Personalized cancer diagnosis

In [1]:

```
# Importing all the neccessary packages

import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
import seaborn as sns
import math
warnings.filterwarnings("ignore")

from collections import Counter
from scipy.sparse import hstack
from nltk.corpus import stopwords
from collections import Counter, defaultdict
from mlxtend.classifier import StackingClassifier


from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

Reading Data

Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

	ID	Gene	Variation	Class
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

Reading Text Data

In [3]:

```
# Since the text data is seperated by || we need to use sep

data_text = pd.read_csv("training_text", sep = "\|\|", engine = "python", names = ["ID","TEXT"], sk
iprows = 1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

	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 Text Data

In [4]:

```
stop_words = set(stopwords.words('english'))

def nlp_preprocessing(total_text, index, column):
    """
    Function that will remove special characters, extra spaces, stop words and convert
    all the text data into lower case.
    """
    if type(total_text) is not int:
        string = ""
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        total_text = re.sub('\s+',' ', total_text)
        total_text = total_text.lower()

        for word in total_text.split():
            if not word in stop_words:
                string += word + " "

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

In [5]:

```
# Processing Text data

start_time = time.clock()

for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
```

```
    else:
        print("there is no text description for id:", index)

print('\nTime took for preprocessing the text :', time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755

Time took for preprocessing the text : 72.6869866 seconds
```

In [6]:

```
# Merging both variants and text file based on ID

result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

In [7]:

```
# Checking data points with missing values

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

Out[7]:

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

In [8]:

```
# As we can see that some datapoints dont have text data so for those data points we will put gene +variation name

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

In [9]:

```
# Chcking the datapoints

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

Out[9]:

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

Splitting Dataset

Splitting data into Train, Test and CV in ration - 64:20:16

In [10]:

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

X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify = y_true, test_size = 0.2)
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify = y_train, test_size = 0.2)
```

In [11]:

```
# Checking datapoints in Train, CV and Test data

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

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

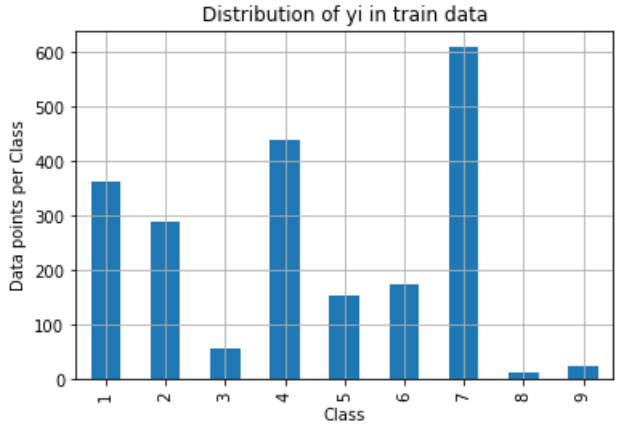
Exploratory Data Analysis

Distribution of y_i's in Train, Test and CV datasets

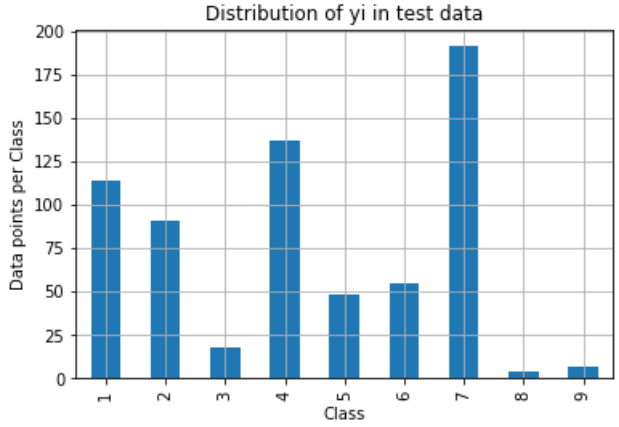
In [13]:

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

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

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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
```

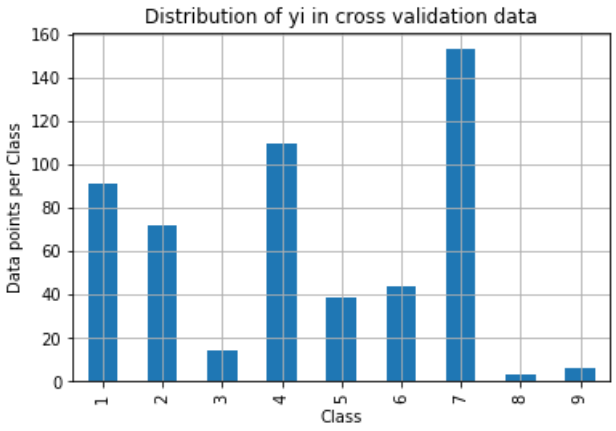
```
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

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




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




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




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

Observations

- Since we have split the dataset by random so the distribution of classes in Train, Test and Cv is nearly the same.
- The given dataset is highly imbalance dataset.
- The dataset have classes 7, 4, 1 and 2 as majority classes
- The dataset have classes 3, 4, 8, 9 as minority classes

Deciding worst multiclass log-loss using a Random model

In [14]:

```
def plot_confusion_matrix(test_y, predict_y):
    """
    This function plots the confusion matrices given y_i, y_i_hat
    """
    C = confusion_matrix(test_y, predict_y)
    A = (((C.T) / (C.sum(axis = 1))).T)
    B =(C / C.sum(axis = 0))
    labels = [1,2,3,4,5,6,7,8,9]

    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize = (20,7))
    sns.heatmap(C, annot = True, cmap = "YlGnBu", fmt = ".3f", xticklabels = labels, yticklabels = l
abels)
    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 = l
abels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    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 = l
abels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
```

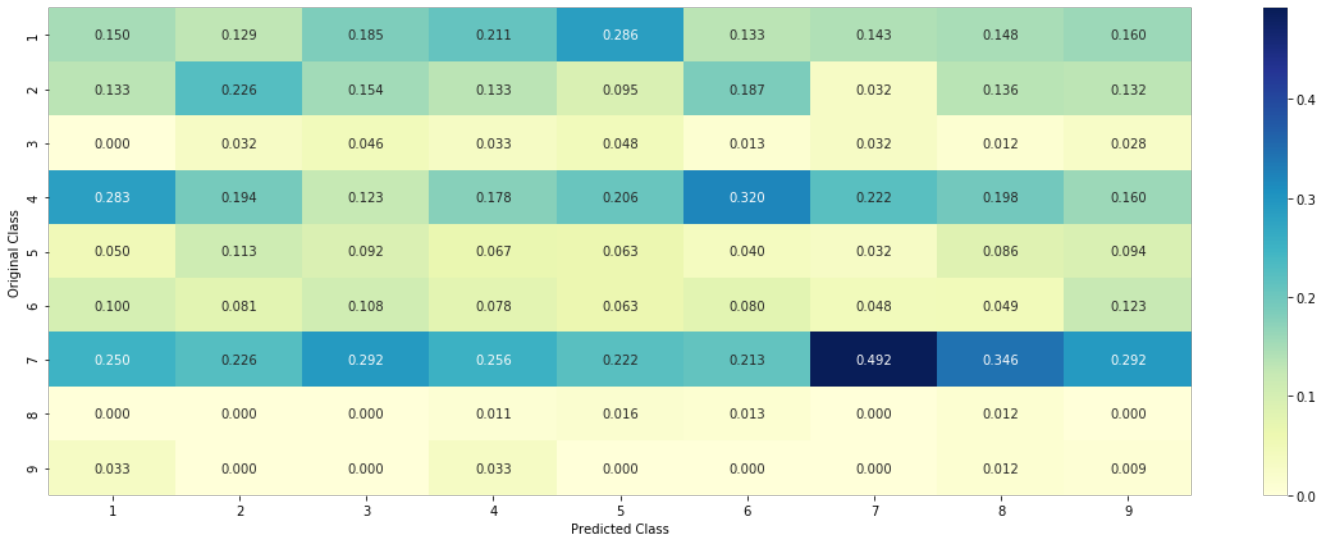
```
    plt.show()
```

In [15]:

```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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_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 2.493197325756756
Log loss on Test Data using Random Model 2.471926090812876
-------------------- Confusion matrix --------------------
```


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



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

Observations

- As we can see from the Random model the worst multiclass log loss for this problem is 2.47.
- The model that we build should have the log loss below 2.47.

Univariate Analysis

In [16]:

```
def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()
    gv_dict = dict()

    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        gv_dict[i]=vec
    return gv_dict

def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = train_df[feature].value_counts()
    gv_fea = []

    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    return gv_fea
```

Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [17]:

```
unique_genes = train_df['Gene'].value_counts()
```

```
print('Number of Unique Genes :', unique_genes.shape[0])

# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 221
BRCA1     172
TP53       98
EGFR       84
BRCA2      82
PTEN       79
KIT        70
BRAF       63
ALK        48
ERBB2      46
PIK3CA     38
Name: Gene, dtype: int64
```

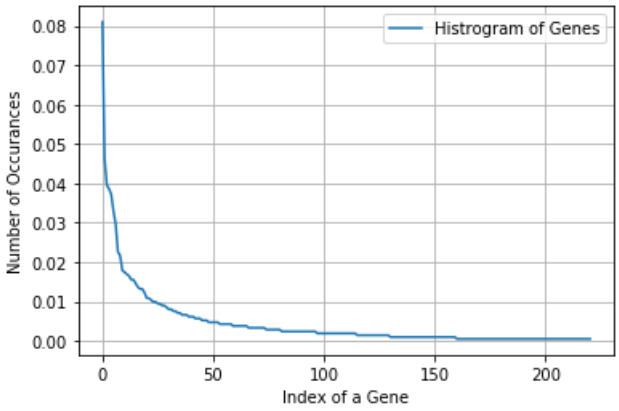
In [18]:

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

