMPRSS2-ETV1 Fusion       2
I31M                      2
G12S                      2
Q61H                      2
Name: Variation, dtype: int64
```

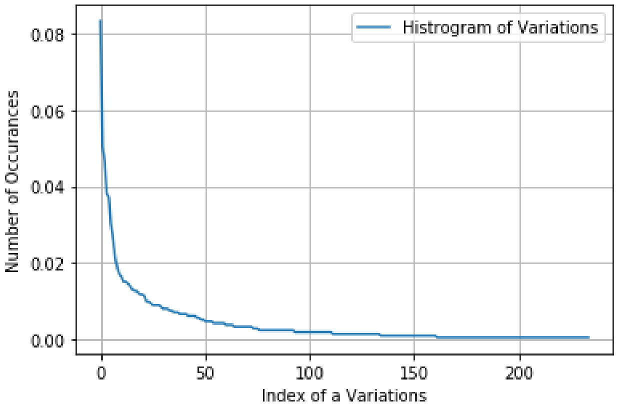
In [50]:

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

```
Ans: There are 1935 different categories of variations in the train data, and they are distibuted
as follows
```

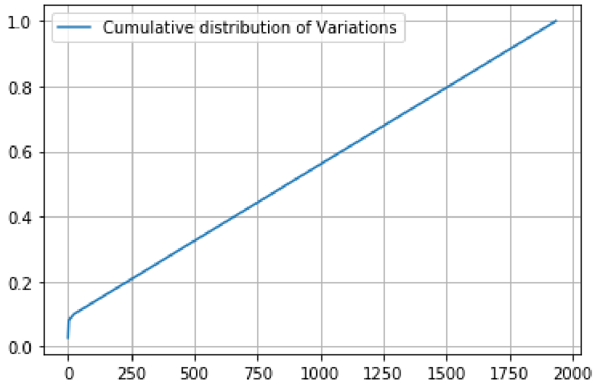
In [51]:

```
s1 = sum(unique_var.values);
h1 = unique_var.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variations')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [52]:

```
c1 = np.cumsum(h1)
plt.plot(c1,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```



In [53]:

```
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_fea(alpha, "Variation", x_train))
# test gene feature
test_variation_feature_responseCoding = np.array(get_fea(alpha, "Variation", x_test))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_fea(alpha, "Variation", x_cv))
```

In [54]:

```
var_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = var_vectorizer.fit_transform(x_train['Variation'])
test_variation_feature_onehotCoding = var_vectorizer.transform(x_test['Variation'])
cv_variation_feature_onehotCoding = var_vectorizer.transform(x_cv['Variation'])
```

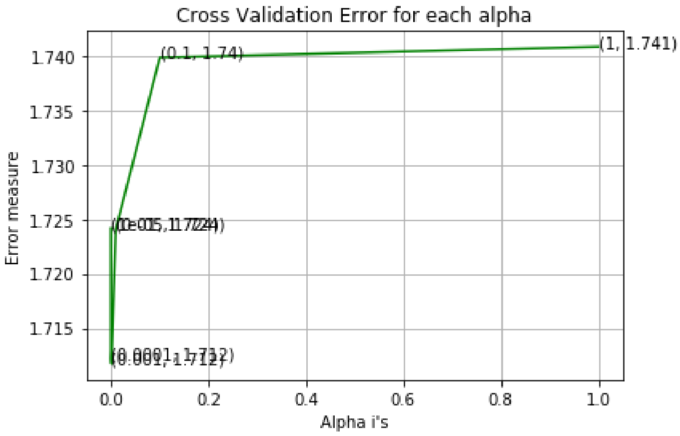
In [55]:

```
from sklearn import linear_model
from sklearn import calibration
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
cv_log_error_array=[]
for i in alpha:
    clf = linear_model.SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7241868979528971
For values of alpha =  0.0001 The log loss is: 1.712187866330471
For values of alpha =  0.001 The log loss is: 1.7116952131282506
For values of alpha =  0.01 The log loss is: 1.7240755351492818
For values of alpha =  0.1 The log loss is: 1.7398767944295868
For values of alpha =  1 The log loss is: 1.7408702563671674
```

In [56]:

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



In [57]:

```
best_alpha1 = np.argmin(cv_log_error_array)
clf = linear_model.SGDClassifier(alpha=alpha[best_alpha1], penalty='l2', loss='log', random_state=4
2)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha1], "The train log loss is:",log_loss(y_train
, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha1], "The cross validation log loss is:",log_l
oss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha1], "The test log loss is:",log_loss(y_test,
predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of best alpha =  0.001 The train log loss is: 1.2225431021086794
For values of best alpha =  0.001 The cross validation log loss is: 1.7116952131282506
For values of best alpha =  0.001 The test log loss is: 1.6742411888664148
```

In [58]:

```
print("Q12. How many data points are covered by total ", unique_var.shape[0], " genes in test and
cross validation data sets?")
test_coverage=x_test[x_test['Variation'].isin(list(set(x_train['Variation'])))].shape[0]
cv_coverage=x_cv[x_cv['Variation'].isin(list(set(x_train['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',x_test.shape[0], ":",(test_coverage/x_test.sha
pe[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',x_cv.shape[0],":" ,(cv_coverage/x_cv.sha
pe[0])*100)
```

```
Q12. How many data points are covered by total  1935  genes in test and cross validation data
```

Q12. How many data points are covered by total 1955 genes in test and cross validation data sets?
Ans
1. In test data 78 out of 665 : 11.729323308270677
2. In cross validation data 52 out of 532 : 9.774436090225564

Univariate Analysis on Text Feature

In [42]:

```
import collections
dictionary = collections.defaultdict(int)
for index, row in x_train.iterrows():
    for word in row['Text'].split():

        dictionary[word] +=1
#print("The number of unique words is ",str(len(dictionary.keys())))
```

In [43]:

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

In [44]:

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

In [45]:

```
text_vectorizer = TfidfVectorizer(max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['Text'].values.astype('U'))
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [46]:

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

```
    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(x_train)


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 [47]:

```
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)
```

In [48]:

```
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding =
(test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [49]:

```
from sklearn.preprocessing import StandardScaler
train_text_feature_onehotCoding=
StandardScaler(with_mean=False).fit_transform(train_text_feature_onehotCoding)


# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['Text'])
# don't forget to normalize every feature
test_text_feature_onehotCoding =
StandardScaler(with_mean=False).fit_transform(test_text_feature_onehotCoding)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['Text'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding =
StandardScaler(with_mean=False).fit_transform(cv_text_feature_onehotCoding)
```

In [68]:

```
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [69]:

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