```
Ans: There are 221 different categories of genes in the train data, and they are distibuted as follows
```

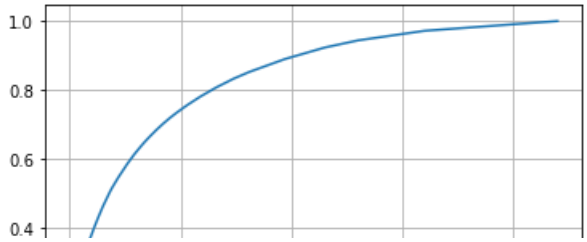
In [19]:

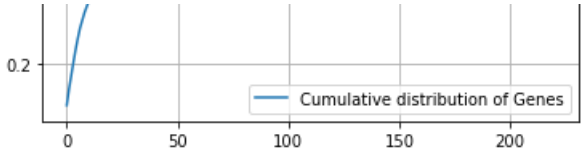
```
s = sum(unique_genes.values)
h = unique_genes.values/s

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 [20]:

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





Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable

1. One hot Encoding
2. Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

In [21]:

```
alpha = 1

train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [22]:

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

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

In [23]:

```
# one-hot encoding of Gene feature

gene_vectorizer = CountVectorizer()

train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [24]:

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

Out[24]:

```
1235     PIM1
1947    MEF2B
349      CDH1
1293     HRAS
1523      ALK
Name: Gene, dtype: object
```

In [25]:

```
gene_vectorizer.get_feature_names()
```

Out[25]:

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

```
'ar',
'araf',
'arid1b',
'arid2',
'arid5b',
'atm',
'atr',
'atrx',
'aurka',
'aurkb',
'axl',
'b2m',
'bap1',
'bard1',
'bcl10',
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
```

```
'flt3',
'foxa1',
'foxl2',
'gata3',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
```

```
 'pms2',
 'pole',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rac1',
 'rad50',
 'rad51c',
 'rad51d',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'ros1',
 'runx1',
 'rxra',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [26]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 220)
```

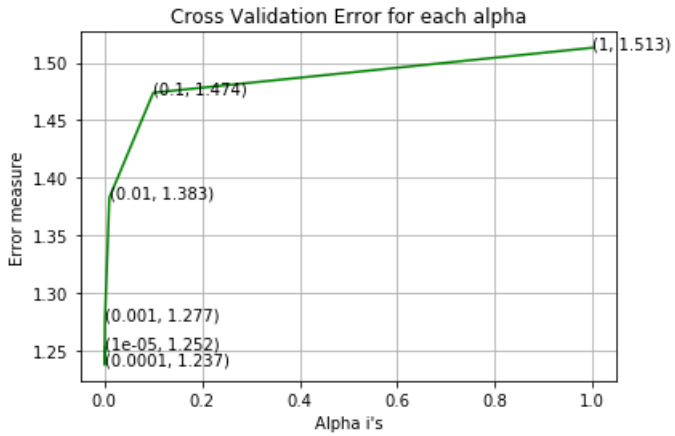
Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [27]:

```
# Checking How much important is gene feature in prediction the y_i
# Applying simple lR with calibrated Classifier

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

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

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

for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))

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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.2519727022733864
For values of alpha =  0.0001 The log loss is: 1.2374780120880384
For values of alpha =  0.001 The log loss is: 1.2765209731105742
For values of alpha =  0.01 The log loss is: 1.3826951230597384
For values of alpha =  0.1 The log loss is: 1.4738753485882086
For values of alpha =  1 The log loss is: 1.5129240427028416
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9801066159682027
For values of best alpha =  0.0001 The cross validation log loss is: 1.2374780120880384
For values of best alpha =  0.0001 The test log loss is: 1.1825686001504472
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

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

In [28]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0
], " genes in train dataset?")

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  221  genes in train dataset?
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 508 out of  532 : 95.48872180451127
```

Observations

- As we can see that alone gene feature has reduced the log loss from 2.47 to 1.18.
- Which is a significant reduction.
- Gene feature is important.

Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [29]:

```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])

# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1936
Truncating_Mutations    54
Deletion                47
Amplification           44
Fusions                 19
G12V                     4
Overexpression           4
Q61H                     3
E330K                    2
G13D                     2
EWSR1-ETV1_Fusion        2
Name: Variation, dtype: int64
```

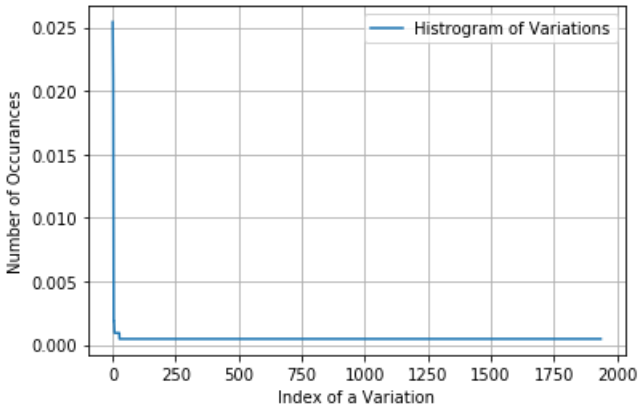
In [30]:

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

```
Ans: There are 1936 different categories of variations in the train data, and they are distibuted
as follows
```

In [31]:

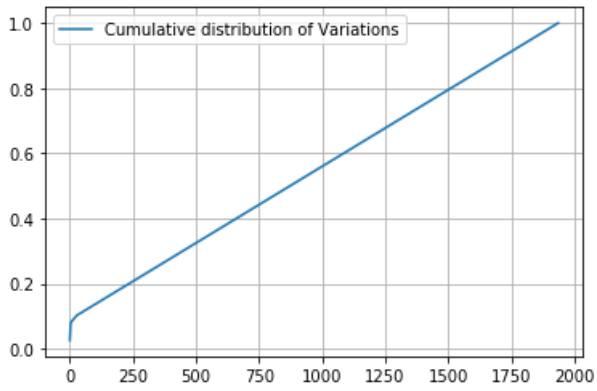
```
s = sum(unique_variations.values);
h = unique_variations.values / s;

plt.plot(h, label = "Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [32]:

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

```
[0.02542373 0.04755179 0.06826742 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable

1. One hot Encoding
2. Response coding

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

In [33]:

```
alpha = 1

train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [34]:

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

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

In [35]:

```
# one-hot encoding of variation feature.

variation_vectorizer = CountVectorizer()

train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [36]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1965)
```

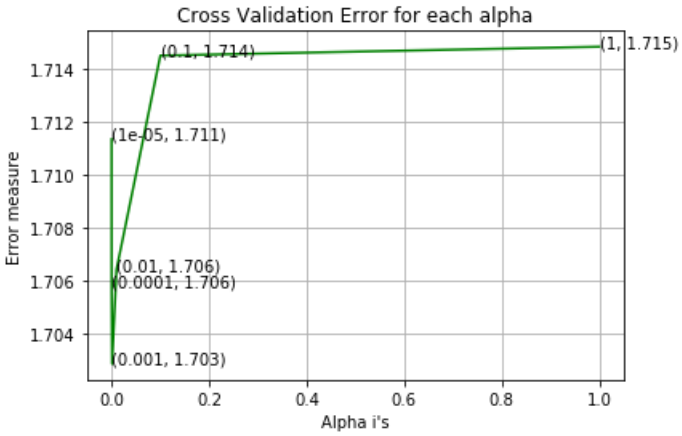
Q10. How good is this Variation feature in predicting y_i?

In [37]:

```
# As earlier buildind a simple Logistic Regression Model with Calibrated Classifer for variation f
eature

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

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

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

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

for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))

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

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

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

```
predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7113269971790552
For values of alpha =  0.0001 The log loss is: 1.7057748031123203
For values of alpha =  0.001 The log loss is: 1.702867177640526
For values of alpha =  0.01 The log loss is: 1.706417070080605
For values of alpha =  0.1 The log loss is: 1.714480832584981
For values of alpha =  1 The log loss is: 1.7148138892068714
```




```
For values of best alpha =  0.001 The train log loss is: 1.0955372575095133
For values of best alpha =  0.001 The cross validation log loss is: 1.702867177640526
For values of best alpha =  0.001 The test log loss is: 1.6993409173046865
```

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

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

In [38]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
st and cross validation data sets?")

test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]

print('Ans: 1. In test data',test_coverage, 'out of', test_df.shape[0], ":", (test_coverage / test_
df.shape[0]) * 100)
print('\t2. In cross validation data',cv_coverage, 'out of ', cv_df.shape[0], ":" , (cv_coverage /
cv_df.shape[0]) * 100)
```

```
Q12. How many data points are covered by total  1936  genes in test and cross validation data
sets?
Ans: 1. In test data 75 out of 665 : 11.278195488721805
 2. In cross validation data 57 out of  532 : 10.714285714285714
```

Observations

- By using only variation feature the log loss is reduced to 1.70.
- The loss is not reduced as much as it reduced in gene feature.
- Also the variotions in test and CV data is very different from the training.
- Variation feature is important but not as much as gene feature.

Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?

5. Is the text feature stable across train, test and CV datasets?

In [39]:

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

In [40]:

```
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 [41]:

```
text_vectorizer = CountVectorizer(min_df=3)

train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_features= text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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 : 53760
```

In [42]:

```
dict_list = []

for i in range(1,10):
    cls_text = train_df[train_df['Class'] == i]
    dict_list.append(extract_dictionary_paddle(cls_text))

total_dict = extract_dictionary_paddle(train_df)
confuse_array = []

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

confuse_array = np.array(confuse_array)
```

In [43]:

```
# Response coding of text features

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

In [44]:

```
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 [45]:

```
# Normalizing

train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

In [46]:

```
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 [47]:

```
# Number of words for a given frequency

print(Counter(sorted_text_occur))
```

```
Counter({3: 5833, 4: 3805, 5: 2809, 6: 2693, 8: 1997, 7: 1987, 9: 1802, 10: 1653, 12: 1180, 13: 116
6, 11: 1101, 14: 988, 15: 968, 16: 731, 18: 667, 20: 579, 17: 576, 19: 534, 24: 531, 21: 499, 22:
444, 25: 431, 30: 412, 28: 409, 23: 396, 27: 366, 26: 352, 45: 334, 52: 295, 32: 295, 29: 278, 33:
261, 36: 257, 31: 255, 35: 253, 40: 239, 34: 238, 42: 232, 39: 213, 38: 210, 37: 193, 44: 190, 41:
181, 51: 157, 56: 156, 54: 156, 48: 156, 57: 154, 55: 154, 47: 149, 60: 143, 50: 142, 43: 137, 46:
135, 49: 134, 69: 123, 53: 121, 63: 118, 58: 118, 61: 113, 62: 109, 64: 107, 59: 107, 70: 104, 67:
103, 72: 100, 75: 98, 65: 96, 68: 94, 84: 93, 71: 90, 81: 89, 90: 88, 66: 88, 76: 86, 78: 85, 80:
81, 77: 79, 74: 74, 87: 73, 93: 72, 82: 72, 73: 72, 88: 71, 108: 67, 105: 67, 91: 66, 86: 61, 98:
60, 95: 60, 85: 60, 104: 59, 96: 58, 92: 58, 79: 56, 100: 55, 99: 53, 89: 53, 126: 52, 106: 52, 1
03: 52, 120: 51, 112: 51, 94: 50, 83: 49, 109: 47, 117: 46, 111: 46, 110: 46, 113: 45, 134: 43, 13
1: 43, 121: 43, 135: 42, 115: 42, 107: 42, 102: 42, 114: 41, 128: 40, 124: 39, 97: 39, 158: 38, 14
5: 38, 129: 38, 118: 38, 143: 37, 141: 37, 140: 36, 138: 36, 122: 36, 119: 36, 155: 35, 148: 35, 1
30: 34, 125: 34, 213: 33, 144: 33, 133: 33, 127: 33, 116: 33, 156: 32, 152: 32, 150: 32, 159: 31,
136: 31, 204: 30, 137: 30, 123: 30, 180: 29, 164: 29, 149: 29, 139: 29, 132: 29, 212: 28, 211: 28,
195: 28, 174: 28, 170: 28, 168: 28, 147: 28, 142: 28, 179: 27, 171: 27, 162: 27, 101: 27, 175: 26,
172: 26, 169: 26, 157: 25, 201: 24, 177: 24, 166: 24, 151: 24, 279: 23, 227: 23, 205: 23, 196: 23,
182: 23, 178: 23, 163: 23, 146: 23, 250: 22, 246: 22, 226: 22, 209: 22, 203: 22, 200: 22, 190: 22,
181: 22, 176: 22, 167: 22, 160: 22, 153: 22, 282: 21, 241: 21, 221: 21, 208: 21, 198: 21, 194: 21,
192: 21, 161: 21, 260: 20, 188: 20, 186: 20, 185: 20, 184: 20, 214: 19, 210: 19, 202: 19, 154: 19,
291: 18, 278: 18, 256: 18, 247: 18, 230: 18, 228: 18, 220: 18, 207: 18, 183: 18, 165: 18, 304: 17,
263: 17, 229: 17, 224: 17, 223: 17, 218: 17, 216: 17, 215: 17, 197: 17, 189: 17, 266: 16, 173: 16,
314: 15, 301: 15, 299: 15, 298: 15, 288: 15, 277: 15, 274: 15, 239: 15, 234: 15, 222: 15, 219: 15,
206: 15, 391: 14, 356: 14, 333: 14, 290: 14, 283: 14, 280: 14, 258: 14, 243: 14, 240: 14, 238: 14,
235: 14, 199: 14, 193: 14, 187: 14, 387: 13, 369: 13, 327: 13, 324: 13, 319: 13, 318: 13, 312: 13,
309: 13, 293: 13, 289: 13, 267: 13, 254: 13, 252: 13, 245: 13, 242: 13, 237: 13, 357: 12, 316: 12,
313: 12, 294: 12, 287: 12, 281: 12, 271: 12, 265: 12, 261: 12, 255: 12, 244: 12, 233: 12, 225: 12,
217: 12, 191: 12, 405: 11, 390: 11, 388: 11, 358: 11, 355: 11, 346: 11, 342: 11, 323: 11, 285: 11,
276: 11, 270: 11, 264: 11, 262: 11, 257: 11, 253: 11, 248: 11, 232: 11, 438: 10, 396: 10, 360: 10,
348: 10, 344: 10, 334: 10, 326: 10, 321: 10, 320: 10, 311: 10, 300: 10, 295: 10, 272: 10, 236: 10,
508: 9, 460: 9, 452: 9, 444: 9, 435: 9, 407: 9, 401: 9, 399: 9, 368: 9, 364: 9, 359: 9, 351: 9,
345: 9, 341: 9, 338: 9, 331: 9, 325: 9, 308: 9, 297: 9, 292: 9, 284: 9, 275: 9, 249: 9, 754: 8,
653: 8, 572: 8, 552: 8, 513: 8, 488: 8, 487: 8, 470: 8, 455: 8, 434: 8, 431: 8, 429: 8, 426: 8,
418: 8, 417: 8, 413: 8, 409: 8, 403: 8, 400: 8, 394: 8, 380: 8, 376: 8, 374: 8, 361: 8, 353: 8,
340: 8, 339: 8, 337: 8, 306: 8, 273: 8, 269: 8, 268: 8, 251: 8, 872: 7, 797: 7, 723: 7, 661: 7,
639: 7, 636: 7, 613: 7, 598: 7, 565: 7, 563: 7, 548: 7, 538: 7, 522: 7, 503: 7, 501: 7, 495: 7,
467: 7, 450: 7, 449: 7, 445: 7, 442: 7, 440: 7, 436: 7, 430: 7, 411: 7, 406: 7, 384: 7, 377: 7,
372: 7, 347: 7, 343: 7, 332: 7, 330: 7, 310: 7, 303: 7, 259: 7, 231: 7, 886: 6, 851: 6, 806: 6,
775: 6, 772: 6, 757: 6, 692: 6, 675: 6, 638: 6, 631: 6, 615: 6, 611: 6, 570: 6, 554: 6, 542: 6,
541: 6, 540: 6, 526: 6, 520: 6, 509: 6, 506: 6, 498: 6, 491: 6, 472: 6, 468: 6, 465: 6, 454: 6,
453: 6, 447: 6, 446: 6, 439: 6, 437: 6, 428: 6, 422: 6, 415: 6, 414: 6, 412: 6, 410: 6, 402: 6,
395: 6, 393: 6, 392: 6, 389: 6, 385: 6, 383: 6, 382: 6, 381: 6, 379: 6, 378: 6, 367: 6, 363: 6,
329: 6, 322: 6, 315: 6, 302: 6, 296: 6, 286: 6, 1632: 5, 1234: 5, 1153: 5, 913: 5, 874: 5, 868: 5,
860: 5, 847: 5, 834: 5, 831: 5, 815: 5, 804: 5, 802: 5, 795: 5, 781: 5, 770: 5, 767: 5, 752: 5,
```