```
Counter({256.0324849055361: 1, 178.3816624374742: 1, 140.45984613867194: 1, 130.43066735676106: 1,
127.75542145631107: 1, 119.1917435308816: 1, 118.85908052062645: 1, 115.40209909444846: 1,
111.09683674714152: 1, 105.52230930150519: 1, 103.76810391729184: 1, 103.71771595368224: 1,
92.25913092142532: 1, 89.7875761974773: 1, 86.21251190242434: 1, 84.65864397567624: 1,
82.4526641530949: 1, 80.92591448985857: 1, 79.5302635251487: 1, 79.3277873849562: 1,
78.78137294125482: 1, 74.29432390431106: 1, 72.38097872607801: 1, 71.24776461462177: 1,
71.08299564211984: 1, 68.51611350997787: 1, 68.3630594952936: 1, 67.9686128541555: 1,
66.38866108326388: 1, 65.20428769727813: 1, 64.29466598315388: 1, 64.1974683002231: 1,
62.52195452819578: 1, 62.453302796317125: 1, 60.06071876067673: 1, 59.34563855450449: 1,
56.789968886250904: 1, 56.68645464092824: 1, 56.31809489748819: 1, 54.618173073542735: 1,
51.20254042650178: 1, 50.89220738417602: 1, 50.8336010088054: 1, 49.280884644977675: 1,
48.56442427225561: 1, 47.38025829743799: 1, 45.72915516811051: 1, 45.59686491175654: 1,
```

45.544662771423845: 1, 45.521077157449184: 1, 45.42606393645052: 1, 44.86485681434462: 1,
44.0921983862502: 1, 43.76612257626823: 1, 43.41422810672954: 1, 43.387640913565676: 1,
43.3554700334962: 1, 43.22964945752437: 1, 42.16160438661359: 1, 41.94758203756765: 1,
41.64791469657831: 1, 41.54059805120249: 1, 40.935586229473856: 1, 40.832687000716625: 1,
40.723528019324434: 1, 40.421431694345934: 1, 39.60511183509434: 1, 39.301673372394156: 1,
38.99134798573894: 1, 38.87729669467892: 1, 38.694218381349714: 1, 38.65992237549571: 1,
38.505398231569316: 1, 38.14562675717343: 1, 38.05473482360706: 1, 37.966651358318856: 1,
37.142702156139755: 1, 36.93563320423913: 1, 36.75962158295182: 1, 36.570305257318324: 1,
36.009847205582666: 1, 35.907837081216265: 1, 35.89908137586457: 1, 35.58920862173034: 1,
35.39754058170106: 1, 34.997580044374295: 1, 34.797184751596276: 1, 33.85722715727683: 1,
33.70480074304447: 1, 33.63412410646113: 1, 33.56931699692082: 1, 33.381399622119176: 1,
33.154430052265155: 1, 33.086821678124466: 1, 32.759043420064145: 1, 32.61759621885425: 1,
32.51685046575764: 1, 32.40109686825218: 1, 32.39709646321827: 1, 32.286460625913335: 1,
31.869421277009188: 1, 31.809344603237086: 1, 31.73722441327498: 1, 31.658718788491207: 1,
31.609549139866342: 1, 31.60096040031878: 1, 31.426999580797407: 1, 31.342941833329686: 1,
31.31113245153246: 1, 31.158472965909866: 1, 30.875167451168778: 1, 30.7783169642084: 1,
30.742575662331888: 1, 30.7039018729751: 1, 30.61107431063841: 1, 30.572003694891347: 1,
30.528512661125983: 1, 30.343440110254704: 1, 30.292084532941825: 1, 30.261049553345373: 1,
30.195303962069282: 1, 29.720927686513267: 1, 29.47528081470705: 1, 29.177977308143166: 1,
29.121044668479655: 1, 29.110861470843883: 1, 29.102991690811297: 1, 29.08117751692162: 1,
29.02087654530322: 1, 28.624079511277092: 1, 28.429617988451042: 1, 28.170266252203078: 1,
28.16748759354708: 1, 28.089634376048213: 1, 27.93338157868159: 1, 27.743196157079264: 1,
27.740396941971216: 1, 27.69239534091688: 1, 27.490948375865024: 1, 27.131302851194512: 1,
27.05090091050544: 1, 27.00768869028692: 1, 26.656393210973167: 1, 26.649710422476534: 1,
26.610359313361506: 1, 26.55502358351552: 1, 26.4308597326911: 1, 26.108469600536676: 1,
26.051659687464845: 1, 25.97115623157136: 1, 25.802934411336995: 1, 25.753876368279357: 1,
25.69286540603384: 1, 25.528889931941663: 1, 25.429206911607864: 1, 25.333137948665915: 1,
25.30312375126414: 1, 25.044018006795795: 1, 24.97041772309181: 1, 24.912283621310127: 1,
24.87587087226879: 1, 24.786981442620988: 1, 24.74998958818173: 1, 24.62725823678452: 1,
24.464064924936405: 1, 24.34377407454752: 1, 24.18713281378215: 1, 24.187114228486497: 1,
24.089991435175087: 1, 24.08967727698999: 1, 24.075815995606632: 1, 24.012024924013318: 1,
23.884632881222604: 1, 23.787989590871323: 1, 23.752240980369795: 1, 23.658703656548457: 1,
23.495287943789567: 1, 23.480638042813013: 1, 23.418232649049564: 1, 23.383216371390212: 1,
23.382301924401613: 1, 23.379990719724507: 1, 23.377324755924015: 1, 23.36854761621814: 1,
23.35141124915804: 1, 23.086883563562058: 1, 23.07889219103022: 1, 23.06585232382933: 1,
23.052703111238838: 1, 23.02012891524968: 1, 22.84064522156017: 1, 22.83691301592171: 1, 22.8122992
9611076: 1, 22.67409340675974: 1, 22.643926324725143: 1, 22.630516700193724: 1, 22.41538438592613:
1, 22.322492679638707: 1, 22.297365090720465: 1, 22.238057806891813: 1, 22.18841328303785: 1,
22.160551045799053: 1, 22.147346717392157: 1, 22.13007080208893: 1, 22.09583243945118: 1,
22.079686567334598: 1, 22.049717852493433: 1, 22.039533419501122: 1, 21.999006444563133: 1,
21.882649504214235: 1, 21.830523802174238: 1, 21.830056331854426: 1, 21.81973104134781: 1,
21.75606606891291: 1, 21.606169613521846: 1, 21.602804424370216: 1, 21.57768850200562: 1,
21.547683015829865: 1, 21.544520527395804: 1, 21.47553739031508: 1, 21.4324015969022: 1,
21.40711719823009: 1, 21.333955738462475: 1, 21.329924683816344: 1, 21.29850625292857: 1,
21.29447060135074: 1, 21.258227039905414: 1, 21.237154342105555: 1, 21.220791893858266: 1,
21.167439625340478: 1, 21.13216304933544: 1, 21.131403826012615: 1, 21.090624244293863: 1,
21.044390840622842: 1, 20.92440868649175: 1, 20.891766632348215: 1, 20.864305099375798: 1,
20.85410724879033: 1, 20.818399626515166: 1, 20.809107464315463: 1, 20.65387475729441: 1,
20.64701345232972: 1, 20.56013741742348: 1, 20.526753234991453: 1, 20.37207821609942: 1,
20.349616961166866: 1, 20.3074829710851: 1, 20.220625890473645: 1, 20.21530934981141: 1,
20.177130973250378: 1, 20.15477377575323: 1, 20.111318100939027: 1, 20.077096956226647: 1,
20.04855814812951: 1, 19.997941625593942: 1, 19.939157150036383: 1, 19.772914928311682: 1,
19.695952736949287: 1, 19.662510693710868: 1, 19.62718474039339: 1, 19.610187283957988: 1,
19.552071728144124: 1, 19.461721552832735: 1, 19.422432477315798: 1, 19.42149344645637: 1,
19.305250173613967: 1, 19.302692312525764: 1, 19.297973014653593: 1, 19.250762123832033: 1,
19.175715907801173: 1, 19.171744796193597: 1, 19.167087718300323: 1, 19.136981272302037: 1,
19.07827245121586: 1, 19.064813791429223: 1, 19.00608554874164: 1, 18.984317758443535: 1,
18.917751576566175: 1, 18.897573729680683: 1, 18.878843610092037: 1, 18.876069846613895: 1,
18.874160816250296: 1, 18.863394513520785: 1, 18.761029767452243: 1, 18.759675043486286: 1,
18.723473152969042: 1, 18.71529671056424: 1, 18.60417972396934: 1, 18.599473064120605: 1,
18.554641567323923: 1, 18.5496511950768: 1, 18.502362373771167: 1, 18.47624761719784: 1,
18.349999586604188: 1, 18.2682135081336: 1, 18.149188391537507: 1, 18.143651863708822: 1,
18.14245628504526: 1, 18.13748542025271: 1, 18.119270503052864: 1, 18.079151445249135: 1,
18.01052465680708: 1, 18.009526274148776: 1, 17.96410541384796: 1, 17.935629956120348: 1,
17.89911484934852: 1, 17.837510540952813: 1, 17.832381981364087: 1, 17.762700271995545: 1,
17.76137090650753: 1, 17.68166864228543: 1, 17.60080270419125: 1, 17.592448771461143: 1,
17.5805096744489: 1, 17.5663160573159: 1, 17.565435439373594: 1, 17.534465598742514: 1,
17.534252597771356: 1, 17.528919590403152: 1, 17.521506830052157: 1, 17.517314529771642: 1,
17.48737634203834: 1, 17.453692423212512: 1, 17.445762930741875: 1, 17.426691988559462: 1,
17.41703910857034: 1, 17.407040748118263: 1, 17.40255085205233: 1, 17.396274594582874: 1,
17.29612187888562: 1, 17.252924367370138: 1, 17.245120915093757: 1, 17.235086263104197: 1,
17.232048545417033: 1, 17.21941393523616: 1, 17.193255909017996: 1, 17.152594599063868: 1,
17.131531096831985: 1, 17.085237821686057: 1, 17.08484732262983: 1, 17.05424999753784: 1,
17.05110871358835: 1, 17.043797595530105: 1, 17.03144148971319: 1, 16.948692629937494: 1,
16.937177739710158: 1, 16.923989493530904: 1, 16.868580801992792: 1, 16.844875152354916: 1,
16.840103071609914: 1, 16.821228636142376: 1, 16.79285332537393: 1, 16.74553915854032: 1,
16.698778723384017: 1, 16.68483689072582: 1, 16.68359097370887: 1, 16.674617031908216: 1,

16.64878255008816: 1, 16.638946486225475: 1, 16.632648012013444: 1, 16.62500593063709: 1,
16.543753499267694: 1, 16.43268010016387: 1, 16.42898981874059: 1, 16.411647623500755: 1,
16.408545947200025: 1, 16.39808910832705: 1, 16.382421021289517: 1, 16.357639538904422: 1,
16.30781111543063: 1, 16.297961379886846: 1, 16.29793059432529: 1, 16.25337594068664: 1,
16.22687813975127: 1, 16.19988957200225: 1, 16.14022970834242: 1, 16.120312285188707: 1,
16.089129199237707: 1, 16.03679568012648: 1, 15.99256768494451: 1, 15.98961748781362: 1, 15.9744193
8844258: 1, 15.882367472620937: 1, 15.86040067275781: 1, 15.841308831601822: 1,
15.840112241343123: 1, 15.81093519284755: 1, 15.803422345668242: 1, 15.795220870300895: 1,
15.748172096412981: 1, 15.730679414920926: 1, 15.61696336344569: 1, 15.597378161707574: 1,
15.570388683113027: 1, 15.555883453561703: 1, 15.551273736461876: 1, 15.510993265679783: 1,
15.450985165483099: 1, 15.449262183085338: 1, 15.44297943856704: 1, 15.430750650335806: 1,
15.396072812088322: 1, 15.384398548787054: 1, 15.380679897543828: 1, 15.372946277443875: 1,
15.36736079130563: 1, 15.330063235446497: 1, 15.328846334439815: 1, 15.298486385282718: 1,
15.255319015545435: 1, 15.238370809779207: 1, 15.186630429298448: 1, 15.13712248921759: 1,
15.126002532036457: 1, 15.078254817611272: 1, 15.074760844572204: 1, 15.024544210086942: 1,
15.01307406387257: 1, 15.002081563109499: 1, 14.989558457646503: 1, 14.986099590532017: 1,
14.92625060800574: 1, 14.910267066160127: 1, 14.89934309211074: 1, 14.886839967765807: 1,
14.88199682379035: 1, 14.846959578585295: 1, 14.826819306118717: 1, 14.812506493359386: 1,
14.783244451857431: 1, 14.78148275481311: 1, 14.743694541447876: 1, 14.735256962646183: 1,
14.724262430083805: 1, 14.718648859876476: 1, 14.714113007628718: 1, 14.677451239969841: 1,
14.671876059905491: 1, 14.66557488344462: 1, 14.659805177439772: 1, 14.658087896400355: 1,
14.652220450804252: 1, 14.61292927542616: 1, 14.6110522198506: 1, 14.606318946331221: 1,
14.592050421909478: 1, 14.577061394140785: 1, 14.529323710762714: 1, 14.510883314089812: 1,
14.49844368215094: 1, 14.480369854651562: 1, 14.477326631778707: 1, 14.44869934432505: 1,
14.426187587071478: 1, 14.412362813405977: 1, 14.371955460731943: 1, 14.304937953492844: 1,
14.287502457012272: 1, 14.267109644501126: 1, 14.240733376031496: 1, 14.209139116184321: 1,
14.19386576753029: 1, 14.192017977311874: 1, 14.183185138916565: 1, 14.127303518265412: 1,
14.104900956570795: 1, 14.029076175158757: 1, 14.021596010010573: 1, 13.950519185186137: 1,
13.94893211710736: 1, 13.931402563768678: 1, 13.931145839423493: 1, 13.92839980427078: 1,
13.87247347878609: 1, 13.854552768103437: 1, 13.831685941389853: 1, 13.825486378470854: 1,
13.7584486099282: 1, 13.738651696405817: 1, 13.701238303201723: 1, 13.697881089234075: 1,
13.686046580319651: 1, 13.613862765984406: 1, 13.57326605308302: 1, 13.557495097539453: 1,
13.520405499976722: 1, 13.514714907953504: 1, 13.51220262564316: 1, 13.507402125291323: 1,
13.498201163184563: 1, 13.481059589860527: 1, 13.477536437751322: 1, 13.471436832808973: 1,
13.447509433004571: 1, 13.435736324841905: 1, 13.39171805632893: 1, 13.385267201126759: 1,
13.379802437382235: 1, 13.373954804657789: 1, 13.328879940770262: 1, 13.315202044058077: 1,
13.308857742649591: 1, 13.299364602842582: 1, 13.22405298347616: 1, 13.196874254669384: 1,
13.117396273997228: 1, 13.082934564141835: 1, 13.07806343568439: 1, 13.072795232402953: 1,
13.067643695601348: 1, 13.021199840239346: 1, 13.019313362911879: 1, 13.00319294996925: 1,
12.937561299976688: 1, 12.898725169590795: 1, 12.864609811918054: 1, 12.846467203996491: 1,
12.840743418706014: 1, 12.837197960195663: 1, 12.831636748089203: 1, 12.807183184973853: 1,
12.797489879255163: 1, 12.786523823638673: 1, 12.752971192608506: 1, 12.745899658818262: 1,
12.742123977116186: 1, 12.73674693348271: 1, 12.72685349095316: 1, 12.705427232954012: 1,
12.699778153939523: 1, 12.689420011784996: 1, 12.66660177623283: 1, 12.656673934873538: 1,
12.637631184078716: 1, 12.629097893070425: 1, 12.597326289753347: 1, 12.593983743568746: 1,
12.570710290001843: 1, 12.566307337870741: 1, 12.545727341082205: 1, 12.533397974986839: 1,
12.525154666133483: 1, 12.515883779125314: 1, 12.489239768847595: 1, 12.475228259364753: 1,
12.431587257275753: 1, 12.392433375839408: 1, 12.385489163361237: 1, 12.375725970810025: 1,
12.359436307529792: 1, 12.352338456356646: 1, 12.347428872631568: 1, 12.307073021380953: 1,
12.304521506708737: 1, 12.280782727894094: 1, 12.26672784101839: 1, 12.220587222035718: 1,
12.21073307219019: 1, 12.196485288173985: 1, 12.174440163103428: 1, 12.154012310734736: 1,
12.153932455734951: 1, 12.149893582717286: 1, 12.140246319122639: 1, 12.125906299850715: 1,
12.120138402287669: 1, 12.111841457076821: 1, 12.111090680238364: 1, 12.068377317341128: 1,
12.063727356630825: 1, 12.062351027797307: 1, 12.037191779911128: 1, 12.027767907537319: 1,
12.002464936533537: 1, 11.992948300751076: 1, 11.975257188081486: 1, 11.973296092211365: 1,
11.960085135047143: 1, 11.959609194558716: 1, 11.936394028374973: 1, 11.873276391213903: 1,
11.872413902585514: 1, 11.857616748789727: 1, 11.85579723074533: 1, 11.851981545714303: 1,
11.843945787819875: 1, 11.837715354647568: 1, 11.836663131707224: 1, 11.778927482014474: 1,
11.769579447624489: 1, 11.763980595631926: 1, 11.760775038290223: 1, 11.720986723196306: 1,
11.692384977342197: 1, 11.672418743332589: 1, 11.648741190704587: 1, 11.632594529183736: 1,
11.61130283808665: 1, 11.60525014373658: 1, 11.591713076107336: 1, 11.570548208423618: 1,
11.558421207486532: 1, 11.55367359583606: 1, 11.535811950006737: 1, 11.516041763532336: 1,
11.515958033063518: 1, 11.499623239435282: 1, 11.47220723734634: 1, 11.458673075650838: 1,
11.435463677947942: 1, 11.403493225991735: 1, 11.39439036649137: 1, 11.393537666281372: 1,
11.373573535221828: 1, 11.36324767394146: 1, 11.338600554481541: 1, 11.336695310881597: 1,
11.333552886772813: 1, 11.325224614717078: 1, 11.307788778642143: 1, 11.302214178711315: 1,
11.287489917499823: 1, 11.277909841813678: 1, 11.217185124411099: 1, 11.21083238939135: 1,
11.164232980640652: 1, 11.154176732017534: 1, 11.14775700126706: 1, 11.14346958440377: 1,
11.123518448135956: 1, 11.122856473268042: 1, 11.105591354245574: 1, 11.095176863784914: 1,
11.083221191440016: 1, 11.076999456014963: 1, 11.070404464732446: 1, 11.06487784208289: 1,
11.059385547386373: 1, 11.054281781168521: 1, 11.053987176807036: 1, 11.04255087433312: 1,
11.025139837128494: 1, 10.996902048337233: 1, 10.990938679221657: 1, 10.982200826222556: 1,
10.974879414481512: 1, 10.948307203527499: 1, 10.946364954951125: 1, 10.919676818698463: 1,
10.909237350800451: 1, 10.899453991904496: 1, 10.89483618807983: 1, 10.875337648416448: 1,
10.865166293247212: 1, 10.840906411956947: 1, 10.814012690504654: 1, 10.770076383060765: 1,
10.769674893414937: 1, 10.74868532787153: 1, 10.748228522374912: 1, 10.742918369299838: 1,
10.733770191634386: 1, 10.730829001396701: 1, 10.71457882382519: 1, 10.705728844983396: 1,

10.695999332952592: 1, 10.69061079602558: 1, 10.683343730066417: 1, 10.654229878400795: 1,
10.629336715468881: 1, 10.625246223642767: 1, 10.60685589740998: 1, 10.600483360979846: 1,
10.588637177662674: 1, 10.576374166900044: 1, 10.529628278523393: 1, 10.518247567074694: 1,
10.515893181065945: 1, 10.504304081882635: 1, 10.498845614654284: 1, 10.472525196055809: 1,
10.460952589208903: 1, 10.454057675849661: 1, 10.415846716882495: 1, 10.413294978815168: 1,
10.399637311106613: 1, 10.392204709630494: 1, 10.391789803339805: 1, 10.387119518177107: 1,
10.386338543933613: 1, 10.357766001249928: 1, 10.353147416976393: 1, 10.350687359825292: 1,
10.3283864343196: 1, 10.306593151505385: 1, 10.301024395996718: 1, 10.297399129085507: 1,
10.292292480342487: 1, 10.285201861982642: 1, 10.252461699810558: 1, 10.2397204037254: 1,
10.225175256862064: 1, 10.215822515113917: 1, 10.204072538396554: 1, 10.18816080731716: 1,
10.178917617006894: 1, 10.159891105421357: 1, 10.150485391327603: 1, 10.11531081528246: 1,
10.11454385092215: 1, 10.101853946002883: 1, 10.067990448215655: 1, 10.064835995871848: 1,
10.06075625380461: 1, 10.043830775560195: 1, 10.033061868834611: 1, 10.01159593695807: 1,
10.011350837680657: 1, 9.999571384625376: 1, 9.99481297029434: 1, 9.98848749929868: 1,
9.971399826864369: 1, 9.951103668343432: 1, 9.91867881082361: 1, 9.916568106168713: 1,
9.91356750274851: 1, 9.904569289570786: 1, 9.899342846214617: 1, 9.896540194223219: 1,
9.883112804944817: 1, 9.870675839983095: 1, 9.85770795101326: 1, 9.847110428375677: 1,
9.833258712754253: 1, 9.831481478900544: 1, 9.816775726008505: 1, 9.811725835156619: 1,
9.809807969737848: 1, 9.807736324248085: 1, 9.803341579301335: 1, 9.80039847909238: 1,
9.772528113120563: 1, 9.76758471741519: 1, 9.757454752468222: 1, 9.75074187564558: 1,
9.723090384528238: 1, 9.722066640675134: 1, 9.710695882996559: 1, 9.685560632017166: 1,
9.669301788171875: 1, 9.660843867438833: 1, 9.636396237124174: 1, 9.635815312488: 1,
9.635567634940735: 1, 9.632973539637883: 1, 9.62172896367493: 1, 9.609348983698679: 1,
9.601620674505215: 1, 9.594907039080805: 1, 9.58283697606048: 1, 9.562484345873301: 1,
9.560254376373523: 1, 9.551259335364534: 1, 9.526133251700198: 1, 9.504944129730628: 1,
9.490320716223417: 1, 9.484368787147975: 1, 9.478006687376029: 1, 9.473464803191881: 1,
9.467727848447371: 1, 9.46659455629809: 1, 9.466328891674012: 1, 9.462596685521632: 1,
9.453705006896953: 1, 9.42776253920146: 1, 9.423436855515071: 1, 9.413427387066799: 1,
9.408204377828527: 1, 9.399012769189513: 1, 9.381806392994081: 1, 9.379961148695568: 1,
9.37480106907153: 1, 9.36752545598058: 1, 9.356527610210751: 1, 9.35442693793468: 1,
9.34721765556578: 1, 9.340349063672415: 1, 9.332080431765966: 1, 9.312984155596757: 1,
9.294677800631478: 1, 9.290879551791072: 1, 9.288693161773153: 1, 9.284823269856382: 1,
9.272430945244965: 1, 9.26952646653055: 1, 9.253982372252489: 1, 9.249123139731553: 1,
9.243621830272653: 1, 9.219283646935715: 1, 9.209764110812866: 1, 9.207337640561832: 1,
9.207254941590284: 1, 9.202205005300389: 1, 9.201535913308405: 1, 9.190008482730287: 1,
9.16737628200644: 1, 9.1607324567973: 1, 9.148658396033015: 1, 9.145410386164635: 1,
9.124003527090135: 1, 9.108055496437027: 1, 9.094545131644065: 1, 9.075515763678439: 1,
9.07012017725075: 1, 9.069893790435032: 1, 9.069238985030875: 1, 9.054142340580398: 1,
9.050804574496938: 1, 9.044785594505338: 1, 9.043486088208141: 1, 9.0363242401017: 1,
9.015744726253196: 1, 9.00263901767358: 1, 8.995543388634683: 1, 8.981436907818072: 1,
8.946963240130463: 1, 8.941080873166474: 1, 8.939771882487452: 1, 8.939514041233608: 1,
8.935308680825754: 1, 8.930052275606544: 1, 8.922796782434869: 1, 8.916761741019984: 1,
8.905408822980794: 1, 8.893524267737451: 1, 8.872510091467799: 1, 8.86961752672653: 1,
8.852953622272517: 1, 8.85256341083158: 1, 8.820424218665622: 1, 8.80909376366318: 1,
8.77374167383432: 1, 8.769104873675282: 1, 8.766548968034495: 1, 8.74642636039956: 1,
8.7445116808962: 1, 8.741110595353284: 1, 8.740640857383681: 1, 8.730236253134402: 1,
8.72567299550526: 1, 8.718168261921477: 1, 8.717354819164234: 1, 8.716084703140616: 1,
8.69584461628832: 1, 8.686197981768363: 1, 8.664936747773583: 1, 8.664801932703687: 1,
8.649749931429792: 1, 8.637679826559479: 1, 8.631062429782574: 1, 8.62968441818046: 1,
8.618311904589232: 1, 8.616623371629208: 1, 8.612624480762312: 1, 8.608954862590329: 1,
8.595771414328713: 1, 8.593839138052514: 1, 8.575381829100712: 1, 8.529814086165244: 1,
8.527852026740996: 1, 8.517504764710035: 1, 8.511313079147675: 1, 8.470937073781139: 1,
8.458682205376473: 1, 8.452734340969984: 1, 8.451195192894613: 1, 8.448022159562797: 1,
8.437387708628824: 1, 8.4091036536384: 1, 8.405707191982344: 1, 8.405130194299808: 1,
8.397172236877328: 1, 8.39694548832809: 1, 8.39104081369838: 1, 8.387405483735327: 1,
8.382984988087921: 1, 8.366171686647354: 1, 8.357727894120226: 1, 8.351836800373817: 1,
8.345788100620725: 1, 8.342502241789735: 1, 8.332833503927192: 1, 8.323481002173043: 1,
8.315304879387488: 1, 8.297494554465219: 1, 8.296458845490127: 1, 8.294435110983468: 1,
8.276953238394984: 1, 8.270298966703768: 1, 8.26714175980015: 1, 8.256168984281663: 1,
8.224626211565047: 1, 8.213961688920836: 1, 8.204447932510192: 1, 8.193668379230495: 1,
8.17252053206911: 1, 8.171986692195809: 1, 8.166788751023445: 1, 8.15966003105413: 1,
8.071631088555934: 1, 8.045048561235376: 1, 8.042993067723613: 1, 8.036471916840632: 1,
8.005109571963196: 1, 8.003852946995506: 1, 8.002069745392335: 1, 7.997718272295204: 1,
7.9887617412099114: 1, 7.959350924915069: 1, 7.958230252615605: 1, 7.95330818182801: 1,
7.937207120243148: 1, 7.928991561701733: 1, 7.92677925650088: 1, 7.916381259133647: 1,
7.914232267816127: 1, 7.913030010724033: 1, 7.910205797667173: 1, 7.909599170397424: 1,
7.900342917293533: 1, 7.893143692082361: 1, 7.8794744872347104: 1, 7.871562597264438: 1,
7.852330033300377: 1, 7.789284444767293: 1, 7.784819602308681: 1, 7.765075853056269: 1,
7.760314643757338: 1, 7.750055975882187: 1, 7.732667340235483: 1, 7.728672616597277: 1,
7.717111899290664: 1, 7.717107626728446: 1, 7.716242144135149: 1, 7.712914727684852: 1,
7.711211861965507: 1, 7.708255674367745: 1, 7.702541127766582: 1, 7.696796398528797: 1,
7.646431383968494: 1, 7.635901528796838: 1, 7.604226616452186: 1, 7.586354459819475: 1,
7.581403027258789: 1, 7.5726140171285055: 1, 7.541217268862663: 1, 7.539777820908305: 1,
7.515193115987626: 1, 7.491605365957293: 1, 7.4896997832251335: 1, 7.445711930242344: 1,
7.439477896128686: 1, 7.437358652231383: 1, 7.4339247392078835: 1, 7.4270979933369805: 1,
7.410231135369695: 1, 7.395620949556691: 1, 7.38865842441162: 1, 7.3332661159110195: 1,
7.305351492365063: 1, 7.292310950600754: 1, 7.277305870031149: 1, 7.276498824536766: 1,

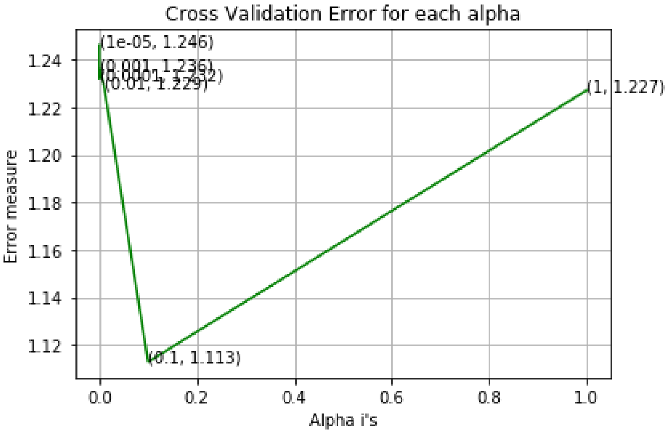
```
7.265786931354561: 1, 7.23582209878993: 1, 7.217007884510861: 1, 7.194417212736994: 1,
7.1460124231436435: 1, 7.117169030493916: 1, 7.106885141945008: 1, 7.0417728091733505: 1,
7.019475525067777: 1, 7.019355771398975: 1, 7.00332030558315: 1, 6.993315172863072: 1,
6.9857599244885655: 1, 6.954357571418443: 1, 6.9395539569022215: 1, 6.899450150951412: 1,
6.845268535546686: 1, 6.83832020360053: 1, 6.83792407897493: 1, 6.837437711601083: 1,
6.808947995702676: 1, 6.808319892413633: 1, 6.769321898006743: 1, 6.698586461619424: 1,
6.606720820739456: 1, 6.561621453856587: 1, 6.2486952447890065: 1})
```

In [70]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2461058880483369
For values of alpha =  0.0001 The log loss is: 1.2319675577759515
For values of alpha =  0.001 The log loss is: 1.2359114718607898
For values of alpha =  0.01 The log loss is: 1.2285638226710707
```

```
For values of alpha =  0.1 The log loss is: 1.1128496378707156
For values of alpha =  1 The log loss is: 1.227137601069132
```




```
For values of best alpha =  0.1 The train log loss is: 0.8231047298008423
For values of best alpha =  0.1 The cross validation log loss is: 1.1128496378707156
For values of best alpha =  0.1 The test log loss is: 1.1327366035274309
```

In [71]:

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

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

In [72]:

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

```
94.6 % of word of test data appeared in train data
93.6 % of word of Cross Validation appeared in train data
```

In [73]:

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

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

In [74]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = calibration.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 [75]:

```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(max_features=2000)

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

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

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

        else:


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


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

In [76]:

```
from scipy.sparse import hstack
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_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
r()
y_train = np.array(list(x_train['Class']))

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

x_cv_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
y_cv = np.array(list(x_cv['Class']))


train_gene_var_responseCoding = 
np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = 
np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = 
np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

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

In [77]:

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

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

In [78]:

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

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

Machine Learning Models

Naive Bayes

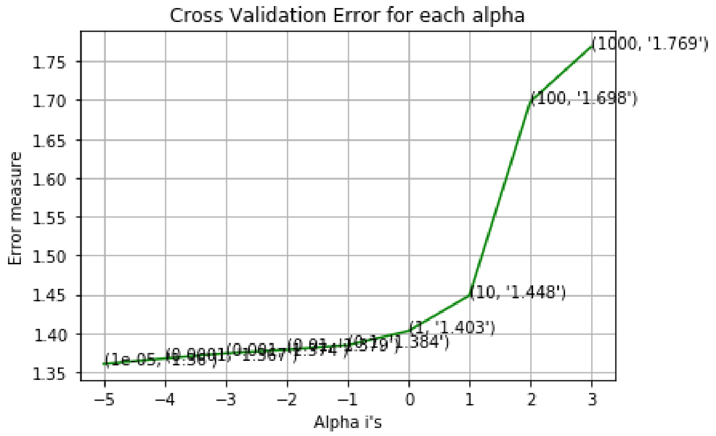
Hyper parameter tuning

In [79]:

```
from sklearn.naive_bayes import MultinomialNB
alpha = [0.00001, 0.0001, 0.001, 0.01,0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(x_train_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_onehotCoding, y_train)
    sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding)
    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_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_onehotCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_onehotCoding, y_train)
```

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

```
for alpha = 1e-05
Log Loss : 1.3603444761855885
for alpha = 0.0001
Log Loss : 1.3673116577507625
for alpha = 0.001
Log Loss : 1.3740123604882213
for alpha = 0.01
Log Loss : 1.3785934288689972
for alpha = 0.1
Log Loss : 1.3843362298835724
for alpha = 1
Log Loss : 1.4026060643781977
for alpha = 10
Log Loss : 1.4483904302815447
for alpha = 100
Log Loss : 1.6977869378620716
for alpha = 1000
Log Loss : 1.768602814728309
```



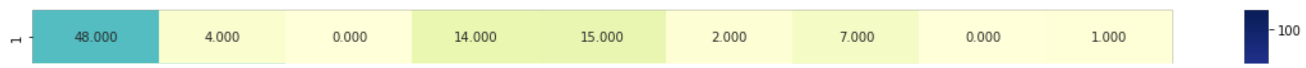

```
For values of best alpha =  1e-05 The train log loss is: 1.1551560341637221
For values of best alpha =  1e-05 The cross validation log loss is: 1.3603444761855885
For values of best alpha =  1e-05 The test log loss is: 1.336711106922002
```

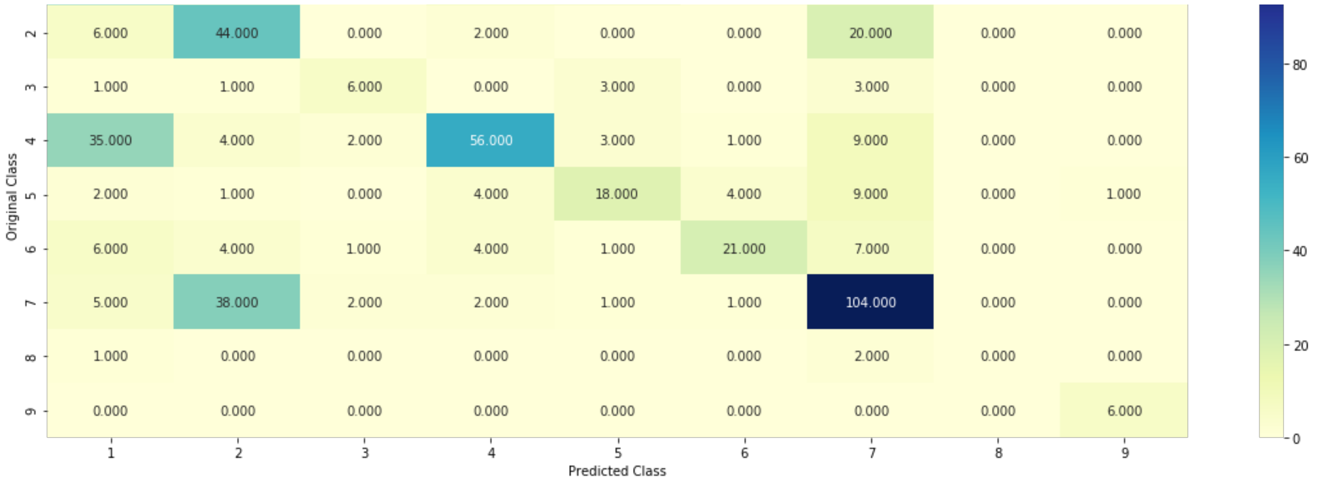
Testing the model with best hyper paramters

In [80]:

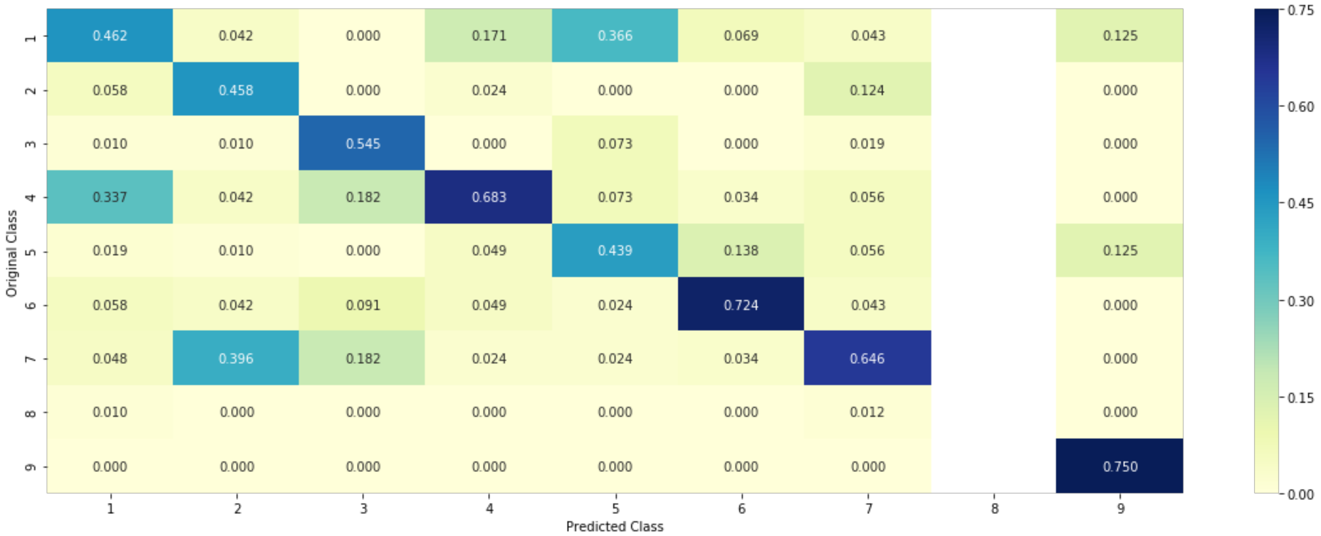
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(x_train_onehotCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_onehotCoding, y_train)
sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding)
# to avoid rounding error while multiplying probabilites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

```
Log Loss : 1.3603444761855885
Number of missclassified point : 0.43045112781954886
-------------------- Confusion matrix --------------------
```

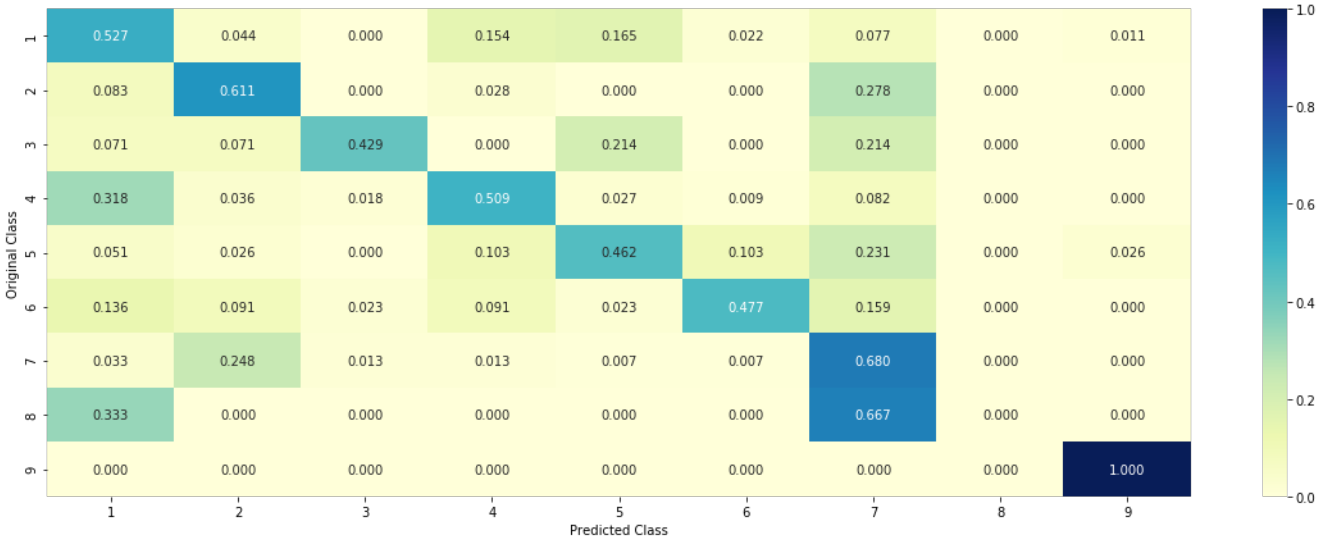





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



Feature Importance, Correctly classified point

In [81]:

```
test_point_index = 5
no_feature = 100
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls][:,:no_feature]

print("-"*50)
get_impfeature_names(indices[0], x_test.iloc[test_point_index]
['Text'],x_test.iloc[test_point_index]['Gene'],x_test.iloc[test_point_index]['Variation'], no_featu
re)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0943 0.0715 0.0164 0.1111 0.0406 0.0465 0.6106 0.0051 0.0038]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Feature Importance, Incorrectly classified point

In [82]:

```
test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_poi
nt_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0966 0.2177 0.0167 0.1132 0.0414 0.0477 0.4574 0.0053 0.004 ]]
Actual Class : 7
--------------------------------------------------
Out of the top  50  features  0 are present in query point
```

K Nearest Neighbour Classification

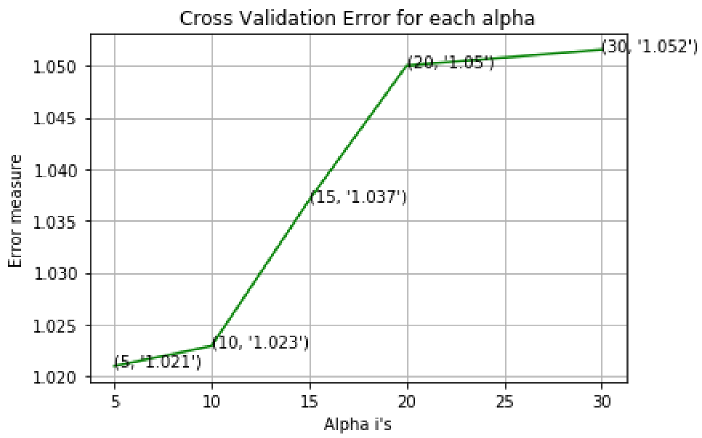
Hyper parameter tuning

In [83]:

```
from sklearn.neighbors import KNeighborsClassifier
alpha = [5, 10, 15, 20, 30]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, y_train)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    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(train_x_responseCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, y_train)

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

```
for alpha = 5
Log Loss : 1.020979220603989
for alpha = 10
Log Loss : 1.0229098351894523
for alpha = 15
Log Loss : 1.037000959269844
for alpha = 20
Log Loss : 1.0500077376297565
for alpha = 30
Log Loss : 1.0515556308216163
```



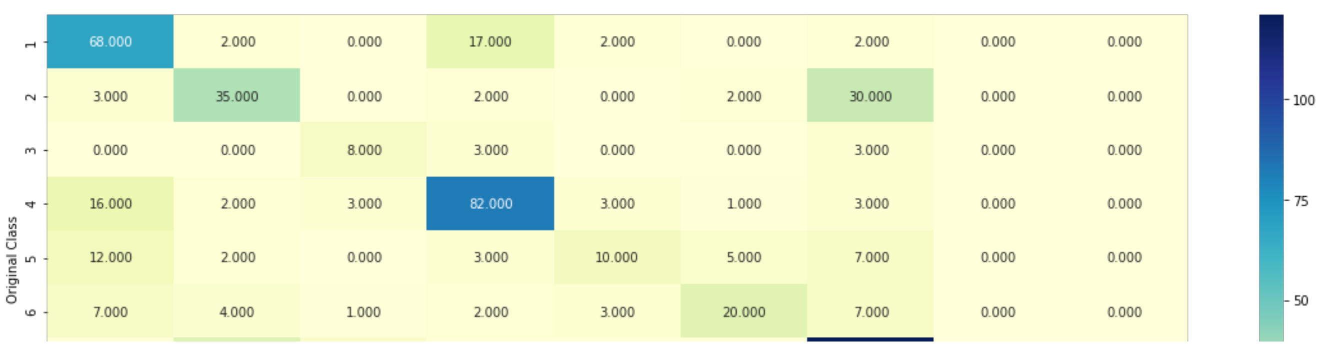

```
For values of best alpha =  5 The train log loss is: 0.4911720095683869
For values of best alpha =  5 The cross validation log loss is: 1.020979220603989
For values of best alpha =  5 The test log loss is: 1.0545683625931086
```

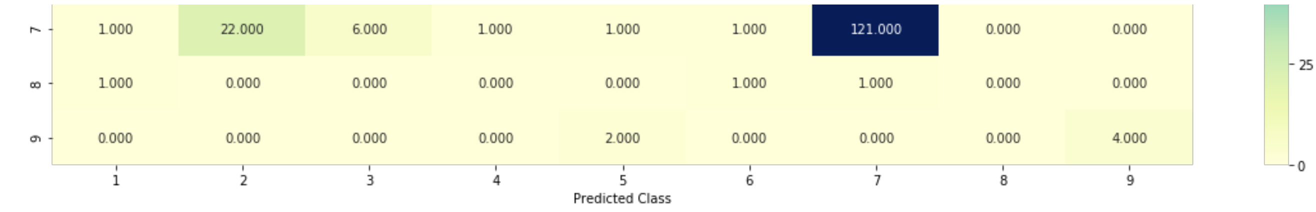
Testing the model with best hyper paramters

In [84]:

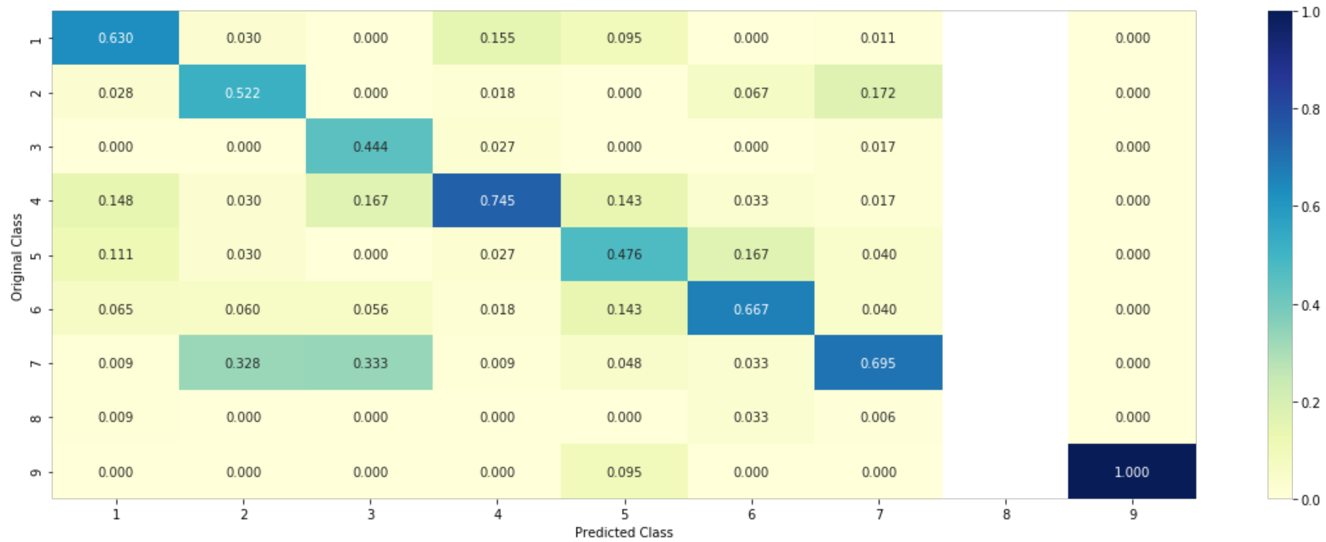
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, y_train, cv_x_responseCoding, y_cv, clf)
```