724: 5, 714: 5, 694: 5, 668: 5, 652: 5, 650: 5, 644: 5, 632: 5, 624: 5, 607: 5, 605: 5, 602: 5,
601: 5, 599: 5, 592: 5, 586: 5, 575: 5, 571: 5, 566: 5, 550: 5, 549: 5, 545: 5, 544: 5, 543: 5,
537: 5, 536: 5, 529: 5, 527: 5, 525: 5, 514: 5, 511: 5, 499: 5, 489: 5, 485: 5, 482: 5, 474: 5,
473: 5, 466: 5, 462: 5, 461: 5, 457: 5, 433: 5, 432: 5, 427: 5, 420: 5, 419: 5, 404: 5, 370: 5,
336: 5, 307: 5, 305: 5, 2084: 4, 1876: 4, 1711: 4, 1609: 4, 1452: 4, 1430: 4, 1356: 4, 1226: 4, 120
4: 4, 1180: 4, 1174: 4, 1148: 4, 1115: 4, 1107: 4, 1065: 4, 1025: 4, 966: 4, 960: 4, 927: 4, 926: 4
, 904: 4, 893: 4, 884: 4, 883: 4, 882: 4, 854: 4, 850: 4, 842: 4, 839: 4, 828: 4, 818: 4, 809: 4,
800: 4, 798: 4, 788: 4, 764: 4, 756: 4, 750: 4, 742: 4, 735: 4, 734: 4, 722: 4, 721: 4, 720: 4,
718: 4, 717: 4, 709: 4, 707: 4, 701: 4, 700: 4, 699: 4, 693: 4, 681: 4, 676: 4, 667: 4, 654: 4,
648: 4, 637: 4, 635: 4, 623: 4, 619: 4, 616: 4, 614: 4, 610: 4, 606: 4, 603: 4, 591: 4, 590: 4,
589: 4, 587: 4, 581: 4, 574: 4, 564: 4, 562: 4, 560: 4, 558: 4, 555: 4, 532: 4, 524: 4, 516: 4,
515: 4, 507: 4, 505: 4, 497: 4, 478: 4, 475: 4, 471: 4, 456: 4, 448: 4, 443: 4, 425: 4, 423: 4,
421: 4, 416: 4, 408: 4, 398: 4, 397: 4, 386: 4, 375: 4, 371: 4, 366: 4, 362: 4, 352: 4, 350: 4,
349: 4, 328: 4, 317: 4, 4527: 3, 3572: 3, 3413: 3, 2648: 3, 2645: 3, 2581: 3, 2527: 3, 2460: 3, 245
9: 3, 2368: 3, 2332: 3, 2269: 3, 2155: 3, 2065: 3, 2036: 3, 2018: 3, 1939: 3, 1925: 3, 1886: 3, 184
8: 3, 1837: 3, 1813: 3, 1786: 3, 1696: 3, 1650: 3, 1636: 3, 1605: 3, 1581: 3, 1576: 3, 1571: 3, 155
9: 3, 1527: 3, 1514: 3, 1490: 3, 1481: 3, 1479: 3, 1470: 3, 1458: 3, 1456: 3, 1428: 3, 1410: 3, 138
3: 3, 1374: 3, 1370: 3, 1365: 3, 1348: 3, 1335: 3, 1315: 3, 1312: 3, 1306: 3, 1301: 3, 1283: 3, 127
8: 3, 1261: 3, 1257: 3, 1256: 3, 1253: 3, 1245: 3, 1231: 3, 1214: 3, 1203: 3, 1201: 3, 1200: 3, 117
5: 3, 1167: 3, 1163: 3, 1140: 3, 1133: 3, 1120: 3, 1119: 3, 1111: 3, 1109: 3, 1108: 3, 1069: 3, 106
2: 3, 1047: 3, 1044: 3, 1039: 3, 1023: 3, 1010: 3, 1004: 3, 999: 3, 994: 3, 990: 3, 987: 3, 983: 3,
977: 3, 970: 3, 964: 3, 959: 3, 957: 3, 955: 3, 953: 3, 952: 3, 930: 3, 924: 3, 919: 3, 918: 3,
914: 3, 909: 3, 908: 3, 903: 3, 899: 3, 880: 3, 878: 3, 877: 3, 866: 3, 848: 3, 846: 3, 844: 3,
843: 3, 835: 3, 830: 3, 826: 3, 824: 3, 823: 3, 822: 3, 817: 3, 805: 3, 780: 3, 778: 3, 777: 3,
773: 3, 760: 3, 747: 3, 746: 3, 745: 3, 740: 3, 731: 3, 729: 3, 725: 3, 715: 3, 710: 3, 705: 3,
703: 3, 698: 3, 691: 3, 688: 3, 686: 3, 685: 3, 684: 3, 678: 3, 677: 3, 673: 3, 665: 3, 658: 3,
649: 3, 646: 3, 643: 3, 640: 3, 634: 3, 630: 3, 625: 3, 620: 3, 617: 3, 597: 3, 595: 3, 594: 3,
584: 3, 578: 3, 577: 3, 559: 3, 556: 3, 547: 3, 546: 3, 539: 3, 531: 3, 528: 3, 519: 3, 518: 3,
517: 3, 510: 3, 502: 3, 500: 3, 496: 3, 494: 3, 493: 3, 492: 3, 490: 3, 481: 3, 480: 3, 469: 3,
464: 3, 463: 3, 459: 3, 451: 3, 424: 3, 373: 3, 354: 3, 12195: 2, 7917: 2, 7035: 2, 6635: 2, 6294:
2, 5766: 2, 5728: 2, 4883: 2, 4842: 2, 4803: 2, 4400: 2, 4364: 2, 4220: 2, 4175: 2, 3916: 2, 3738:
2, 3736: 2, 3722: 2, 3554: 2, 3495: 2, 3483: 2, 3467: 2, 3452: 2, 3432: 2, 3381: 2, 3323: 2, 3288:
2, 3287: 2, 3275: 2, 3223: 2, 3212: 2, 3200: 2, 3151: 2, 2997: 2, 2964: 2, 2890: 2, 2798: 2, 2792:
2, 2753: 2, 2750: 2, 2740: 2, 2683: 2, 2638: 2, 2633: 2, 2603: 2, 2592: 2, 2571: 2, 2568: 2, 2564:
2, 2545: 2, 2501: 2, 2498: 2, 2491: 2, 2423: 2, 2409: 2, 2377: 2, 2361: 2, 2316: 2, 2268: 2, 2204:
2, 2177: 2, 2144: 2, 2142: 2, 2129: 2, 2128: 2, 2110: 2, 2103: 2, 2100: 2, 2093: 2, 2081: 2, 2069:
2, 2057: 2, 2049: 2, 2048: 2, 2042: 2, 2001: 2, 1995: 2, 1990: 2, 1974: 2, 1973: 2, 1964: 2, 1949:
2, 1938: 2, 1930: 2, 1923: 2, 1911: 2, 1891: 2, 1890: 2, 1884: 2, 1880: 2, 1857: 2, 1854: 2, 1850:
2, 1843: 2, 1833: 2, 1827: 2, 1812: 2, 1802: 2, 1798: 2, 1788: 2, 1781: 2, 1778: 2, 1774: 2, 1732:
2, 1731: 2, 1728: 2, 1720: 2, 1718: 2, 1716: 2, 1714: 2, 1708: 2, 1704: 2, 1703: 2, 1701: 2, 1668:
2, 1662: 2, 1657: 2, 1649: 2, 1638: 2, 1637: 2, 1635: 2, 1627: 2, 1619: 2, 1617: 2, 1613: 2, 1603:
2, 1597: 2, 1593: 2, 1592: 2, 1589: 2, 1582: 2, 1579: 2, 1574: 2, 1566: 2, 1565: 2, 1562: 2, 1556:
2, 1531: 2, 1529: 2, 1523: 2, 1513: 2, 1512: 2, 1508: 2, 1495: 2, 1483: 2, 1469: 2, 1455: 2, 1449:
2, 1438: 2, 1425: 2, 1419: 2, 1418: 2, 1413: 2, 1409: 2, 1407: 2, 1399: 2, 1392: 2, 1382: 2, 1373:
2, 1368: 2, 1361: 2, 1358: 2, 1357: 2, 1351: 2, 1349: 2, 1346: 2, 1344: 2, 1342: 2, 1328: 2, 1327:
2, 1323: 2, 1319: 2, 1317: 2, 1299: 2, 1295: 2, 1289: 2, 1288: 2, 1285: 2, 1279: 2, 1275: 2, 1273:
2, 1272: 2, 1271: 2, 1266: 2, 1260: 2, 1254: 2, 1249: 2, 1243: 2, 1236: 2, 1232: 2, 1229: 2, 1224:
2, 1223: 2, 1222: 2, 1220: 2, 1219: 2, 1218: 2, 1215: 2, 1210: 2, 1207: 2, 1206: 2, 1199: 2, 1196:
2, 1190: 2, 1188: 2, 1187: 2, 1177: 2, 1176: 2, 1170: 2, 1169: 2, 1161: 2, 1156: 2, 1152: 2, 1146:
2, 1142: 2, 1139: 2, 1138: 2, 1136: 2, 1134: 2, 1132: 2, 1130: 2, 1127: 2, 1126: 2, 1118: 2, 1104:
2, 1102: 2, 1084: 2, 1081: 2, 1078: 2, 1076: 2, 1072: 2, 1068: 2, 1066: 2, 1064: 2, 1057: 2, 1055:
2, 1053: 2, 1048: 2, 1042: 2, 1041: 2, 1037: 2, 1032: 2, 1027: 2, 1026: 2, 1020: 2, 1018: 2, 1016:
2, 1014: 2, 1013: 2, 1008: 2, 1005: 2, 1002: 2, 997: 2, 982: 2, 980: 2, 979: 2, 976: 2, 975: 2, 972
: 2, 968: 2, 965: 2, 963: 2, 962: 2, 956: 2, 950: 2, 949: 2, 948: 2, 945: 2, 940: 2, 937: 2, 936:
2, 935: 2, 934: 2, 921: 2, 920: 2, 915: 2, 912: 2, 910: 2, 907: 2, 906: 2, 902: 2, 901: 2, 898: 2,
896: 2, 894: 2, 892: 2, 891: 2, 890: 2, 881: 2, 879: 2, 871: 2, 870: 2, 867: 2, 865: 2, 858: 2,
849: 2, 840: 2, 837: 2, 825: 2, 820: 2, 814: 2, 812: 2, 810: 2, 807: 2, 799: 2, 796: 2, 794: 2,
792: 2, 790: 2, 789: 2, 784: 2, 783: 2, 782: 2, 776: 2, 774: 2, 771: 2, 768: 2, 765: 2, 762: 2,
761: 2, 758: 2, 755: 2, 751: 2, 748: 2, 744: 2, 739: 2, 733: 2, 726: 2, 719: 2, 716: 2, 712: 2,
708: 2, 697: 2, 687: 2, 683: 2, 674: 2, 672: 2, 671: 2, 670: 2, 669: 2, 666: 2, 664: 2, 663: 2,
662: 2, 660: 2, 657: 2, 656: 2, 655: 2, 642: 2, 633: 2, 629: 2, 627: 2, 626: 2, 622: 2, 618: 2,
612: 2, 609: 2, 608: 2, 604: 2, 600: 2, 593: 2, 583: 2, 582: 2, 580: 2, 576: 2, 569: 2, 568: 2,
551: 2, 535: 2, 533: 2, 523: 2, 521: 2, 512: 2, 504: 2, 483: 2, 477: 2, 476: 2, 458: 2, 441: 2,
365: 2, 335: 2, 148460: 1, 117452: 1, 79942: 1, 66501: 1, 66369: 1, 66288: 1, 66228: 1, 62456: 1,
61944: 1, 54208: 1, 52598: 1, 48869: 1, 48082: 1, 46716: 1, 46179: 1, 42564: 1, 42490: 1, 42035: 1
, 41944: 1, 41529: 1, 41310: 1, 40177: 1, 39915: 1, 38124: 1, 37841: 1, 37445: 1, 37201: 1, 35819:
1, 35552: 1, 33830: 1, 33374: 1, 33084: 1, 32859: 1, 32563: 1, 32413: 1, 30967: 1, 29087: 1, 29068
: 1, 27865: 1, 26379: 1, 25746: 1, 25664: 1, 25487: 1, 25112: 1, 24387: 1, 24365: 1, 24224: 1, 240
89: 1, 24026: 1, 23736: 1, 23613: 1, 23051: 1, 22899: 1, 21815: 1, 21473: 1, 21407: 1, 21358: 1, 2
1333: 1, 21219: 1, 21007: 1, 20356: 1, 20316: 1, 20253: 1, 20130: 1, 20009: 1, 19786: 1, 19551: 1,
19338: 1, 19162: 1, 18884: 1, 18658: 1, 18498: 1, 18452: 1, 18374: 1, 18246: 1, 18165: 1, 17972: 1
, 17931: 1, 17855: 1, 17841: 1, 17620: 1, 17592: 1, 17562: 1, 17407: 1, 17287: 1, 17239: 1, 17192:
1, 17098: 1, 17045: 1, 17038: 1, 16941: 1, 16727: 1, 16619: 1, 16433: 1, 16323: 1, 16267: 1, 16186
: 1, 15955: 1, 15863: 1, 15848: 1, 15773: 1, 15699: 1, 15500: 1, 15363: 1, 15293: 1, 15269: 1, 152
25: 1, 15114: 1, 15052: 1, 14870: 1, 14748: 1, 14688: 1, 14661: 1, 14623: 1, 14621: 1, 14598: 1, 1
4321: 1, 14317: 1, 14198: 1, 14124: 1, 14078: 1, 13675: 1, 13470: 1, 13430: 1, 13420: 1, 13382: 1,
13333: 1, 13277: 1, 13219: 1, 13215: 1, 13158: 1, 12965: 1, 12930: 1, 12922: 1, 12900: 1, 12851: 1

, 12770: 1, 12610: 1, 12609: 1, 12562: 1, 12523: 1, 12490: 1, 12262: 1, 12259: 1, 12255: 1, 12251:
1, 12226: 1, 12217: 1, 12178: 1, 12120: 1, 12106: 1, 12071: 1, 12067: 1, 12057: 1, 12042: 1, 12035
: 1, 12002: 1, 11966: 1, 11951: 1, 11945: 1, 11889: 1, 11858: 1, 11856: 1, 11790: 1, 11723: 1, 117
00: 1, 11666: 1, 11585: 1, 11579: 1, 11537: 1, 11496: 1, 11401: 1, 11400: 1, 11399: 1, 11384: 1, 1
1260: 1, 11167: 1, 11154: 1, 11056: 1, 11001: 1, 10950: 1, 10867: 1, 10762: 1, 10647: 1, 10632: 1,
10618: 1, 10588: 1, 10472: 1, 10461: 1, 10337: 1, 10333: 1, 10185: 1, 10153: 1, 10124: 1, 10104: 1
, 10103: 1, 10078: 1, 10074: 1, 10017: 1, 9977: 1, 9914: 1, 9859: 1, 9856: 1, 9834: 1, 9790: 1, 978
2: 1, 9712: 1, 9689: 1, 9672: 1, 9599: 1, 9579: 1, 9489: 1, 9439: 1, 9436: 1, 9423: 1, 9420: 1, 941
8: 1, 9412: 1, 9369: 1, 9331: 1, 9310: 1, 9295: 1, 9267: 1, 9240: 1, 9098: 1, 9095: 1, 9084: 1, 902
6: 1, 8994: 1, 8959: 1, 8958: 1, 8955: 1, 8908: 1, 8898: 1, 8851: 1, 8848: 1, 8836: 1, 8829: 1, 880
3: 1, 8761: 1, 8713: 1, 8700: 1, 8610: 1, 8605: 1, 8590: 1, 8584: 1, 8494: 1, 8469: 1, 8455: 1, 844
7: 1, 8389: 1, 8383: 1, 8301: 1, 8259: 1, 8257: 1, 8220: 1, 8211: 1, 8181: 1, 8144: 1, 8118: 1, 811
4: 1, 8097: 1, 8033: 1, 8009: 1, 7998: 1, 7995: 1, 7979: 1, 7978: 1, 7928: 1, 7892: 1, 7884: 1, 783
4: 1, 7833: 1, 7820: 1, 7808: 1, 7783: 1, 7781: 1, 7759: 1, 7731: 1, 7730: 1, 7705: 1, 7682: 1, 757
8: 1, 7513: 1, 7507: 1, 7505: 1, 7503: 1, 7488: 1, 7476: 1, 7473: 1, 7455: 1, 7446: 1, 7440: 1, 742
8: 1, 7413: 1, 7380: 1, 7360: 1, 7336: 1, 7297: 1, 7246: 1, 7210: 1, 7196: 1, 7156: 1, 7155: 1, 715
3: 1, 7151: 1, 7147: 1, 7107: 1, 7101: 1, 7081: 1, 7071: 1, 7063: 1, 7039: 1, 7027: 1, 7025: 1, 702
4: 1, 6979: 1, 6962: 1, 6957: 1, 6953: 1, 6946: 1, 6932: 1, 6892: 1, 6877: 1, 6876: 1, 6870: 1, 682
7: 1, 6814: 1, 6805: 1, 6796: 1, 6795: 1, 6794: 1, 6784: 1, 6773: 1, 6709: 1, 6649: 1, 6629: 1, 661
8: 1, 6610: 1, 6595: 1, 6563: 1, 6562: 1, 6556: 1, 6546: 1, 6544: 1, 6525: 1, 6506: 1, 6470: 1, 645
4: 1, 6430: 1, 6419: 1, 6409: 1, 6383: 1, 6354: 1, 6331: 1, 6322: 1, 6311: 1, 6293: 1, 6288: 1, 625
5: 1, 6245: 1, 6243: 1, 6235: 1, 6225: 1, 6224: 1, 6217: 1, 6203: 1, 6192: 1, 6188: 1, 6177: 1, 617
5: 1, 6168: 1, 6167: 1, 6156: 1, 6143: 1, 6141: 1, 6109: 1, 6102: 1, 6072: 1, 6056: 1, 6047: 1, 603
5: 1, 6023: 1, 6016: 1, 6013: 1, 5994: 1, 5975: 1, 5958: 1, 5937: 1, 5916: 1, 5912: 1, 5900: 1, 589
1: 1, 5873: 1, 5859: 1, 5839: 1, 5794: 1, 5771: 1, 5761: 1, 5752: 1, 5749: 1, 5745: 1, 5727: 1, 572
4: 1, 5715: 1, 5707: 1, 5705: 1, 5697: 1, 5670: 1, 5648: 1, 5636: 1, 5612: 1, 5598: 1, 5595: 1, 559
3: 1, 5591: 1, 5590: 1, 5587: 1, 5582: 1, 5572: 1, 5568: 1, 5511: 1, 5499: 1, 5496: 1, 5492: 1, 548
9: 1, 5469: 1, 5452: 1, 5445: 1, 5444: 1, 5422: 1, 5420: 1, 5410: 1, 5402: 1, 5382: 1, 5374: 1, 535
3: 1, 5331: 1, 5326: 1, 5324: 1, 5309: 1, 5298: 1, 5290: 1, 5288: 1, 5282: 1, 5274: 1, 5268: 1, 524
8: 1, 5244: 1, 5227: 1, 5205: 1, 5181: 1, 5176: 1, 5159: 1, 5158: 1, 5138: 1, 5131: 1, 5130: 1, 508
3: 1, 5064: 1, 5057: 1, 5050: 1, 5041: 1, 5013: 1, 5005: 1, 4994: 1, 4986: 1, 4980: 1, 4963: 1, 495
9: 1, 4950: 1, 4947: 1, 4941: 1, 4938: 1, 4937: 1, 4928: 1, 4923: 1, 4915: 1, 4906: 1, 4904: 1, 490
1: 1, 4895: 1, 4890: 1, 4888: 1, 4875: 1, 4874: 1, 4861: 1, 4858: 1, 4855: 1, 4843: 1, 4824: 1, 481
5: 1, 4790: 1, 4771: 1, 4760: 1, 4750: 1, 4747: 1, 4701: 1, 4699: 1, 4676: 1, 4674: 1, 4665: 1, 465
9: 1, 4648: 1, 4638: 1, 4637: 1, 4635: 1, 4616: 1, 4607: 1, 4596: 1, 4591: 1, 4583: 1, 4582: 1, 455
8: 1, 4522: 1, 4520: 1, 4514: 1, 4507: 1, 4496: 1, 4488: 1, 4485: 1, 4470: 1, 4467: 1, 4464: 1, 446
2: 1, 4461: 1, 4459: 1, 4451: 1, 4450: 1, 4405: 1, 4404: 1, 4397: 1, 4392: 1, 4390: 1, 4389: 1, 438
6: 1, 4370: 1, 4351: 1, 4350: 1, 4345: 1, 4343: 1, 4340: 1, 4323: 1, 4317: 1, 4316: 1, 4306: 1, 429
4: 1, 4293: 1, 4288: 1, 4285: 1, 4257: 1, 4252: 1, 4234: 1, 4226: 1, 4215: 1, 4204: 1, 4194: 1, 418
7: 1, 4184: 1, 4182: 1, 4171: 1, 4157: 1, 4148: 1, 4142: 1, 4138: 1, 4133: 1, 4128: 1, 4119: 1, 411
1: 1, 4102: 1, 4101: 1, 4100: 1, 4094: 1, 4087: 1, 4080: 1, 4069: 1, 4067: 1, 4060: 1, 4059: 1, 405
4: 1, 4048: 1, 4047: 1, 4046: 1, 4045: 1, 4043: 1, 4041: 1, 4039: 1, 4037: 1, 4032: 1, 4026: 1, 402
5: 1, 4017: 1, 4001: 1, 3987: 1, 3974: 1, 3958: 1, 3947: 1, 3946: 1, 3942: 1, 3941: 1, 3938: 1, 393
3: 1, 3932: 1, 3927: 1, 3915: 1, 3910: 1, 3905: 1, 3899: 1, 3879: 1, 3876: 1, 3872: 1, 3870: 1, 386
4: 1, 3858: 1, 3857: 1, 3855: 1, 3853: 1, 3837: 1, 3834: 1, 3820: 1, 3813: 1, 3812: 1, 3802: 1, 379
9: 1, 3797: 1, 3778: 1, 3771: 1, 3766: 1, 3765: 1, 3762: 1, 3758: 1, 3757: 1, 3752: 1, 3749: 1, 372
9: 1, 3726: 1, 3711: 1, 3708: 1, 3696: 1, 3691: 1, 3690: 1, 3689: 1, 3684: 1, 3682: 1, 3677: 1, 366
3: 1, 3660: 1, 3654: 1, 3650: 1, 3645: 1, 3640: 1, 3638: 1, 3637: 1, 3631: 1, 3628: 1, 3627: 1, 362
0: 1, 3615: 1, 3609: 1, 3595: 1, 3585: 1, 3582: 1, 3577: 1, 3574: 1, 3568: 1, 3565: 1, 3563: 1, 355
9: 1, 3558: 1, 3553: 1, 3551: 1, 3544: 1, 3539: 1, 3538: 1, 3534: 1, 3532: 1, 3520: 1, 3518: 1, 351
7: 1, 3516: 1, 3510: 1, 3508: 1, 3505: 1, 3498: 1, 3497: 1, 3492: 1, 3477: 1, 3469: 1, 3466: 1, 345
7: 1, 3456: 1, 3454: 1, 3449: 1, 3446: 1, 3443: 1, 3433: 1, 3431: 1, 3430: 1, 3423: 1, 3422: 1, 341
6: 1, 3410: 1, 3407: 1, 3404: 1, 3394: 1, 3390: 1, 3387: 1, 3384: 1, 3383: 1, 3376: 1, 3370: 1, 336
8: 1, 3355: 1, 3354: 1, 3349: 1, 3347: 1, 3346: 1, 3345: 1, 3344: 1, 3342: 1, 3336: 1, 3334: 1, 333
0: 1, 3326: 1, 3318: 1, 3317: 1, 3315: 1, 3314: 1, 3307: 1, 3306: 1, 3298: 1, 3294: 1, 3292: 1, 328
6: 1, 3284: 1, 3280: 1, 3279: 1, 3268: 1, 3265: 1, 3259: 1, 3254: 1, 3253: 1, 3252: 1, 3247: 1, 324
4: 1, 3238: 1, 3233: 1, 3221: 1, 3219: 1, 3214: 1, 3210: 1, 3205: 1, 3202: 1, 3192: 1, 3190: 1, 318
0: 1, 3177: 1, 3176: 1, 3170: 1, 3156: 1, 3155: 1, 3146: 1, 3137: 1, 3136: 1, 3131: 1, 3128: 1, 312
5: 1, 3124: 1, 3115: 1, 3110: 1, 3103: 1, 3093: 1, 3089: 1, 3082: 1, 3079: 1, 3076: 1, 3068: 1, 306
6: 1, 3065: 1, 3062: 1, 3061: 1, 3053: 1, 3051: 1, 3049: 1, 3045: 1, 3041: 1, 3039: 1, 3037: 1, 303
4: 1, 3031: 1, 3024: 1, 3023: 1, 3020: 1, 3014: 1, 3002: 1, 2992: 1, 2989: 1, 2980: 1, 2979: 1, 297
1: 1, 2969: 1, 2963: 1, 2959: 1, 2952: 1, 2950: 1, 2946: 1, 2943: 1, 2928: 1, 2927: 1, 2924: 1, 291
3: 1, 2912: 1, 2910: 1, 2909: 1, 2906: 1, 2905: 1, 2904: 1, 2900: 1, 2895: 1, 2893: 1, 2891: 1, 288
9: 1, 2886: 1, 2879: 1, 2877: 1, 2876: 1, 2858: 1, 2852: 1, 2848: 1, 2844: 1, 2838: 1, 2837: 1, 283
4: 1, 2832: 1, 2830: 1, 2823: 1, 2821: 1, 2820: 1, 2815: 1, 2810: 1, 2808: 1, 2807: 1, 2804: 1, 279
0: 1, 2789: 1, 2774: 1, 2762: 1, 2759: 1, 2748: 1, 2744: 1, 2736: 1, 2729: 1, 2721: 1, 2719: 1, 271
0: 1, 2708: 1, 2702: 1, 2700: 1, 2699: 1, 2696: 1, 2693: 1, 2671: 1, 2667: 1, 2666: 1, 2659: 1, 265
7: 1, 2639: 1, 2635: 1, 2626: 1, 2625: 1, 2624: 1, 2620: 1, 2618: 1, 2616: 1, 2614: 1, 2613: 1, 261
2: 1, 2610: 1, 2606: 1, 2602: 1, 2601: 1, 2599: 1, 2593: 1, 2587: 1, 2584: 1, 2574: 1, 2561: 1, 256
0: 1, 2557: 1, 2555: 1, 2553: 1, 2551: 1, 2550: 1, 2542: 1, 2537: 1, 2529: 1, 2528: 1, 2522: 1, 251
8: 1, 2512: 1, 2511: 1, 2510: 1, 2508: 1, 2505: 1, 2503: 1, 2502: 1, 2499: 1, 2496: 1, 2495: 1, 249
0: 1, 2481: 1, 2478: 1, 2477: 1, 2471: 1, 2467: 1, 2466: 1, 2464: 1, 2462: 1, 2447: 1, 2446: 1, 243
9: 1, 2436: 1, 2433: 1, 2431: 1, 2418: 1, 2415: 1, 2413: 1, 2410: 1, 2406: 1, 2400: 1, 2394: 1, 238
9: 1, 2385: 1, 2384: 1, 2383: 1, 2376: 1, 2375: 1, 2374: 1, 2371: 1, 2367: 1, 2365: 1, 2363: 1, 235
9: 1, 2357: 1, 2355: 1, 2352: 1, 2349: 1, 2348: 1, 2345: 1, 2336: 1, 2329: 1, 2326: 1, 2317: 1, 231
5: 1, 2311: 1, 2307: 1, 2303: 1, 2301: 1, 2299: 1, 2298: 1, 2296: 1, 2295: 1, 2289: 1, 2285: 1, 228
2: 1, 2281: 1, 2278: 1, 2273: 1, 2272: 1, 2271: 1, 2262: 1, 2261: 1, 2254: 1, 2251: 1, 2250: 1, 224