```
Log loss : 1.020979220603989
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```

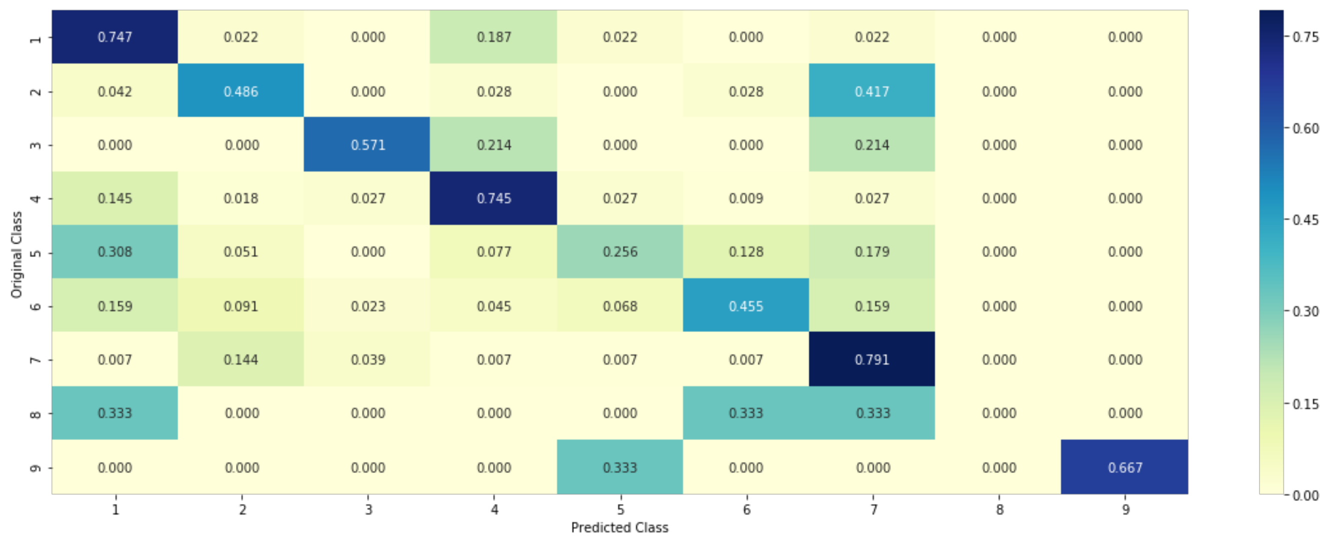





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



Sample Query point -1

In [85]:

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

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

```
print("Fequency of nearest points :",Counter(y_train[neighbors[1][0]]))
```

```
Predicted Class : 6
Actual Class : 7
The  5  nearest neighbours of the test points belongs to classes [2 7 7 7 2]
Fequency of nearest points : Counter({7: 3, 2: 2})
```

Sample Query point -2

In [86]:

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

test_point_index = 100

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

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

Logistic Regression

With Class balancing

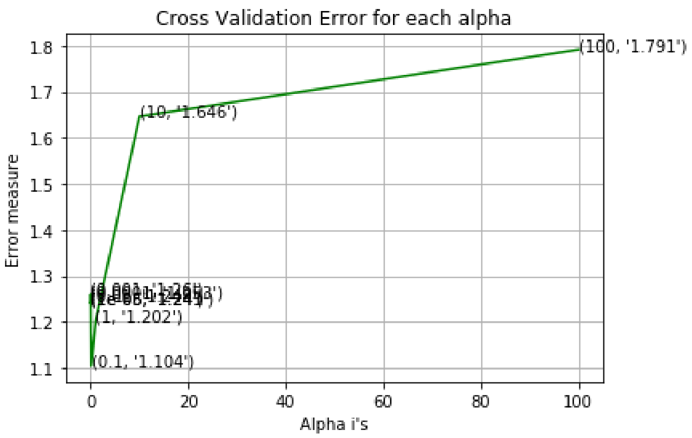
Hyper paramter tuning

In [87]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = linear_model.SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
ndom_state=42)
    clf.fit(x_train_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_onehotCoding, y_train)
    sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding)
    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 = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
loss='log', random_state=42)
```

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

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

```
for alpha = 1e-06
Log Loss : 1.2413875369115692
for alpha = 1e-05
Log Loss : 1.239546162646092
for alpha = 0.0001
Log Loss : 1.2531278688553258
for alpha = 0.001
Log Loss : 1.2604946883825392
for alpha = 0.01
Log Loss : 1.2489638826129315
for alpha = 0.1
Log Loss : 1.1042239998392998
for alpha = 1
Log Loss : 1.2024802309655005
for alpha = 10
Log Loss : 1.6462745477571243
for alpha = 100
Log Loss : 1.7913601760558078
```



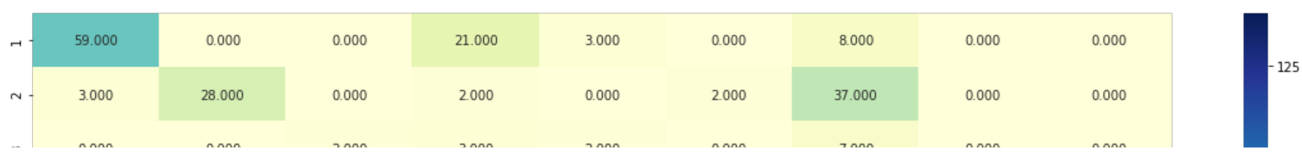

```
For values of best alpha =  0.1 The train log loss is: 0.8130161120131995
For values of best alpha =  0.1 The cross validation log loss is: 1.1042239998392998
For values of best alpha =  0.1 The test log loss is: 1.1140010957700772
```

Testing the model with best hyper paramters

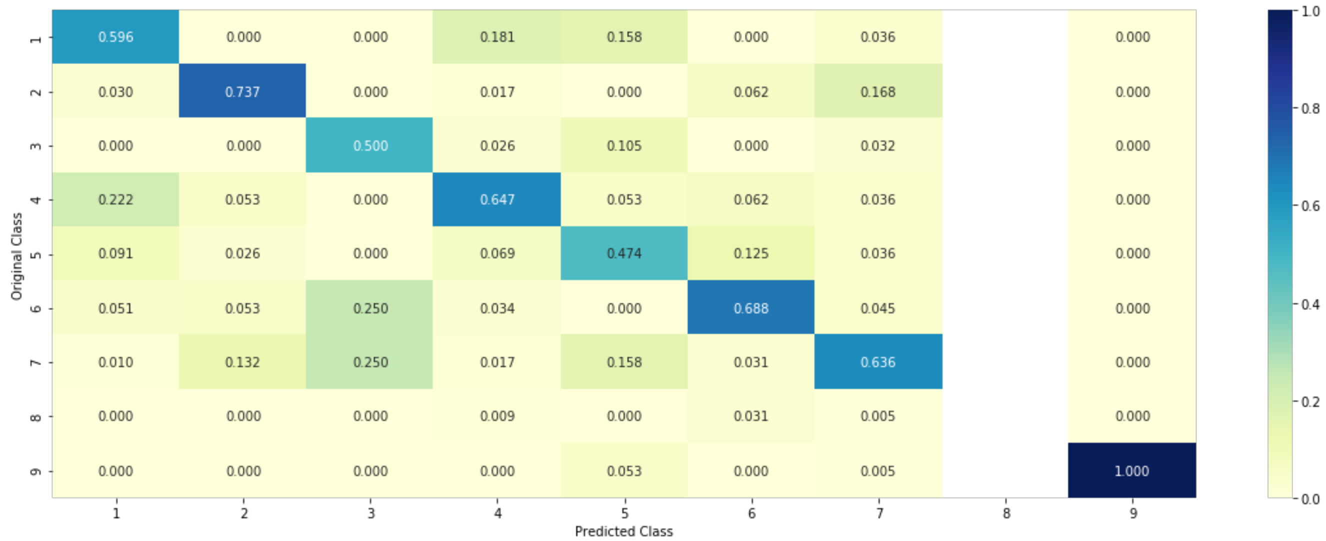
In [88]:

```
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(x_train_onehotCoding, y_train, x_cv_onehotCoding, y_cv, clf)
```

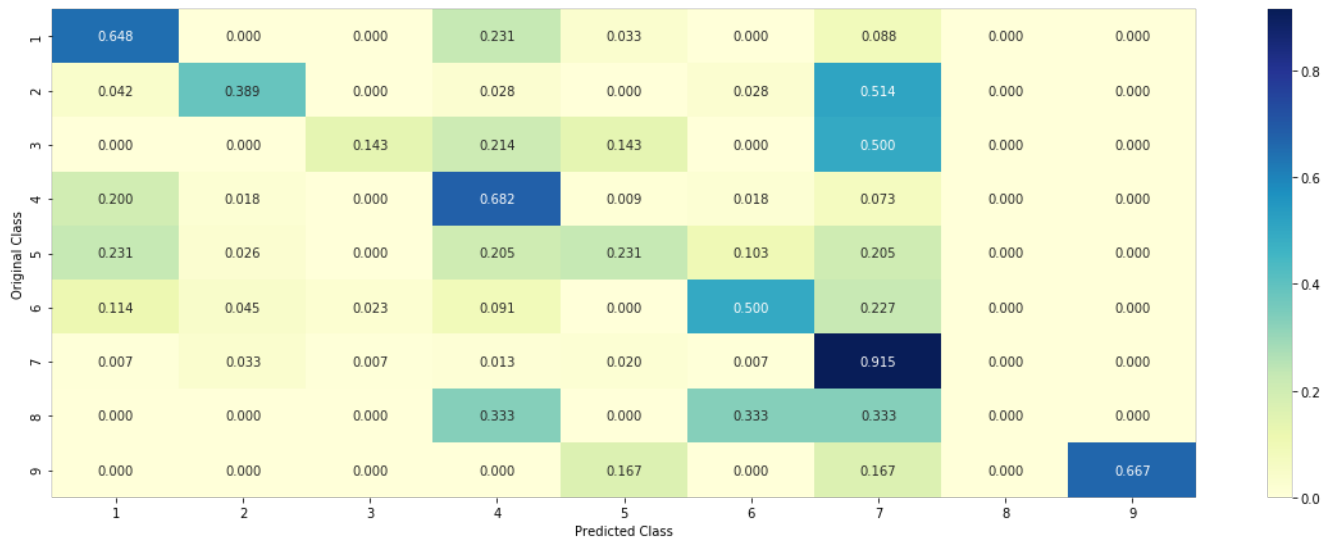
```
Log loss : 1.1042239998392998
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Incorrectly classified points

In [89]:

```
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
loss='log', random_state=42)
clf.fit(x_train_onehotCoding,y_train)
test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_poi
nt_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.028  0.2928 0.0078 0.0394 0.0257 0.0688 0.5311 0.0051 0.0014]]
Actual Class : 7
--------------------------------------------------
0 Text feature [free] present in test data point [True]
1 Text feature [affinity] present in test data point [True]
2 Text feature [50] present in test data point [True]
3 Text feature [affect] present in test data point [True]
4 Text feature [26] present in test data point [True]
5 Text feature [57] present in test data point [True]
7 Text feature [enzyme] present in test data point [True]
8 Text feature [identified] present in test data point [True]
9 Text feature [4c] present in test data point [True]
14 Text feature [greater] present in test data point [True]
16 Text feature [enriched] present in test data point [True]
18 Text feature [ability] present in test data point [True]
22 Text feature [group] present in test data point [True]
23 Text feature [association] present in test data point [True]
24 Text feature [codons] present in test data point [True]
25 Text feature [73] present in test data point [True]
30 Text feature [corresponding] present in test data point [True]
32 Text feature [59] present in test data point [True]
33 Text feature [alter] present in test data point [True]
36 Text feature [conserved] present in test data point [True]
37 Text feature [3d] present in test data point [True]
38 Text feature [demonstrating] present in test data point [True]
40 Text feature [encoded] present in test data point [True]
42 Text feature [24] present in test data point [True]
43 Text feature [highly] present in test data point [True]
44 Text feature [containing] present in test data point [True]
45 Text feature [increase] present in test data point [True]
46 Text feature [despite] present in test data point [True]
48 Text feature [effects] present in test data point [True]
Out of the top  50  features  29 are present in query point
```

Correctly classified Point

In [90]:

```
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
loss='log', random_state=42)
clf.fit(x_train_onehotCoding,y_train)
test_point_index = 4
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_poi
nt_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.017  0.137  0.0341 0.0557 0.0688 0.0399 0.6369 0.0061 0.0045]]
Actual Class : 7
--------------------------------------------------
0 Text feature [free] present in test data point [True]
1 Text feature [affinity] present in test data point [True]
2 Text feature [50] present in test data point [True]
3 Text feature [affect] present in test data point [True]
4 Text feature [26] present in test data point [True]
5 Text feature [57] present in test data point [True]
6 Text feature [classes] present in test data point [True]
```

```
7 Text feature [enzyme] present in test data point [True]
8 Text feature [identified] present in test data point [True]
9 Text feature [4c] present in test data point [True]
10 Text feature [blot] present in test data point [True]
13 Text feature [discovery] present in test data point [True]
14 Text feature [greater] present in test data point [True]
16 Text feature [enriched] present in test data point [True]
18 Text feature [ability] present in test data point [True]
20 Text feature [anti] present in test data point [True]
22 Text feature [group] present in test data point [True]
23 Text feature [association] present in test data point [True]
24 Text feature [codons] present in test data point [True]
25 Text feature [73] present in test data point [True]
27 Text feature [already] present in test data point [True]
29 Text feature [grown] present in test data point [True]
30 Text feature [corresponding] present in test data point [True]
32 Text feature [59] present in test data point [True]
33 Text feature [alter] present in test data point [True]
34 Text feature [conservation] present in test data point [True]
35 Text feature [inhibits] present in test data point [True]
36 Text feature [conserved] present in test data point [True]
37 Text feature [3d] present in test data point [True]
38 Text feature [demonstrating] present in test data point [True]
39 Text feature [induction] present in test data point [True]
40 Text feature [encoded] present in test data point [True]
41 Text feature [green] present in test data point [True]
42 Text feature [24] present in test data point [True]
43 Text feature [highly] present in test data point [True]
44 Text feature [containing] present in test data point [True]
45 Text feature [increase] present in test data point [True]
46 Text feature [despite] present in test data point [True]
47 Text feature [250] present in test data point [True]
48 Text feature [effects] present in test data point [True]
Out of the top  50  features  40 are present in query point
```

Without Class balancing

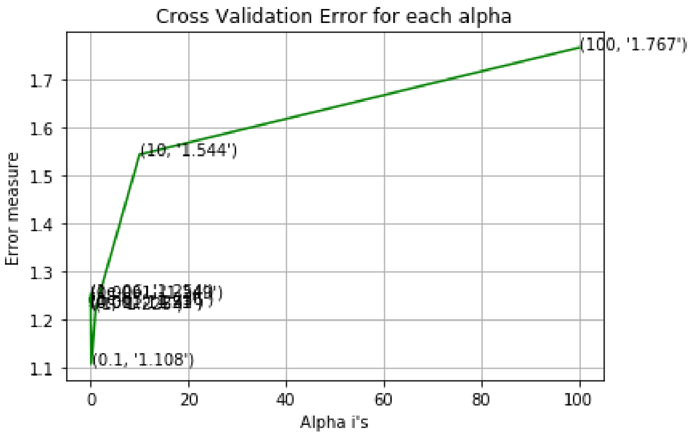
Hyper paramter tuning

In [91]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = linear_model.SGDClassifier( alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(x_train_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_onehotCoding, y_train)
    sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding)
    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 = linear_model.SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
2)
clf.fit(x_train_onehotCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(x_train_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_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_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(x_test_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.253858170118555
for alpha = 1e-05
Log Loss : 1.2362390450743816
for alpha = 0.0001
Log Loss : 1.2485005701885803
for alpha = 0.001
Log Loss : 1.2296760085406928
for alpha = 0.01
Log Loss : 1.2306056039910478
for alpha = 0.1
Log Loss : 1.1084239830057898
for alpha = 1
Log Loss : 1.2252765926738387
for alpha = 10
Log Loss : 1.544199720610038
for alpha = 100
Log Loss : 1.7665027674217917
```



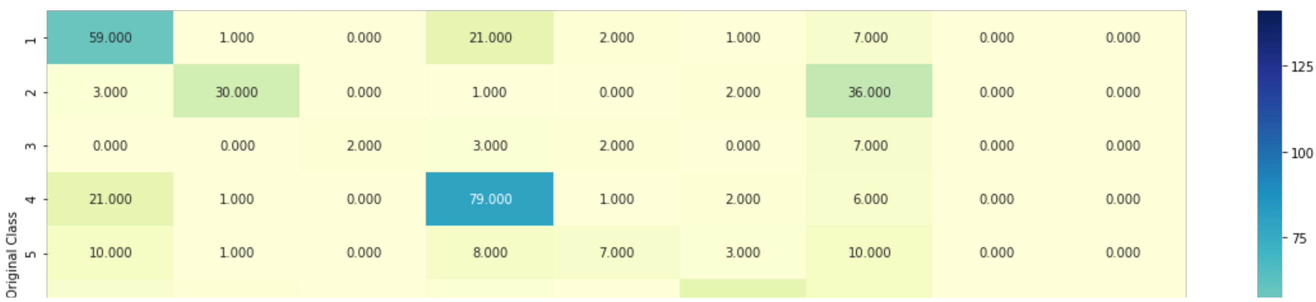

```
For values of best alpha =  0.1 The train log loss is: 0.8174854460389374
For values of best alpha =  0.1 The cross validation log loss is: 1.1084239830057898
For values of best alpha =  0.1 The test log loss is: 1.1269479773519668
```

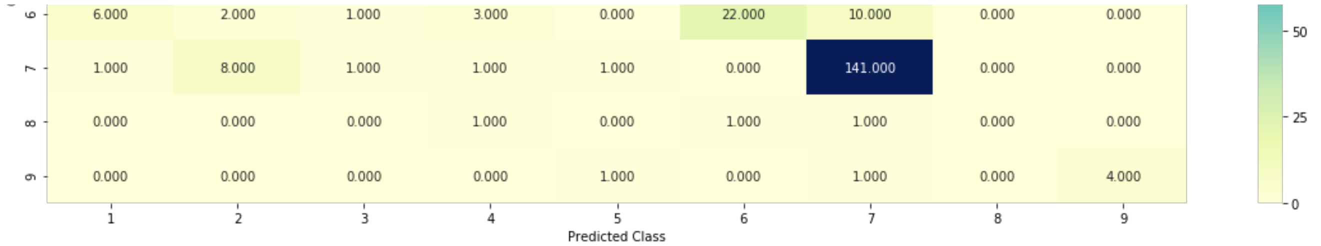
Testing model with best hyper parameters

In [92]:

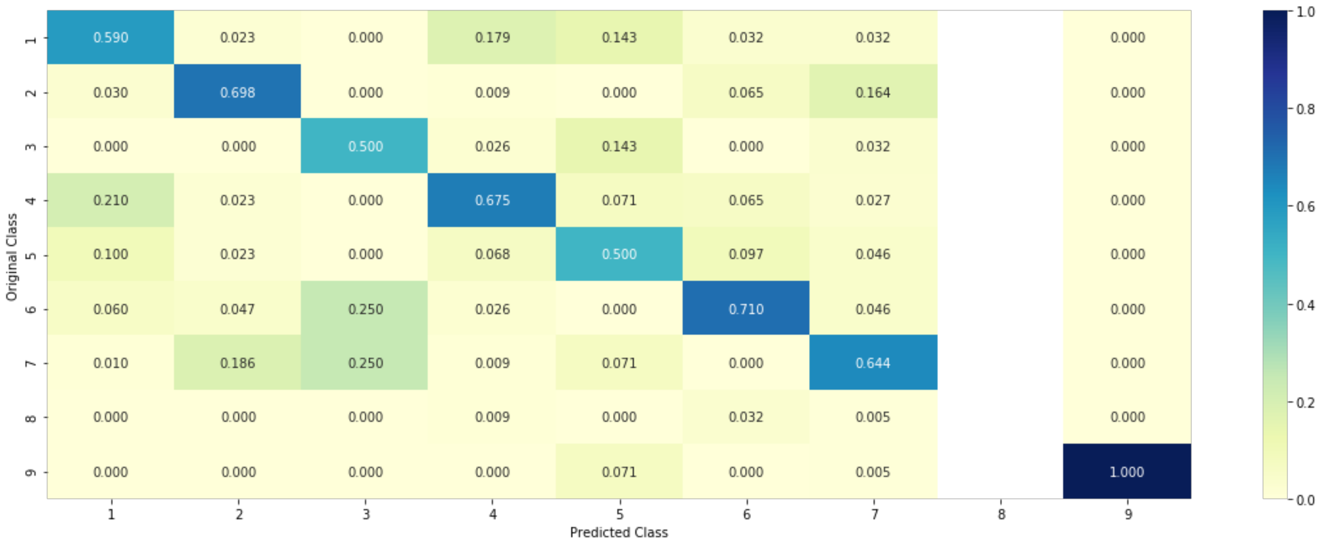
```
clf = linear_model.SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
2)
predict_and_plot_confusion_matrix(x_train_onehotCoding, y_train, x_cv_onehotCoding, y_cv, clf)
```

```
Log loss : 1.1084239830057898
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```

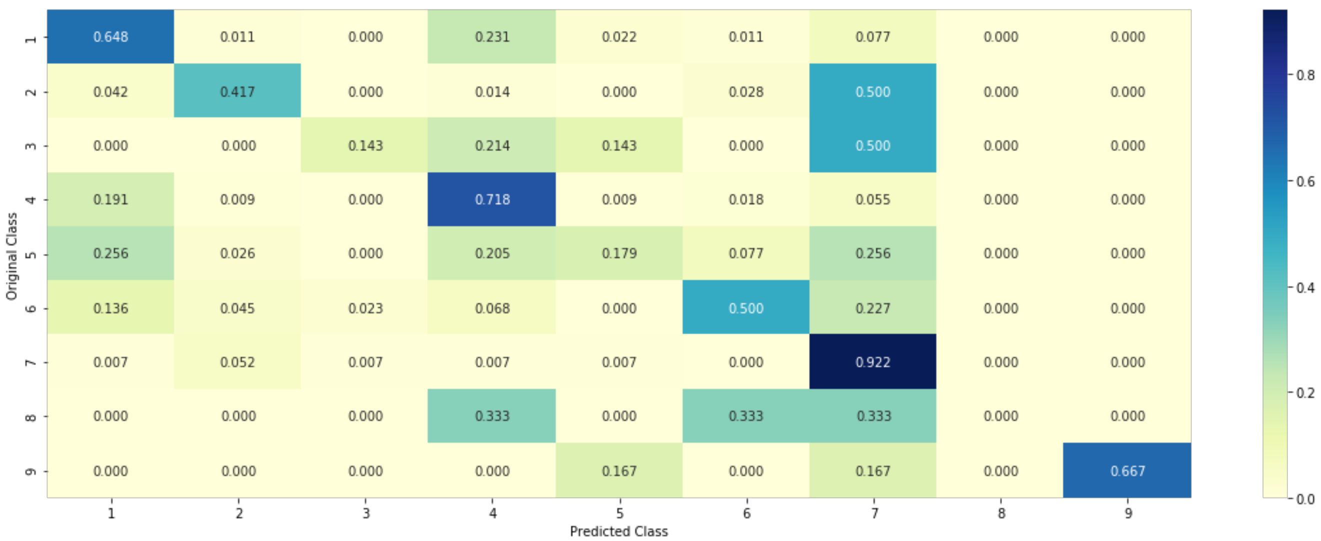





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



Feature Importance, Incorrectly Classified point

In [93]:

```
clf = linear_model.SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
2)
clf.fit(x_train_onehotCoding,y_train)
test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_point_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0305 0.2977 0.0071 0.0336 0.0243 0.0619 0.5381 0.0056 0.0013]]
Actual Class : 7
--------------------------------------------------
0 Text feature [free] present in test data point [True]
1 Text feature [57] present in test data point [True]
2 Text feature [26] present in test data point [True]
4 Text feature [effects] present in test data point [True]
5 Text feature [50] present in test data point [True]
6 Text feature [affinity] present in test data point [True]
9 Text feature [4c] present in test data point [True]
10 Text feature [affect] present in test data point [True]
12 Text feature [73] present in test data point [True]
13 Text feature [enzyme] present in test data point [True]
14 Text feature [constructs] present in test data point [True]
15 Text feature [identified] present in test data point [True]
19 Text feature [codons] present in test data point [True]
24 Text feature [greater] present in test data point [True]
28 Text feature [containing] present in test data point [True]
29 Text feature [89] present in test data point [True]
30 Text feature [enriched] present in test data point [True]
31 Text feature [104] present in test data point [True]
32 Text feature [association] present in test data point [True]
34 Text feature [inhibited] present in test data point [True]
36 Text feature [ability] present in test data point [True]
37 Text feature [experiment] present in test data point [True]
39 Text feature [evaluation] present in test data point [True]
40 Text feature [inhibition] present in test data point [True]
41 Text feature [90] present in test data point [True]
45 Text feature [corresponding] present in test data point [True]
46 Text feature [24] present in test data point [True]
47 Text feature [59] present in test data point [True]
48 Text feature [buffer] present in test data point [True]
49 Text feature [alter] present in test data point [True]
Out of the top  50  features  30 are present in query point
```

Feature Importance, Correctly Classified point

In [94]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0179 0.1353 0.0309 0.0574 0.0621 0.0407 0.6464 0.0054 0.0037]]
Actual Class : 7
--------------------------------------------------
0 Text feature [free] present in test data point [True]
1 Text feature [57] present in test data point [True]
2 Text feature [26] present in test data point [True]
4 Text feature [effects] present in test data point [True]
5 Text feature [50] present in test data point [True]
6 Text feature [affinity] present in test data point [True]
7 Text feature [blot] present in test data point [True]
8 Text feature [classes] present in test data point [True]
9 Text feature [4c] present in test data point [True]
10 Text feature [affect] present in test data point [True]
12 Text feature [73] present in test data point [True]
13 Text feature [enzyme] present in test data point [True]
```

```
14 Text feature [constructs] present in test data point [True]
15 Text feature [identified] present in test data point [True]
17 Text feature [discovery] present in test data point [True]
19 Text feature [codons] present in test data point [True]
22 Text feature [250] present in test data point [True]
24 Text feature [greater] present in test data point [True]
25 Text feature [anti] present in test data point [True]
27 Text feature [already] present in test data point [True]
28 Text feature [containing] present in test data point [True]
29 Text feature [89] present in test data point [True]
30 Text feature [enriched] present in test data point [True]
32 Text feature [association] present in test data point [True]
33 Text feature [conservation] present in test data point [True]
34 Text feature [inhibited] present in test data point [True]
36 Text feature [ability] present in test data point [True]
37 Text feature [experiment] present in test data point [True]
38 Text feature [inhibits] present in test data point [True]
39 Text feature [evaluation] present in test data point [True]
40 Text feature [inhibition] present in test data point [True]
41 Text feature [90] present in test data point [True]
43 Text feature [bottom] present in test data point [True]
45 Text feature [corresponding] present in test data point [True]
46 Text feature [24] present in test data point [True]
47 Text feature [59] present in test data point [True]
49 Text feature [alter] present in test data point [True]
Out of the top  50  features  37 are present in query point
```

Linear Support Vector Machines

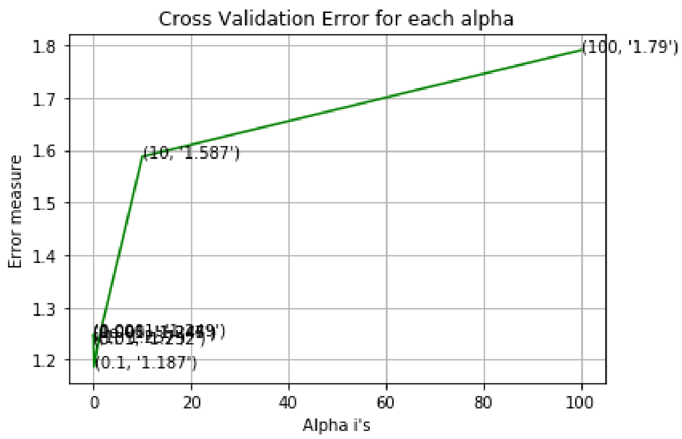
Testing model with best hyper parameters

In [95]:

```
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 = linear_model.SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge',
random_state=42)
    clf.fit(x_train_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_onehotCoding, y_train)
    sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding)
    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='balanced')
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
loss='hinge', random_state=42)
clf.fit(x_train_onehotCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(x_train_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(x_cv_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(x_test_onehotCoding)
```

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

```
for C = 1e-05
Log Loss : 1.2449966489595787
for C = 0.0001
Log Loss : 1.2485168680969072
for C = 0.001
Log Loss : 1.246575837204197
for C = 0.01
Log Loss : 1.2321343436154815
for C = 0.1
Log Loss : 1.1865120101068545
for C = 1
Log Loss : 1.2366232548780012
for C = 10
Log Loss : 1.5874271074144328
for C = 100
Log Loss : 1.7901665946149337
```

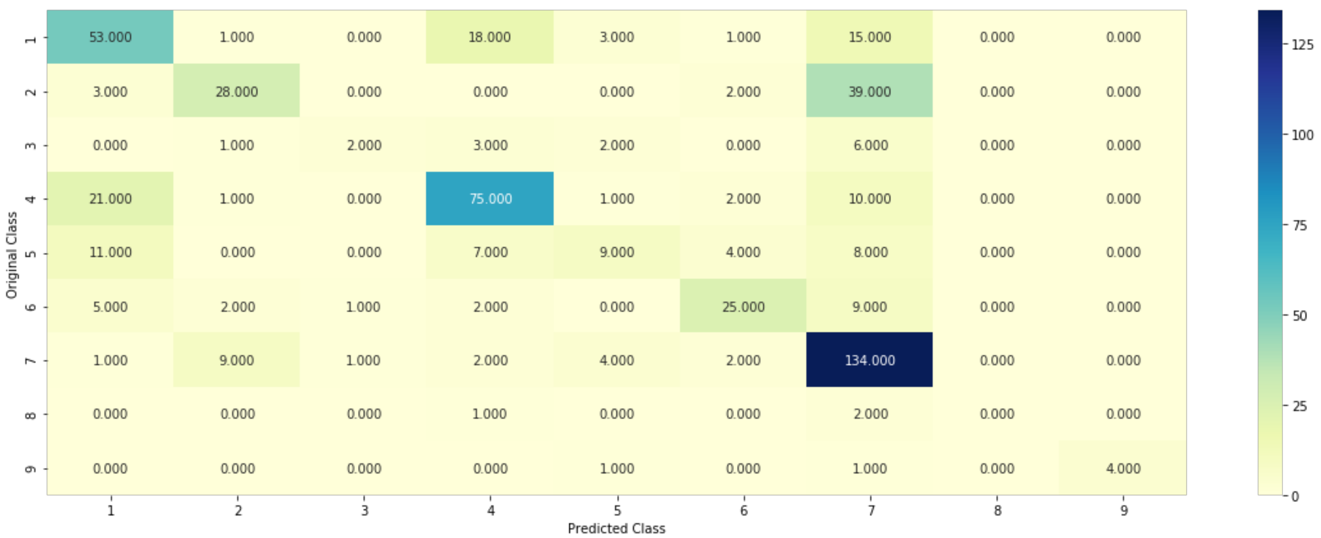



```
For values of best alpha =  0.1 The train log loss is: 0.8975234959812588
For values of best alpha =  0.1 The cross validation log loss is: 1.1865120101068545
For values of best alpha =  0.1 The test log loss is: 1.2028781335907444
```

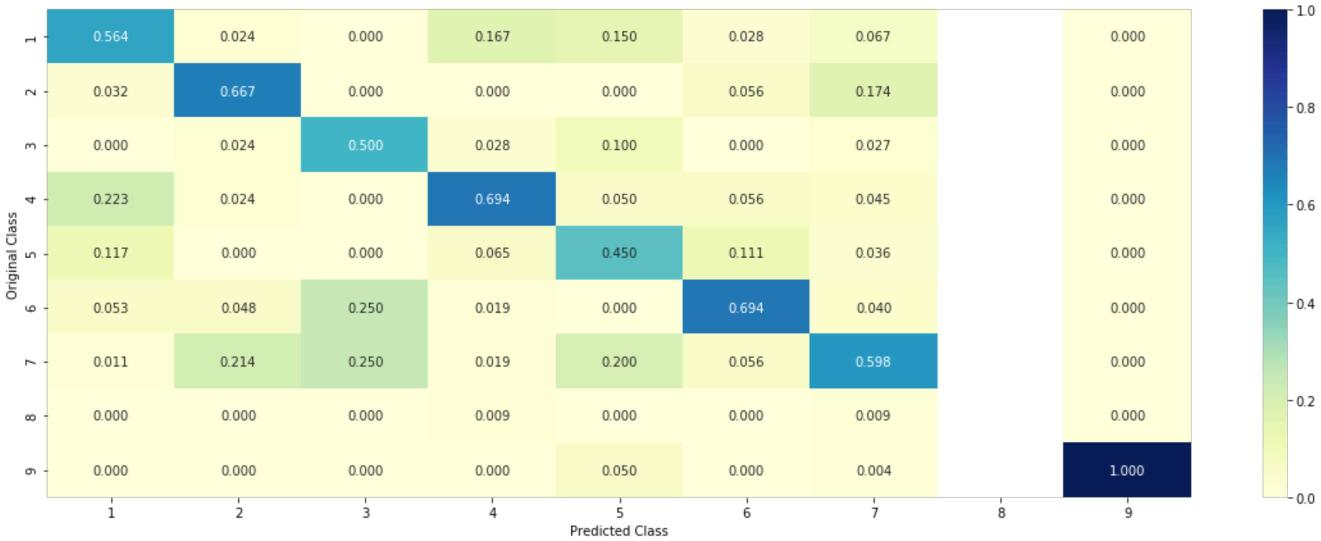
In [96]:

```
clf = linear_model.SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
=42,class_weight='balanced')
predict_and_plot_confusion_matrix(x_train_onehotCoding, y_train,x_cv_onehotCoding,y_cv, clf)
```

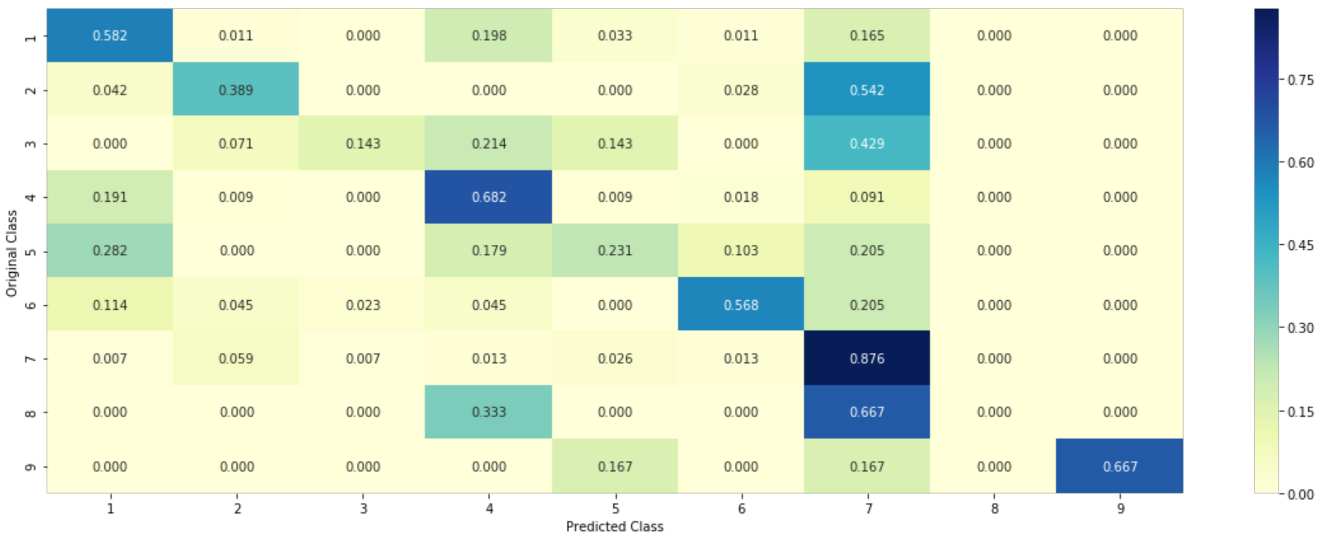
```
Log loss : 1.1865120101068545
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```