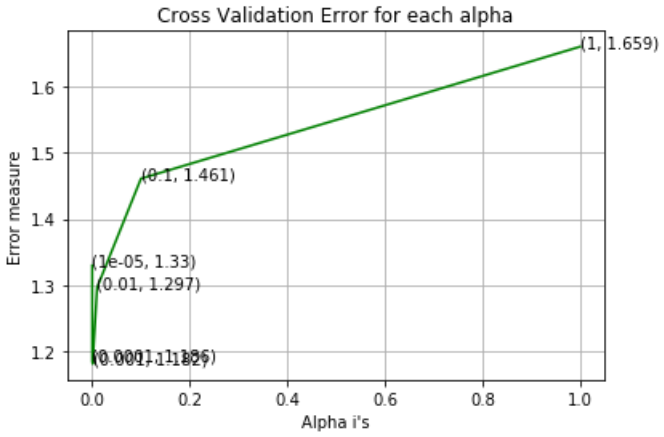
```
8: 1, 2247: 1, 2243: 1, 2241: 1, 2237: 1, 2229: 1, 2227: 1, 2220: 1, 2217: 1, 2215: 1, 2210: 1, 220
8: 1, 2207: 1, 2206: 1, 2202: 1, 2201: 1, 2196: 1, 2195: 1, 2191: 1, 2186: 1, 2184: 1, 2172: 1, 217
0: 1, 2166: 1, 2162: 1, 2159: 1, 2158: 1, 2157: 1, 2156: 1, 2154: 1, 2152: 1, 2146: 1, 2145: 1, 214
3: 1, 2141: 1, 2139: 1, 2137: 1, 2133: 1, 2130: 1, 2127: 1, 2126: 1, 2121: 1, 2120: 1, 2119: 1, 211
7: 1, 2116: 1, 2111: 1, 2109: 1, 2104: 1, 2102: 1, 2096: 1, 2091: 1, 2089: 1, 2086: 1, 2082: 1, 207
9: 1, 2078: 1, 2073: 1, 2071: 1, 2068: 1, 2067: 1, 2066: 1, 2063: 1, 2059: 1, 2058: 1, 2054: 1, 205
1: 1, 2045: 1, 2035: 1, 2034: 1, 2030: 1, 2023: 1, 2022: 1, 2020: 1, 2016: 1, 2014: 1, 2010: 1, 200
5: 1, 2002: 1, 1993: 1, 1991: 1, 1989: 1, 1981: 1, 1980: 1, 1975: 1, 1972: 1, 1971: 1, 1968: 1, 196
7: 1, 1966: 1, 1963: 1, 1960: 1, 1954: 1, 1947: 1, 1946: 1, 1945: 1, 1942: 1, 1941: 1, 1940: 1, 193
7: 1, 1932: 1, 1929: 1, 1928: 1, 1927: 1, 1921: 1, 1920: 1, 1919: 1, 1918: 1, 1916: 1, 1915: 1, 191
4: 1, 1913: 1, 1909: 1, 1908: 1, 1905: 1, 1902: 1, 1901: 1, 1899: 1, 1898: 1, 1897: 1, 1896: 1, 189
4: 1, 1893: 1, 1892: 1, 1887: 1, 1879: 1, 1878: 1, 1875: 1, 1871: 1, 1869: 1, 1866: 1, 1864: 1, 186
3: 1, 1862: 1, 1855: 1, 1853: 1, 1849: 1, 1846: 1, 1845: 1, 1841: 1, 1835: 1, 1831: 1, 1830: 1, 182
9: 1, 1825: 1, 1824: 1, 1822: 1, 1820: 1, 1819: 1, 1818: 1, 1808: 1, 1805: 1, 1804: 1, 1803: 1, 179
3: 1, 1791: 1, 1789: 1, 1787: 1, 1785: 1, 1784: 1, 1783: 1, 1782: 1, 1779: 1, 1773: 1, 1771: 1, 177
0: 1, 1769: 1, 1768: 1, 1767: 1, 1766: 1, 1760: 1, 1759: 1, 1757: 1, 1754: 1, 1752: 1, 1751: 1, 174
8: 1, 1746: 1, 1743: 1, 1740: 1, 1738: 1, 1737: 1, 1733: 1, 1727: 1, 1726: 1, 1723: 1, 1721: 1, 171
7: 1, 1709: 1, 1706: 1, 1700: 1, 1699: 1, 1694: 1, 1691: 1, 1690: 1, 1686: 1, 1683: 1, 1680: 1, 167
8: 1, 1677: 1, 1676: 1, 1674: 1, 1673: 1, 1672: 1, 1671: 1, 1666: 1, 1665: 1, 1664: 1, 1658: 1, 165
6: 1, 1655: 1, 1652: 1, 1651: 1, 1648: 1, 1647: 1, 1644: 1, 1642: 1, 1641: 1, 1640: 1, 1634: 1, 163
1: 1, 1629: 1, 1628: 1, 1622: 1, 1618: 1, 1616: 1, 1615: 1, 1612: 1, 1608: 1, 1607: 1, 1606: 1, 159
9: 1, 1590: 1, 1588: 1, 1587: 1, 1586: 1, 1575: 1, 1570: 1, 1569: 1, 1568: 1, 1567: 1, 1564: 1, 156
1: 1, 1560: 1, 1557: 1, 1555: 1, 1554: 1, 1551: 1, 1549: 1, 1547: 1, 1546: 1, 1545: 1, 1544: 1, 154
2: 1, 1540: 1, 1539: 1, 1536: 1, 1535: 1, 1534: 1, 1528: 1, 1526: 1, 1524: 1, 1520: 1, 1516: 1, 150
5: 1, 1502: 1, 1499: 1, 1496: 1, 1494: 1, 1489: 1, 1487: 1, 1486: 1, 1484: 1, 1476: 1, 1472: 1, 146
7: 1, 1463: 1, 1457: 1, 1454: 1, 1447: 1, 1446: 1, 1440: 1, 1434: 1, 1433: 1, 1432: 1, 1431: 1, 142
7: 1, 1426: 1, 1423: 1, 1422: 1, 1421: 1, 1417: 1, 1415: 1, 1414: 1, 1411: 1, 1406: 1, 1405: 1, 140
2: 1, 1401: 1, 1400: 1, 1397: 1, 1395: 1, 1394: 1, 1391: 1, 1390: 1, 1387: 1, 1385: 1, 1384: 1, 137
9: 1, 1376: 1, 1372: 1, 1369: 1, 1367: 1, 1364: 1, 1363: 1, 1360: 1, 1355: 1, 1352: 1, 1350: 1, 134
3: 1, 1333: 1, 1331: 1, 1326: 1, 1324: 1, 1322: 1, 1321: 1, 1320: 1, 1313: 1, 1311: 1, 1309: 1, 130
8: 1, 1307: 1, 1305: 1, 1303: 1, 1302: 1, 1300: 1, 1294: 1, 1292: 1, 1291: 1, 1290: 1, 1281: 1, 128
0: 1, 1277: 1, 1276: 1, 1270: 1, 1269: 1, 1267: 1, 1265: 1, 1263: 1, 1262: 1, 1259: 1, 1255: 1, 125
0: 1, 1248: 1, 1247: 1, 1244: 1, 1242: 1, 1241: 1, 1235: 1, 1233: 1, 1230: 1, 1228: 1, 1227: 1, 122
1: 1, 1217: 1, 1216: 1, 1212: 1, 1211: 1, 1209: 1, 1208: 1, 1202: 1, 1197: 1, 1195: 1, 1193: 1, 119
2: 1, 1191: 1, 1186: 1, 1184: 1, 1183: 1, 1181: 1, 1179: 1, 1173: 1, 1168: 1, 1166: 1, 1165: 1, 116
4: 1, 1162: 1, 1158: 1, 1151: 1, 1150: 1, 1149: 1, 1147: 1, 1145: 1, 1144: 1, 1143: 1, 1141: 1, 113
7: 1, 1131: 1, 1129: 1, 1123: 1, 1122: 1, 1117: 1, 1116: 1, 1112: 1, 1106: 1, 1105: 1, 1103: 1, 110
1: 1, 1100: 1, 1099: 1, 1098: 1, 1097: 1, 1096: 1, 1094: 1, 1093: 1, 1092: 1, 1090: 1, 1088: 1, 108
3: 1, 1082: 1, 1079: 1, 1077: 1, 1075: 1, 1073: 1, 1071: 1, 1070: 1, 1063: 1, 1061: 1, 1060: 1, 105
9: 1, 1058: 1, 1054: 1, 1052: 1, 1051: 1, 1049: 1, 1045: 1, 1040: 1, 1038: 1, 1036: 1, 1033: 1, 103
0: 1, 1029: 1, 1028: 1, 1024: 1, 1021: 1, 1017: 1, 1015: 1, 1012: 1, 1009: 1, 1006: 1, 1000: 1, 996
: 1, 993: 1, 992: 1, 991: 1, 986: 1, 984: 1, 981: 1, 978: 1, 974: 1, 971: 1, 969: 1, 967: 1, 961:
1, 954: 1, 951: 1, 946: 1, 942: 1, 941: 1, 938: 1, 932: 1, 929: 1, 923: 1, 922: 1, 916: 1, 911: 1,
905: 1, 900: 1, 895: 1, 889: 1, 887: 1, 876: 1, 875: 1, 873: 1, 869: 1, 862: 1, 856: 1, 855: 1,
852: 1, 838: 1, 836: 1, 833: 1, 832: 1, 829: 1, 827: 1, 819: 1, 813: 1, 811: 1, 808: 1, 803: 1,
801: 1, 793: 1, 791: 1, 787: 1, 786: 1, 779: 1, 769: 1, 766: 1, 763: 1, 753: 1, 749: 1, 738: 1,
737: 1, 732: 1, 730: 1, 728: 1, 727: 1, 713: 1, 711: 1, 706: 1, 704: 1, 696: 1, 695: 1, 690: 1,
689: 1, 682: 1, 679: 1, 659: 1, 647: 1, 645: 1, 641: 1, 596: 1, 588: 1, 585: 1, 579: 1, 573: 1,
567: 1, 561: 1, 557: 1, 553: 1, 530: 1, 486: 1, 484: 1})
```