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




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



Testing model with best hyper parameters

For Incorrectly classified point

In [97]:

```
clf = linear_model.SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
=42)
clf.fit(x_train_onehotCoding,y_train)
test_point_index = 1
# test_point_index = 100
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_poi
nt_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0321 0.2526 0.0101 0.0415 0.039  0.0934 0.5242 0.0059 0.0011]]
Actual Class : 7
```

```
--------------------------------------------------
0 Text feature [effects] present in test data point [True]
1 Text feature [26] present in test data point [True]
2 Text feature [free] present in test data point [True]
4 Text feature [57] present in test data point [True]
5 Text feature [73] present in test data point [True]
6 Text feature [constructs] present in test data point [True]
8 Text feature [braf] present in test data point [True]
9 Text feature [evaluation] present in test data point [True]
11 Text feature [50] present in test data point [True]
12 Text feature [enzyme] present in test data point [True]
13 Text feature [89] present in test data point [True]
15 Text feature [biochemical] present in test data point [True]
17 Text feature [experiment] present in test data point [True]
19 Text feature [4c] present in test data point [True]
24 Text feature [aberrations] present in test data point [True]
25 Text feature [enriched] present in test data point [True]
26 Text feature [identified] present in test data point [True]
29 Text feature [buffer] present in test data point [True]
30 Text feature [contribute] present in test data point [True]
31 Text feature [90] present in test data point [True]
34 Text feature [epidermal] present in test data point [True]
36 Text feature [double] present in test data point [True]
37 Text feature [focus] present in test data point [True]
38 Text feature [affinity] present in test data point [True]
39 Text feature [corresponding] present in test data point [True]
40 Text feature [containing] present in test data point [True]
43 Text feature [encoded] present in test data point [True]
45 Text feature [every] present in test data point [True]
46 Text feature [culture] present in test data point [True]
47 Text feature [completely] present in test data point [True]
48 Text feature [highly] present in test data point [True]
49 Text feature [59] present in test data point [True]
Out of the top  50  features  32 are present in query point
```

For Correctly classified point

In [98]:

```
clf = linear_model.SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
=42)
clf.fit(x_train_onehotCoding,y_train)
test_point_index = 4
# test_point_index = 100
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_poi
nt_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.033  0.1167 0.0315 0.0989 0.0715 0.0504 0.5885 0.0056 0.004 ]]
Actual Class : 7
--------------------------------------------------
0 Text feature [effects] present in test data point [True]
1 Text feature [26] present in test data point [True]
2 Text feature [free] present in test data point [True]
4 Text feature [57] present in test data point [True]
5 Text feature [73] present in test data point [True]
6 Text feature [constructs] present in test data point [True]
8 Text feature [braf] present in test data point [True]
9 Text feature [evaluation] present in test data point [True]
11 Text feature [50] present in test data point [True]
12 Text feature [enzyme] present in test data point [True]
13 Text feature [89] present in test data point [True]
14 Text feature [induction] present in test data point [True]
17 Text feature [experiment] present in test data point [True]
18 Text feature [250] present in test data point [True]
19 Text feature [4c] present in test data point [True]
```

```
20 Text feature [blot] present in test data point [True]
22 Text feature [classes] present in test data point [True]
25 Text feature [enriched] present in test data point [True]
26 Text feature [identified] present in test data point [True]
27 Text feature [analysed] present in test data point [True]
28 Text feature [erk2] present in test data point [True]
30 Text feature [contribute] present in test data point [True]
31 Text feature [90] present in test data point [True]
33 Text feature [anti] present in test data point [True]
36 Text feature [double] present in test data point [True]
37 Text feature [focus] present in test data point [True]
38 Text feature [affinity] present in test data point [True]
39 Text feature [corresponding] present in test data point [True]
40 Text feature [containing] present in test data point [True]
43 Text feature [encoded] present in test data point [True]
44 Text feature [0001] present in test data point [True]
45 Text feature [every] present in test data point [True]
46 Text feature [culture] present in test data point [True]
47 Text feature [completely] present in test data point [True]
48 Text feature [highly] present in test data point [True]
49 Text feature [59] present in test data point [True]
Out of the top  50  features  36 are present in query point
```

Random Forest Classifier

Hyper paramter tuning (With One hot Encoding)

In [99]:

```
from sklearn.ensemble import RandomForestClassifier
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_onehotCoding, y_train)
        sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(x_train_onehotCoding, y_train)
        sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding)
        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]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

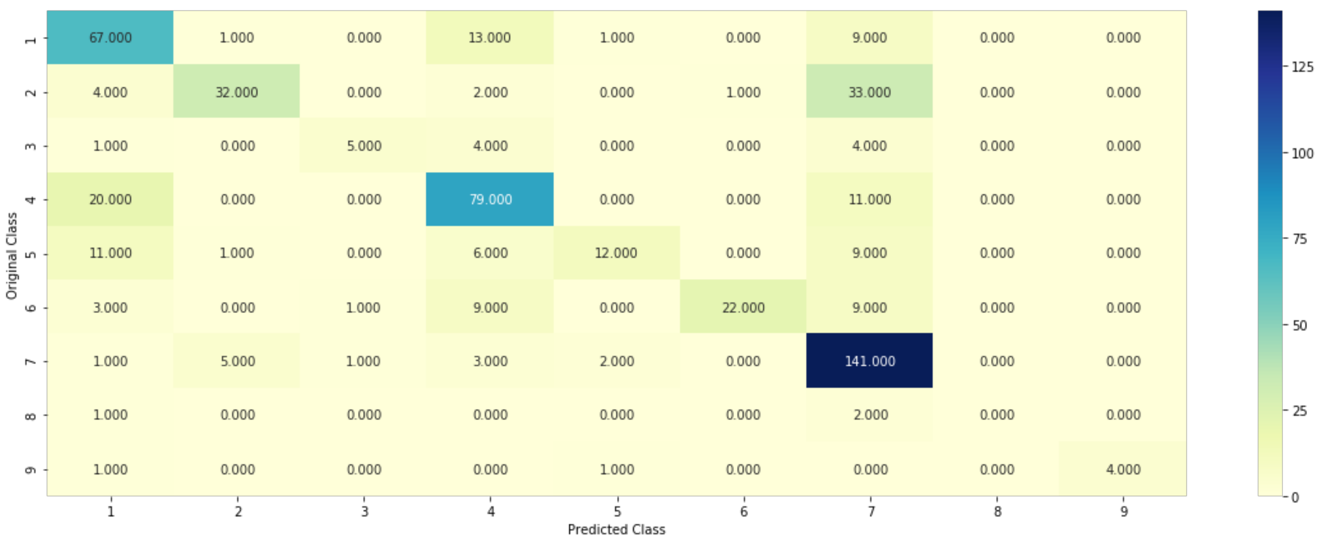
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.133515257607224
for n_estimators = 100 and max depth =  10
Log Loss : 1.0910229242363092
for n_estimators = 200 and max depth =  5
Log Loss : 1.1191672923217484
for n_estimators = 200 and max depth =  10
Log Loss : 1.0815664480088312
for n_estimators = 500 and max depth =  5
Log Loss : 1.113722805765445
for n_estimators = 500 and max depth =  10
Log Loss : 1.0808078636282192
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1135618722467393
for n_estimators = 1000 and max depth =  10
Log Loss : 1.078971826505626
for n_estimators = 2000 and max depth =  5
Log Loss : 1.111408651919868
for n_estimators = 2000 and max depth =  10
Log Loss : 1.0788958334409477
For values of best estimator =  2000 The train log loss is: 0.5508938852591605
For values of best estimator =  2000 The cross validation log loss is: 1.0788958334409477
For values of best estimator =  2000 The test log loss is: 1.0787332466652166
```

Testing model with best hyper parameters (One Hot Encoding)

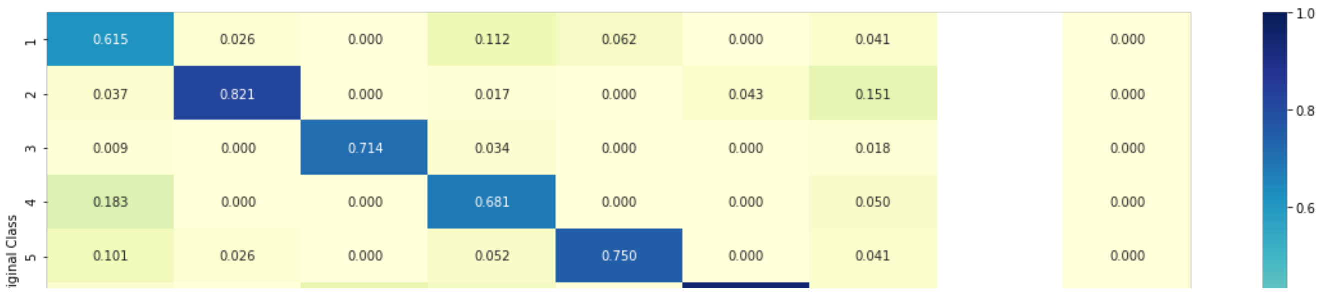
In [100]:

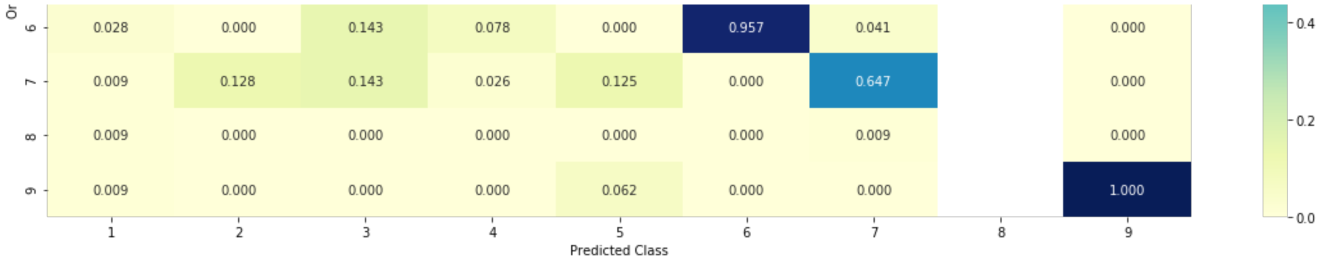
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(x_train_onehotCoding, y_train,x_cv_onehotCoding,y_cv, clf)
```

```
Log loss : 1.0788958334409477
Number of mis-classified points : 0.31954887218045114
-------------------- Confusion matrix --------------------
```

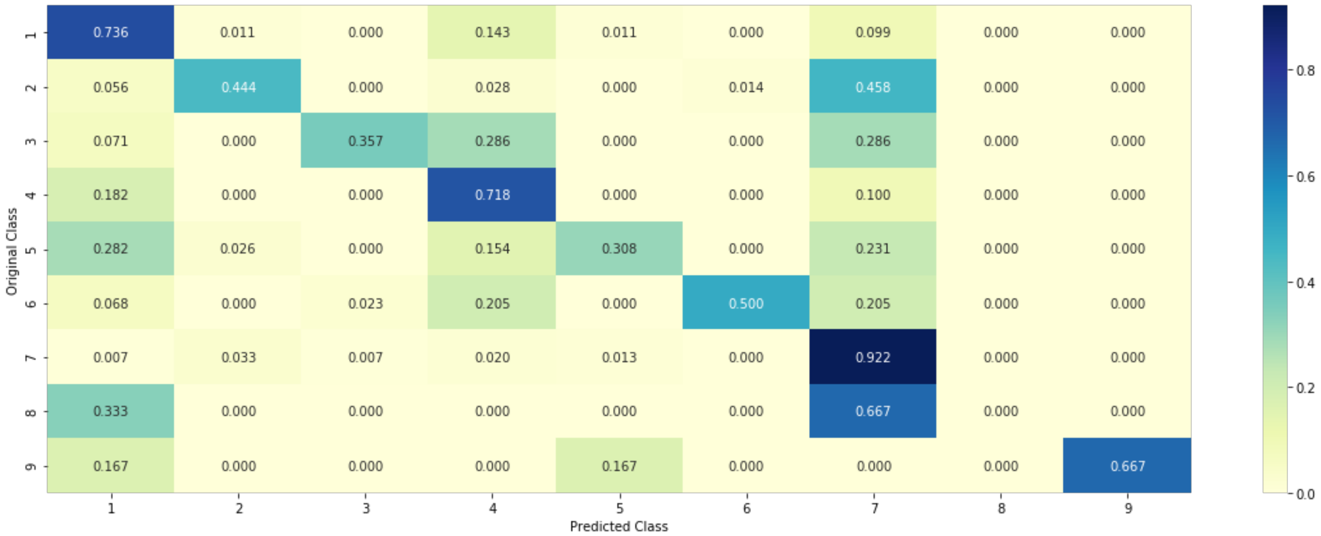



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



Feature Importance

Incorrectly Classified point

In [101]:

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

test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[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],
x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_point_index],x_test['Variation'].ilo
c[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0889 0.1774 0.0187 0.0901 0.0462 0.0403 0.5253 0.0057 0.0073]]
Actual Class : 7
--------------------------------------------------
2 Text feature [50] present in test data point [True]
5 Text feature [4c] present in test data point [True]
6 Text feature [evaluate] present in test data point [True]
7 Text feature [indicate] present in test data point [True]
8 Text feature [correlated] present in test data point [True]
9 Text feature [correlate] present in test data point [True]
10 Text feature [included] present in test data point [True]
11 Text feature [demonstrating] present in test data point [True]
```

```
14 Text feature [colorectal] present in test data point [True]
15 Text feature [activating] present in test data point [True]
17 Text feature [encodes] present in test data point [True]
19 Text feature [influence] present in test data point [True]
20 Text feature [available] present in test data point [True]
25 Text feature [classical] present in test data point [True]
27 Text feature [identify] present in test data point [True]
29 Text feature [antibodies] present in test data point [True]
30 Text feature [distinct] present in test data point [True]
35 Text feature [cause] present in test data point [True]
36 Text feature [calculated] present in test data point [True]
38 Text feature [60] present in test data point [True]
39 Text feature [association] present in test data point [True]
40 Text feature [almost] present in test data point [True]
41 Text feature [conditions] present in test data point [True]
43 Text feature [driven] present in test data point [True]
44 Text feature [include] present in test data point [True]
47 Text feature [first] present in test data point [True]
49 Text feature [allowed] present in test data point [True]
Out of the top  50  features  27 are present in query point
```

Correctly Classified point

In [102]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(x_train_onehotCoding, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_onehotCoding, y_train)

test_point_index = 4
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(x_test_onehotCoding[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],
x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_point_index],x_test['Variation'].ilo
c[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0391 0.0857 0.0149 0.0431 0.036  0.0262 0.7465 0.0038 0.0047]]
Actual Class : 7
--------------------------------------------------
0 Text feature [current] present in test data point [True]
1 Text feature [4d] present in test data point [True]
2 Text feature [50] present in test data point [True]
3 Text feature [classes] present in test data point [True]
5 Text feature [4c] present in test data point [True]
6 Text feature [evaluate] present in test data point [True]
7 Text feature [indicate] present in test data point [True]
8 Text feature [correlated] present in test data point [True]
10 Text feature [included] present in test data point [True]
11 Text feature [demonstrating] present in test data point [True]
12 Text feature [discovery] present in test data point [True]
13 Text feature [anti] present in test data point [True]
14 Text feature [colorectal] present in test data point [True]
15 Text feature [activating] present in test data point [True]
16 Text feature [genetics] present in test data point [True]
17 Text feature [encodes] present in test data point [True]
18 Text feature [feature] present in test data point [True]
19 Text feature [influence] present in test data point [True]
20 Text feature [available] present in test data point [True]
21 Text feature [extracellular] present in test data point [True]
23 Text feature [harbored] present in test data point [True]
25 Text feature [classical] present in test data point [True]
27 Text feature [identify] present in test data point [True]
29 Text feature [antibodies] present in test data point [True]
30 Text feature [distinct] present in test data point [True]
31 Text feature [agents] present in test data point [True]
```

```
35 Text feature [cause] present in test data point [True]
36 Text feature [calculated] present in test data point [True]
38 Text feature [60] present in test data point [True]
39 Text feature [association] present in test data point [True]
40 Text feature [almost] present in test data point [True]
41 Text feature [conditions] present in test data point [True]
43 Text feature [driven] present in test data point [True]
44 Text feature [include] present in test data point [True]
47 Text feature [first] present in test data point [True]
Out of the top  50  features  35 are present in query point
```

Hyper paramter tuning (With Response Coding)

In [103]:

```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_responseCoding, y_train)
        sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, y_train)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        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]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.12233831617013
for n_estimators = 10 and max depth =  3
Log Loss : 1.6487611779488807
for n_estimators = 10 and max depth =  5
Log Loss : 1.527378338663594
for n_estimators = 10 and max depth =  10
Log Loss : 2.009550240601299
for n_estimators = 50 and max depth =  2
Log Loss : 1.6576706443030875
for n_estimators = 50 and max depth =  3
Log Loss : 1.4525796724767268
for n_estimators = 50 and max depth =  5
Log Loss : 1.2994414457160217
```

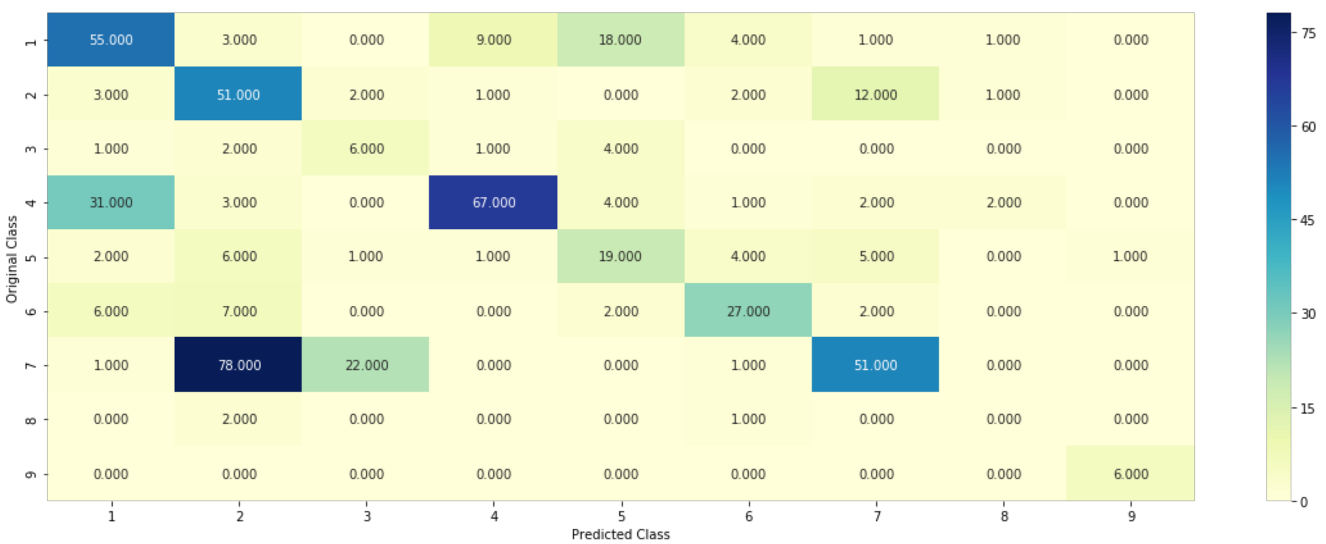
```
for n_estimators = 50 and max depth =  10
Log Loss : 1.7894587325973765
for n_estimators = 100 and max depth =  2
Log Loss : 1.5803988125353867
for n_estimators = 100 and max depth =  3
Log Loss : 1.4595203497471714
for n_estimators = 100 and max depth =  5
Log Loss : 1.2585897500107819
for n_estimators = 100 and max depth =  10
Log Loss : 1.7071291377510038
for n_estimators = 200 and max depth =  2
Log Loss : 1.5606655116633033
for n_estimators = 200 and max depth =  3
Log Loss : 1.4826626422304992
for n_estimators = 200 and max depth =  5
Log Loss : 1.3401835297294566
for n_estimators = 200 and max depth =  10
Log Loss : 1.7397210280588502
for n_estimators = 500 and max depth =  2
Log Loss : 1.6508008447694817
for n_estimators = 500 and max depth =  3
Log Loss : 1.5362160189789984
for n_estimators = 500 and max depth =  5
Log Loss : 1.351508689203686
for n_estimators = 500 and max depth =  10
Log Loss : 1.7197853851508031
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6498519089939658
for n_estimators = 1000 and max depth =  3
Log Loss : 1.545072241210793
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3353991368034757
for n_estimators = 1000 and max depth =  10
Log Loss : 1.695873900785216
For values of best alpha =  100 The train log loss is: 0.058009545185092604
For values of best alpha =  100 The cross validation log loss is: 1.2585897500107786
For values of best alpha =  100 The test log loss is: 1.2565847143840805
```

Testing model with best hyper parameters (Response Coding)

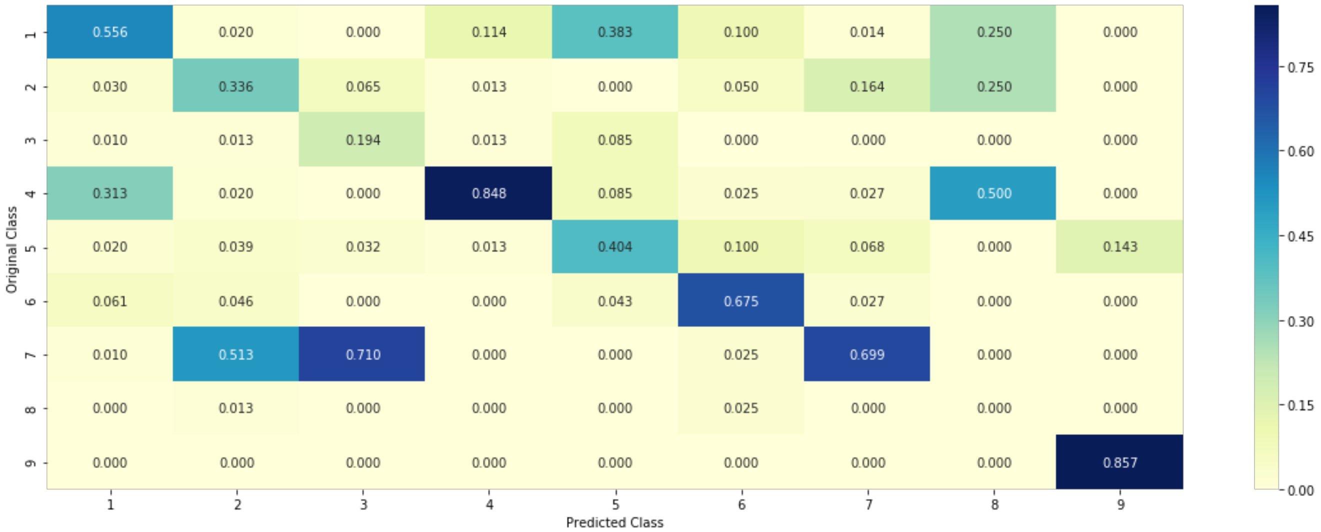
In [104]:

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

```
Log loss : 1.258589750010778
Number of mis-classified points : 0.4699248120300752
-------------------- Confusion matrix --------------------
```




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




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

Feature Importance

Incorrectly Classified point

In [105]:

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


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

```
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0276 0.5077 0.0558 0.0268 0.033  0.1015 0.1985 0.0281 0.0209]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

Correctly Classified point

In [106]:

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


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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0211 0.4529 0.0834 0.02   0.0314 0.039  0.3168 0.0223 0.0131]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
```

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

Stack the models

In [107]:

```
from mlxtend.classifier import StackingClassifier
from sklearn.linear_model import LogisticRegression
clf1 = linear_model.SGDClassifier(alpha=0.1, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(x_train_onehotCoding, y_train)
sig_clf1 = calibration.CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = linear_model.SGDClassifier(alpha=0.1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(x_train_onehotCoding, y_train)
sig_clf2 = calibration.CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.00001)
clf3.fit(x_train_onehotCoding, y_train)
sig_clf3 = calibration.CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(x_train_onehotCoding, y_train)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(y_cv, sig_clf1.predict_proba(x_cv_onehotCoding))))
sig_clf2.fit(x_train_onehotCoding, y_train)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(y_cv, sig_clf2.predict_proba(x_cv_onehotCoding))))
sig_clf3.fit(x_train_onehotCoding, y_train)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(y_cv, sig_clf3.predict_proba(x_cv_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(x_train_onehotCoding, 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_onehotCoding))))
    log_error =log_loss(y_cv, sclf.predict_proba(x_cv_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.10
Support vector machines : Log Loss: 1.20
Naive Bayes : Log Loss: 1.36
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.051
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.586
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.167
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.105
```

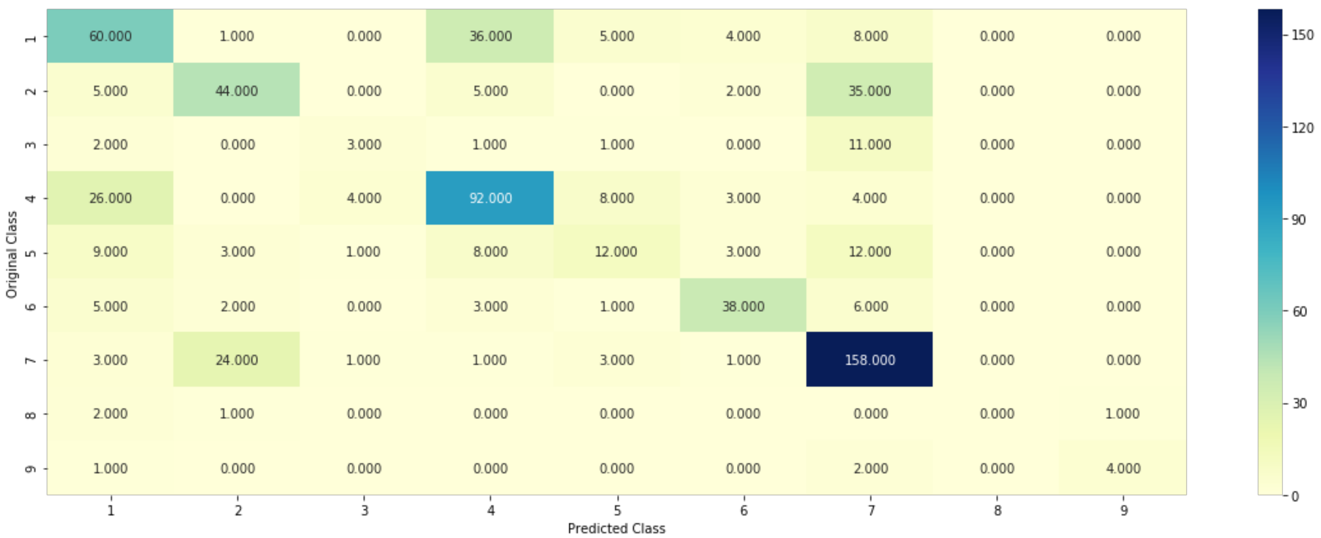
```
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.152
```

Testing with hyper parameter tuning

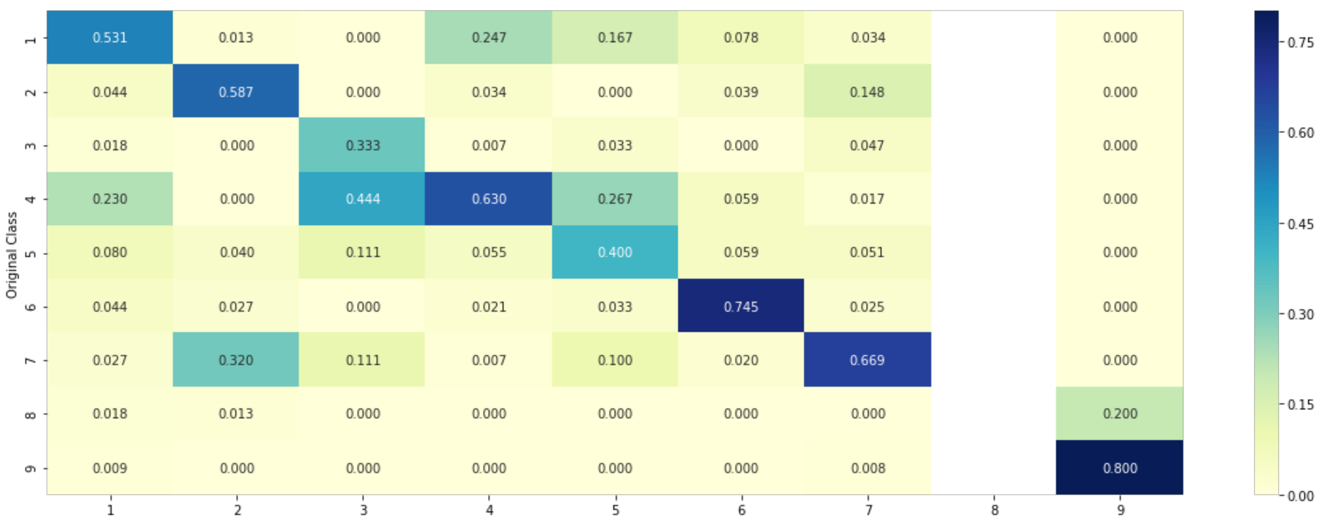
In [108]:

```
lr = LogisticRegression(C=1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(x_train_onehotCoding, y_train)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.6453177317421325
Log loss (CV) on the stacking classifier : 1.1052073091050159
Log loss (test) on the stacking classifier : 1.0726656120741174
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```

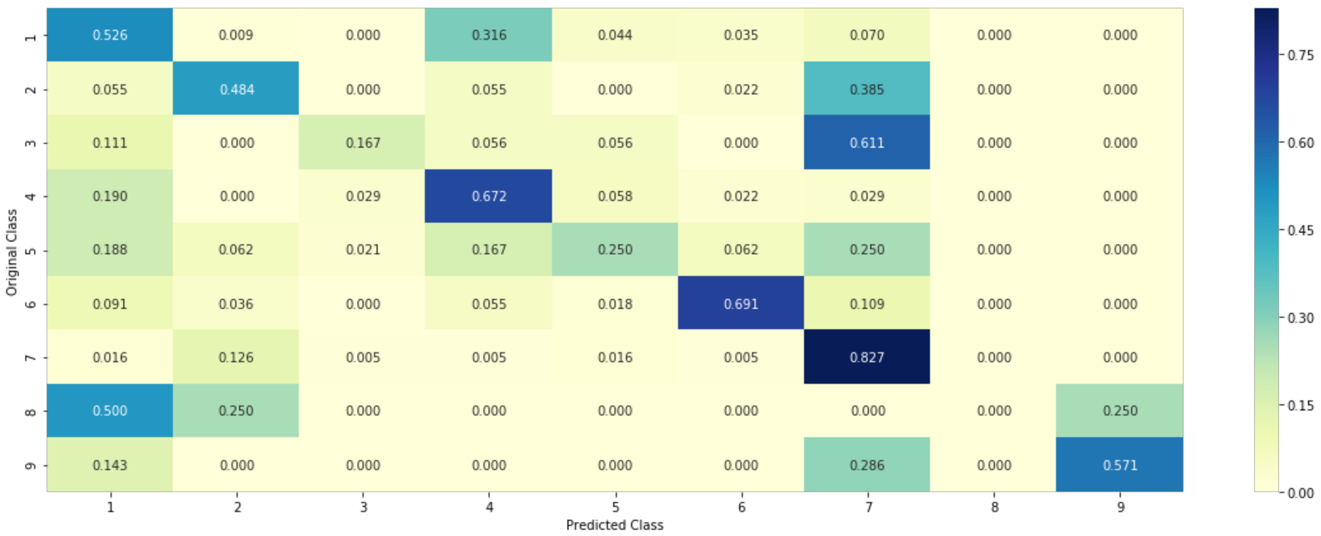



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



Predicted Class

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

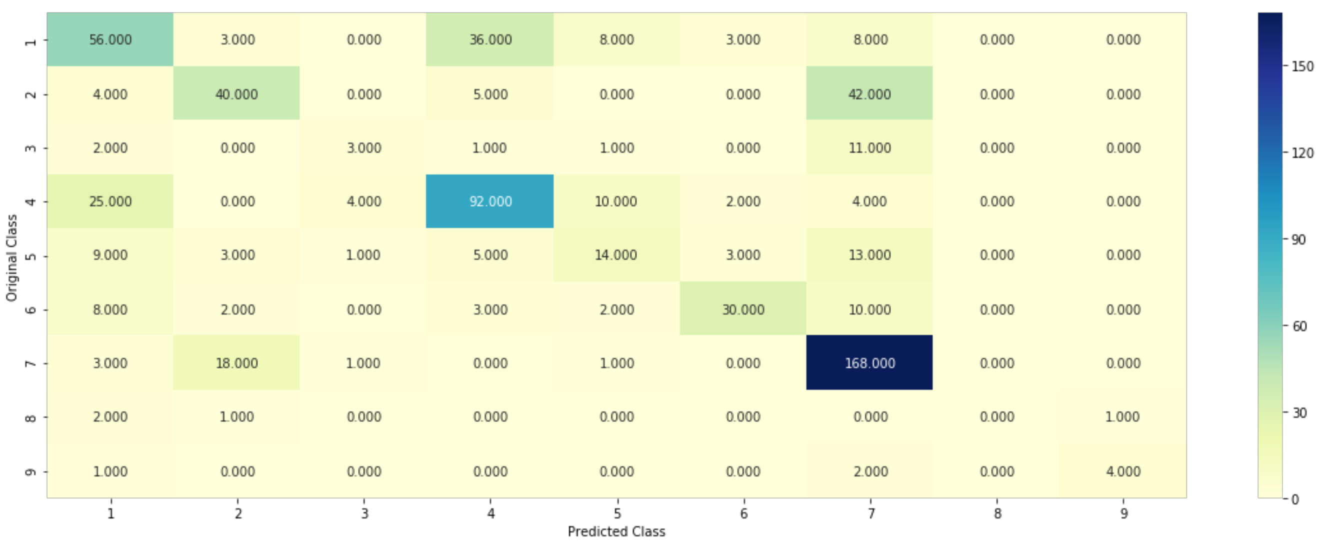


Maximum Voting classifier

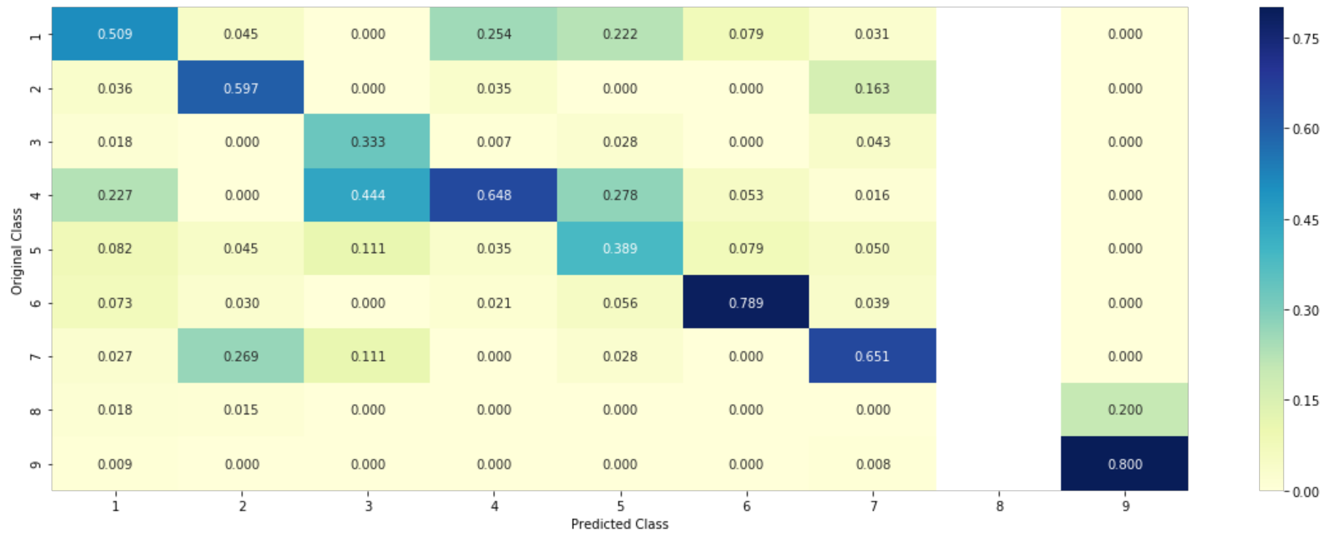
In [109]:

```python
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(x_train_onehotCoding, y_train)
print("Log loss (train) on the VotingClassifier :", log_loss(y_train,
vclf.predict_proba(x_train_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(y_cv,
vclf.predict_proba(x_cv_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(y_test,
vclf.predict_proba(x_test_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(x_test_onehotCoding)-
y_test))/y_test.shape[0])
plot_confusion_matrix(test_y=y_test, predict_y=vclf.predict(x_test_onehotCoding))
```