In [48]:

```
# Training a simple LR model with Calibrated Classifier

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

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

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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_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: 1.329895889274032
For values of alpha =  0.0001 The log loss is: 1.1859995431340724
For values of alpha =  0.001 The log loss is: 1.1816334571051037
For values of alpha =  0.01 The log loss is: 1.2966899246580765
For values of alpha =  0.1 The log loss is: 1.460785675755051
For values of alpha =  1 The log loss is: 1.6594970984852093
```




```
For values of best alpha =  0.001 The train log loss is: 0.6610459690042393
For values of best alpha =  0.001 The cross validation log loss is: 1.1816334571051037
For values of best alpha =  0.001 The test log loss is: 1.1118017728174392
```

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

Ans. Yes, it seems like!

In [49]:

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

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

In [50]:

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

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

```
96.259 % of word of test data appeared in train data
98.575 % of word of Cross Validation appeared in train data
```

Observations

- By only using Text feature the log loss that we have obtained is 1.11.
- The achieved log loss is quite satisfactory given that it is obtained by using only 1 feature.
- Text feature is very important feature.

Applying Machine Learning Models

In [51]:

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

    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    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 [52]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)

    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [53]:

```
# For Naive Bayes to get the names of important features

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

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

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

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

Stacking the three types of features

In [54]:

```
# merging gene, variance and text features

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

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

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

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

train_gene_var_responseCoding =
np.hstack((train_gene_feature_responseCoding,train_variation_feature_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 [55]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55945)
(number of data points * number of features) in test data =  (665, 55945)
(number of data points * number of features) in cross validation data = (532, 55945)
```

In [56]:

```
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)
```

Base Line Model (Naive Bayes)

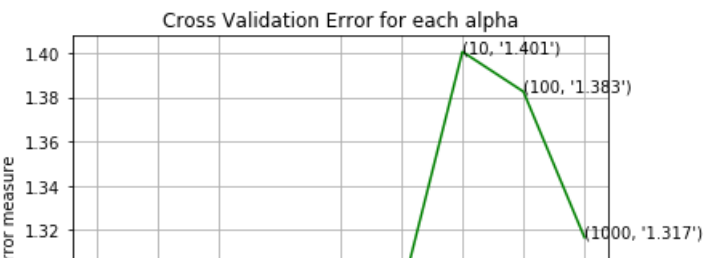
Base Line Model (Naive Bayes)

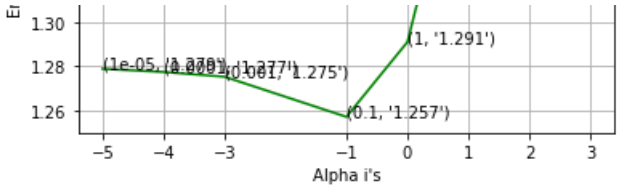
Hyper parameter tuning

In [57]:

```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []

for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha = i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')

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

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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best 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-05
Log Loss : 1.278789308662036
for alpha = 0.0001
Log Loss : 1.2773334660840903
for alpha = 0.001
Log Loss : 1.2751858062794428
for alpha = 0.1
Log Loss : 1.2570026607894458
for alpha = 1
Log Loss : 1.2909053900365035
for alpha = 10
Log Loss : 1.4006459403253997
for alpha = 100
Log Loss : 1.3825815823707301
for alpha = 1000
Log Loss : 1.3166146503891014
```






```
For values of best alpha =  0.1 The train log loss is: 0.8601148179488521
For values of best alpha =  0.1 The cross validation log loss is: 1.2570026607894458
For values of best alpha =  0.1 The test log loss is: 1.2706871237769615
```

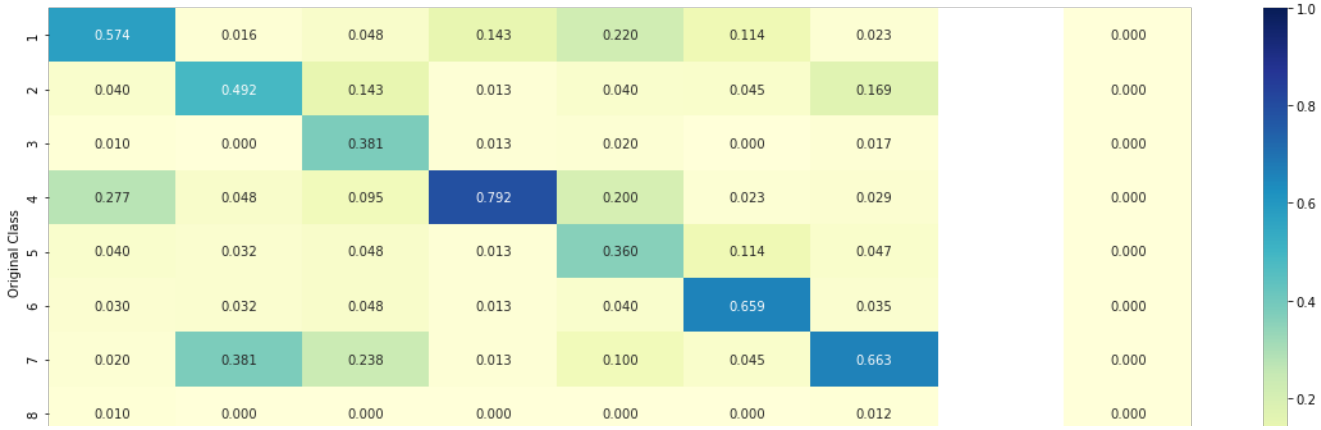
Testing the model with best hyper paramters

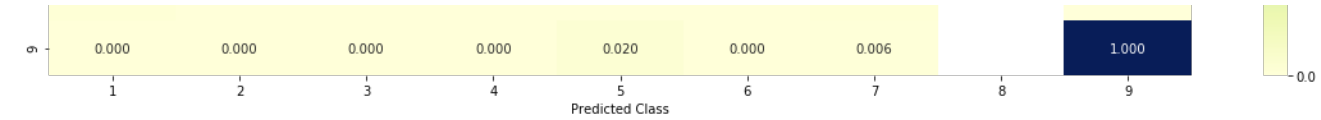
In [58]:

```
clf = MultinomialNB(alpha = alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)

sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)

print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv
_y))/cv_y.shape[0])

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

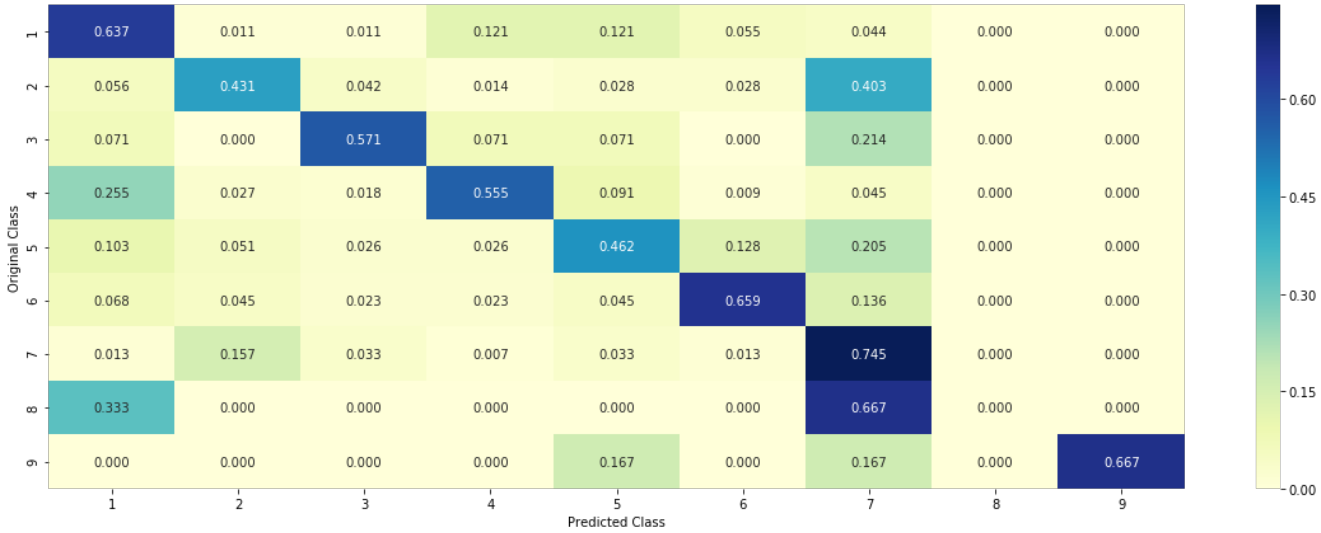
```
Log Loss : 1.2570026607894458
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```


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






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



Observations

- Naive bayes is used to act as a baseline model.
- The best alpha value is 0.1.
- The test log loss by Naive Bayes is 1.27.
- Number of misclassified points are 0.3928.

Feature Importance and Incorrectly classified point

In [59]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1099 0.0991 0.02   0.1291 0.4223 0.0445 0.164  0.0065 0.0047]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Feature Importance and Correctly classified point

In [60]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])

print("Predicted Class :", predicted_cls[0])
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0738 0.0668 0.0136 0.6666 0.0311 0.0299 0.1106 0.0044 0.0032]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

K Nearest Neighbour

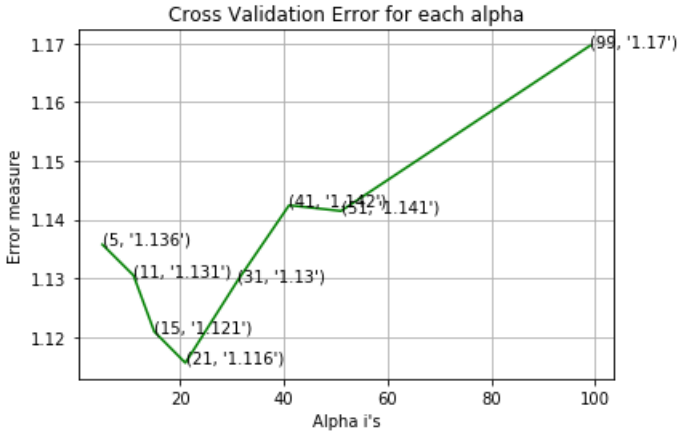
Hyper parameter tuning

In [61]:

```
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []

for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

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

```
for alpha = 5
Log Loss : 1.1357652321631344
for alpha = 11
Log Loss : 1.130503241033282
for alpha = 15
Log Loss : 1.1209391702665708
for alpha = 21
Log Loss : 1.1156120292159917
for alpha = 31
Log Loss : 1.1295146852427798
```

```
for alpha = 41
Log Loss : 1.142443217443858
for alpha = 51
Log Loss : 1.1414654470118355
for alpha = 99
Log Loss : 1.1695959172670027
```



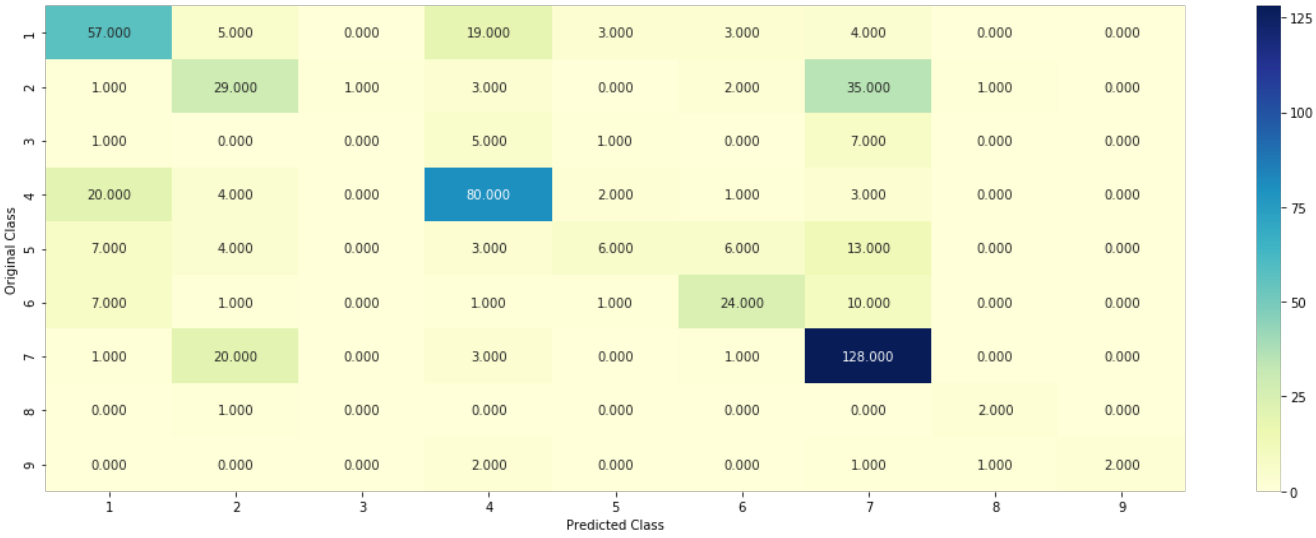

```
For values of best alpha =  21 The train log loss is: 0.7253976058071508
For values of best alpha =  21 The cross validation log loss is: 1.1156120292159917
For values of best alpha =  21 The test log loss is: 1.0704282254037798
```

Testing the model with best hyper paramters

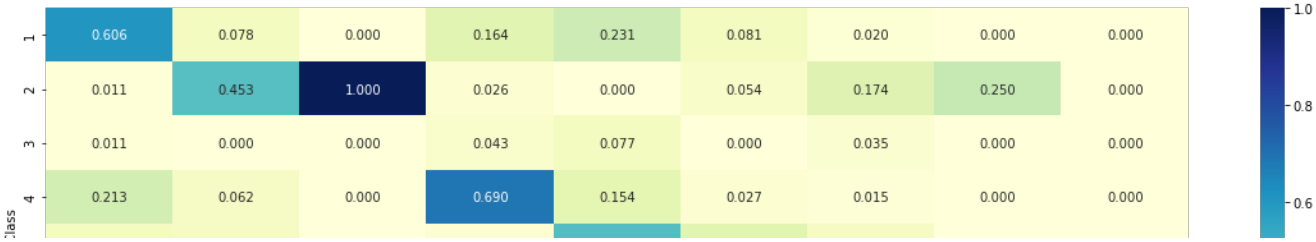
In [62]:

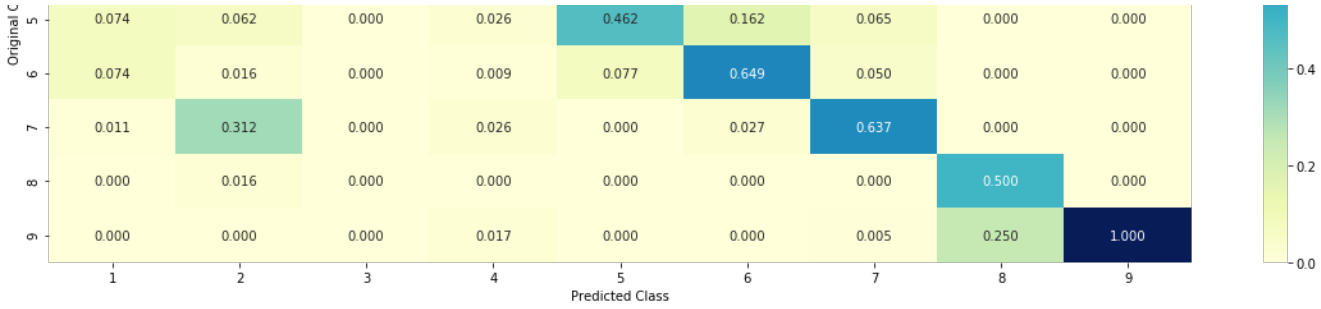
```
clf = KNeighborsClassifier(n_neighbors = alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.1156120292159917
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```