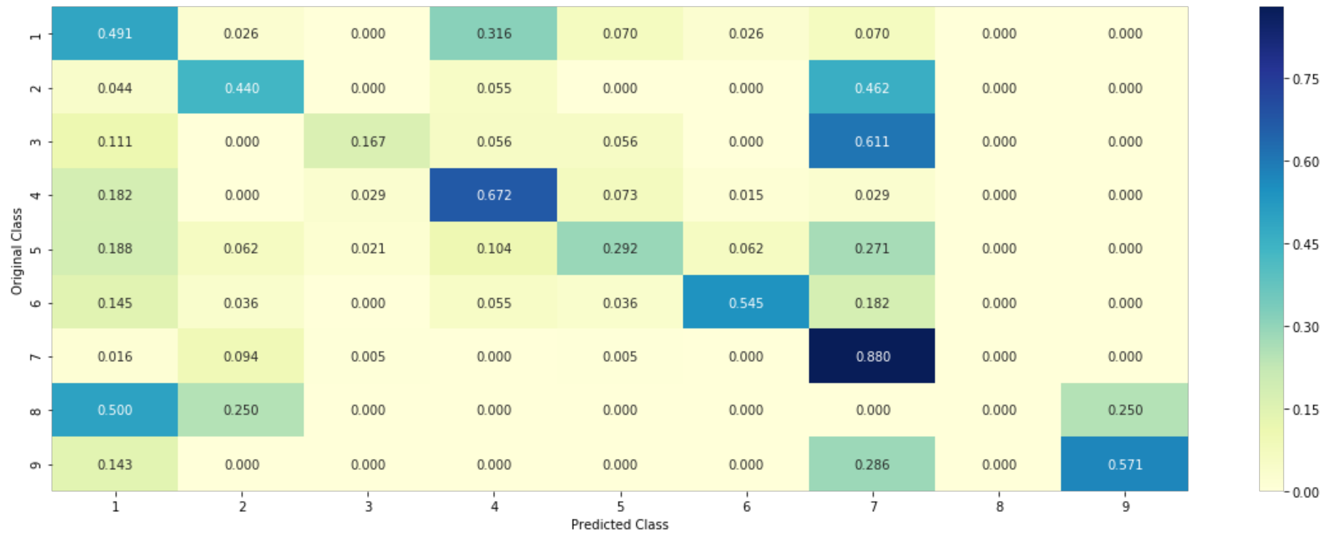
```
Log loss (train) on the VotingClassifier : 0.9115964255920648
Log loss (CV) on the VotingClassifier : 1.1655322881782668
Log loss (test) on the VotingClassifier : 1.1598954674665738
Number of missclassified point : 0.3879699248120301
-------------------- Confusion matrix --------------------
```




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



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



Count Vectorizer features Unigram and Bigram

In [110]:

```
gene_vectorizer1 = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding1 = gene_vectorizer1.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding1 = gene_vectorizer1.transform(x_test['Gene'])
cv_gene_feature_onehotCoding1 = gene_vectorizer1.transform(x_cv['Gene'])

var_vectorizer1 = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding1 = var_vectorizer1.fit_transform(x_train['Variation'])
test_variation_feature_onehotCoding1 = var_vectorizer1.transform(x_test['Variation'])
cv_variation_feature_onehotCoding1 = var_vectorizer1.transform(x_cv['Variation'])
```

In [111]:

```
from sklearn.preprocessing import normalize
text_vectorizer1 = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding1 = text_vectorizer1.fit_transform(x_train['Text'])
# getting all the feature names (words)
train_text_features1= text_vectorizer1.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) vector
train_text_fea_counts1 = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_fea_dict1 = dict(zip(list(train_text_features1),train_text_fea_counts1))
```

```
train_text_feature_onehotCoding1 = normalize(train_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding1 = text_vectorizer1.transform(x_test['Text'])
# don't forget to normalize every feature
test_text_feature_onehotCoding1 = normalize(test_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding1 = text_vectorizer1.transform(x_cv['Text'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding1 = normalize(cv_text_feature_onehotCoding, axis=0)
```

In [112]:

```
train_gene_var_onehotCoding1 =
hstack((train_gene_feature_onehotCoding1,train_variation_feature_onehotCoding1))
test_gene_var_onehotCoding1 =
hstack((test_gene_feature_onehotCoding1,test_variation_feature_onehotCoding1))
cv_gene_var_onehotCoding1 =
hstack((cv_gene_feature_onehotCoding1,cv_variation_feature_onehotCoding1))

x_train_onehotCoding1 = hstack((train_gene_var_onehotCoding1, train_text_feature_onehotCoding1)).t
ocsr()
y_train = np.array(list(x_train['Class']))

x_test_onehotCoding1 = hstack((test_gene_var_onehotCoding1, test_text_feature_onehotCoding1)).tocs
r()
y_test = np.array(list(x_test['Class']))

x_cv_onehotCoding1 = hstack((cv_gene_var_onehotCoding1, cv_text_feature_onehotCoding1)).tocsr()
y_cv = np.array(list(x_cv['Class']))
```

In [113]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", x_train_onehotCoding1.shape
)
print("(number of data points * number of features) in test data = ", x_test_onehotCoding1.shape)
print("(number of data points * number of features) in cross validation data =",
x_cv_onehotCoding1.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3363)
(number of data points * number of features) in test data =  (665, 3363)
(number of data points * number of features) in cross validation data = (532, 3363)
```

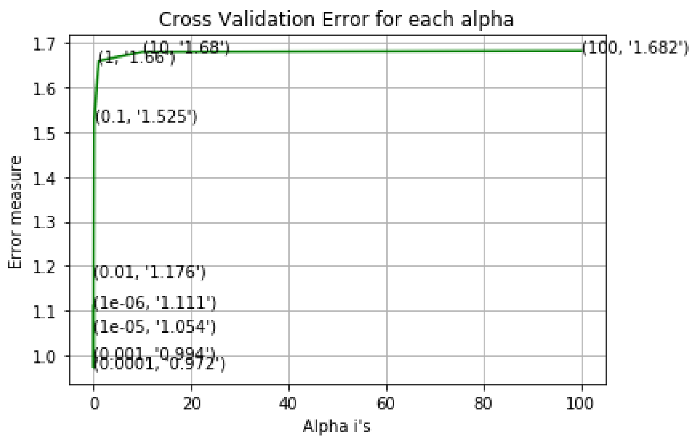
Logistic Regression using Count Vectorizer Features

In [114]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = linear_model.SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
ndom_state=42)
    clf.fit(x_train_onehotCoding, y_train)
    sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_onehotCoding1, y_train)
    sig_clf_probs = sig_clf.predict_proba(x_cv_onehotCoding1)
    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 = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(x_train_onehotCoding1, y_train)
sig_clf = calibration.CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_onehotCoding1, y_train)

predict_y = sig_clf.predict_proba(x_train_onehotCoding1)
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_onehotCoding1)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(x_test_onehotCoding1)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.111056943234811
for alpha = 1e-05
Log Loss : 1.0537416914046105
for alpha = 0.0001
Log Loss : 0.9717664408662117
for alpha = 0.001
Log Loss : 0.9943519057535836
for alpha = 0.01
Log Loss : 1.1760447017902662
for alpha = 0.1
Log Loss : 1.524682695451839
for alpha = 1
Log Loss : 1.6596672665995875
for alpha = 10
Log Loss : 1.679870418874563
for alpha = 100
Log Loss : 1.6822099335141194
```

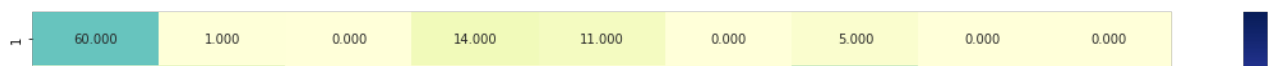



```
For values of best alpha =  0.0001 The train log loss is: 0.45106826343957257
For values of best alpha =  0.0001 The cross validation log loss is: 0.9717664408662117
For values of best alpha =  0.0001 The test log loss is: 0.9951479435687816
```

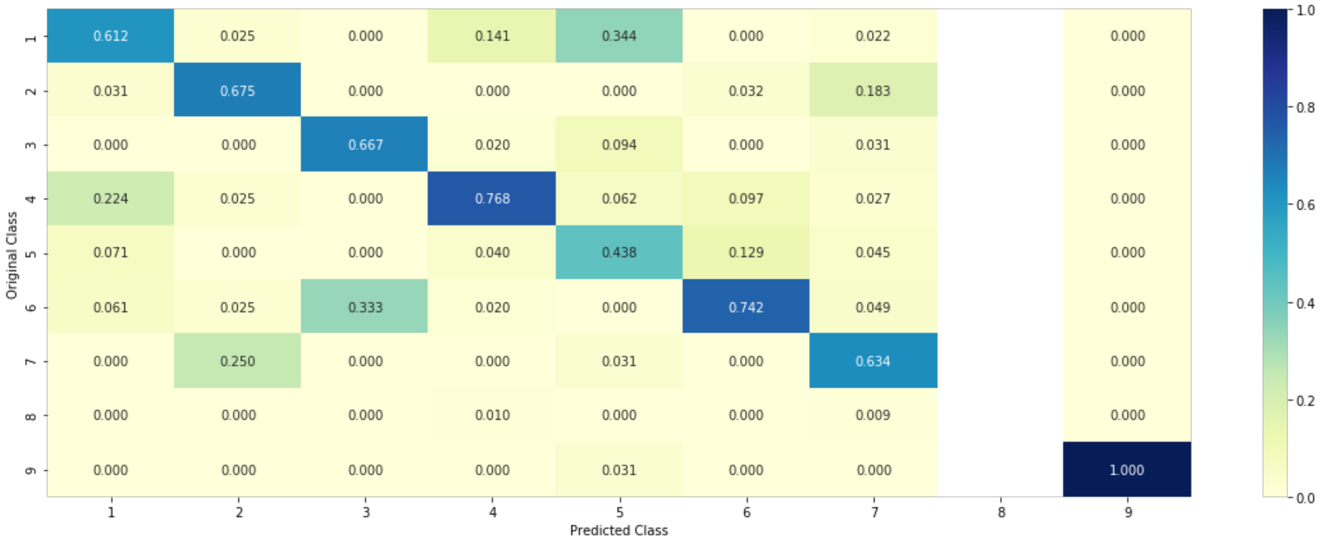
In [115]:

```
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(x_train_onehotCoding1, y_train, x_cv_onehotCoding1, y_cv, clf)
```

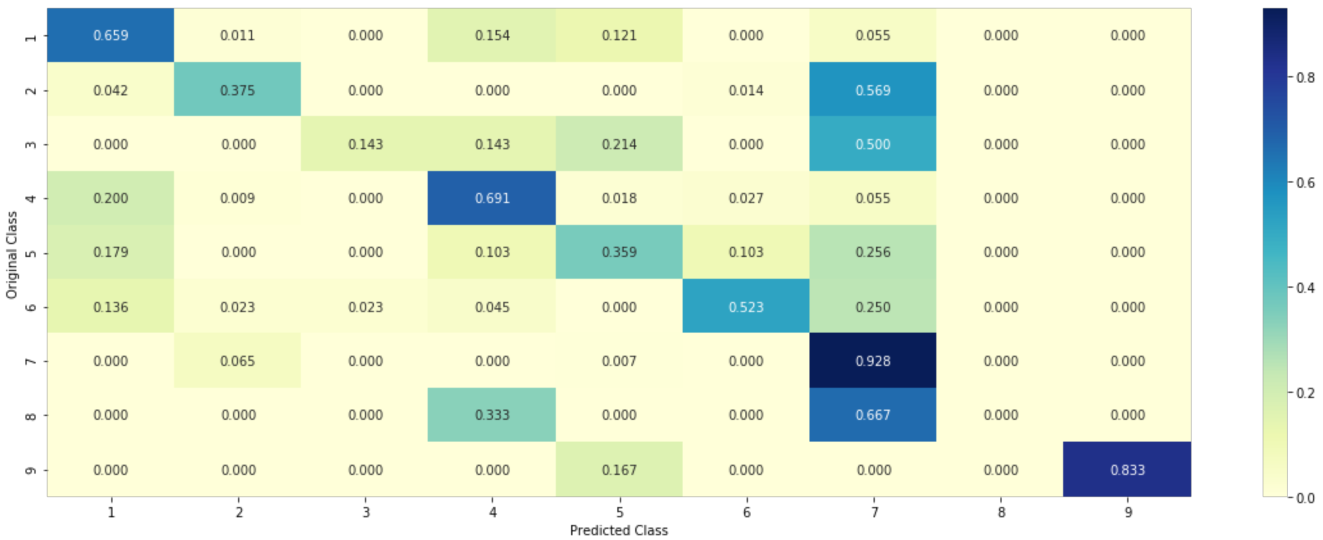
```
Log loss : 0.9717664408662117
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```




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




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



In [116]:

```
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
loss='log', random_state=42)
clf.fit(x_train_onehotCoding1,y_train)
test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding1[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(x_test_onehotCoding1[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_point_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0377 0.2815 0.0022 0.0284 0.0077 0.0635 0.5748 0.0025 0.0018]]
Actual Class : 7
--------------------------------------------------
34 Text feature [35] present in test data point [True]
44 Text feature [contribute] present in test data point [True]
45 Text feature [40] present in test data point [True]
Out of the top  50  features  3 are present in query point
```

In [117]:

```
clf = linear_model.SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(x_train_onehotCoding1,y_train)
test_point_index = 4
no_feature = 50
predicted_cls = sig_clf.predict(x_test_onehotCoding1[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(x_test_onehotCoding1[test_point_index]),4))
print("Actual Class :", y_test[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['Text'].iloc[test_point_index],x_test['Gene'].iloc[test_point_index],x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0073 0.1417 0.0114 0.0152 0.0301 0.0321 0.7578 0.0024 0.002 ]]
Actual Class : 7
--------------------------------------------------
34 Text feature [35] present in test data point [True]
44 Text feature [contribute] present in test data point [True]
45 Text feature [40] present in test data point [True]
Out of the top  50  features  3 are present in query point
```

In [1]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Model","Alpha","Train log-loss","Test log-loss","CV log-loss"]
x.add_row(["MultinomialNB","0.00001","1.15","1.36","1.33" ])
x.add_row(["KNN Classifier","5","0.49","1.05","1.02"])
x.add_row(["Logistic Regression with Class Balancing","0.1","0.813","1.114","1.104" ])
x.add_row(["Logistic Regression without Class Balancing","0.1","0.817","1.12","1.108" ])
x.add_row(["Linear SVM","0.1","0.89","1.20","1.187" ])
x.add_row(["Random Forest Classifier (one hot encoding)","100","0.55","1.07","1.07" ])
x.add_row(["Random Forest Classifier (response coding )","100","0.05","1.25","1.25" ])
x.add_row(["Stacking Classifier","-","0.64","1.07","1.10" ])
x.add_row(["Maximum Voting Classifier","-","0.91","1.15","1.16" ])
print(x)
```

```
+---------------------------------------------+---------+----------------+---------------+------------
-+
|                    Model                    |  Alpha  | Train log-loss | Test log-loss | CV log-loss
 |
+---------------------------------------------+---------+----------------+---------------+------------
-+
|                MultinomialNB                | 0.00001 |      1.15      |      1.36     |     1.33
|
|                KNN Classifier               |    5    |      0.49      |      1.05     |     1.02
|
|   Logistic Regression with Class Balancing  |   0.1   |     0.813      |     1.114     |    1.104
|
```

```
| Logistic Regression without Class Balancing |    0.1    |       0.817       |       1.12       |     1.108
|
|                  Linear SVM                 |    0.1    |        0.89       |       1.20       |     1.187
|
| Random Forest Classifier (one hot encoding) |    100    |        0.55       |       1.07       |      1.07
|
| Random Forest Classifier (response coding ) |    100    |        0.05       |       1.25       |      1.25
|
|             Stacking Classifier             |     -     |        0.64       |       1.07       |      1.10
|
|          Maximum Voting Classifier          |     -     |        0.91       |       1.15       |      1.16
|
+---------------------------------------------+-----------+-------------------+------------------+---------
-+
```

Results with CountVectorizer

In [3]:

```python
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Model","Alpha","Train log-loss","Test log-loss","CV log-loss"]

x.add_row(["Logistic Regression with Class Balancing","0.0001","0.45","0.995","0.97" ])
print(x)
```

```
+-------------------------------------------+--------+----------------+---------------+-------------
|                   Model                   | Alpha  | Train log-loss | Test log-loss | CV log-loss
+-------------------------------------------+--------+----------------+---------------+-------------
| Logistic Regression with Class Balancing  | 0.0001 |      0.45      |     0.995     |     0.97
|
+-------------------------------------------+--------+----------------+---------------+-------------
```

In [ ]: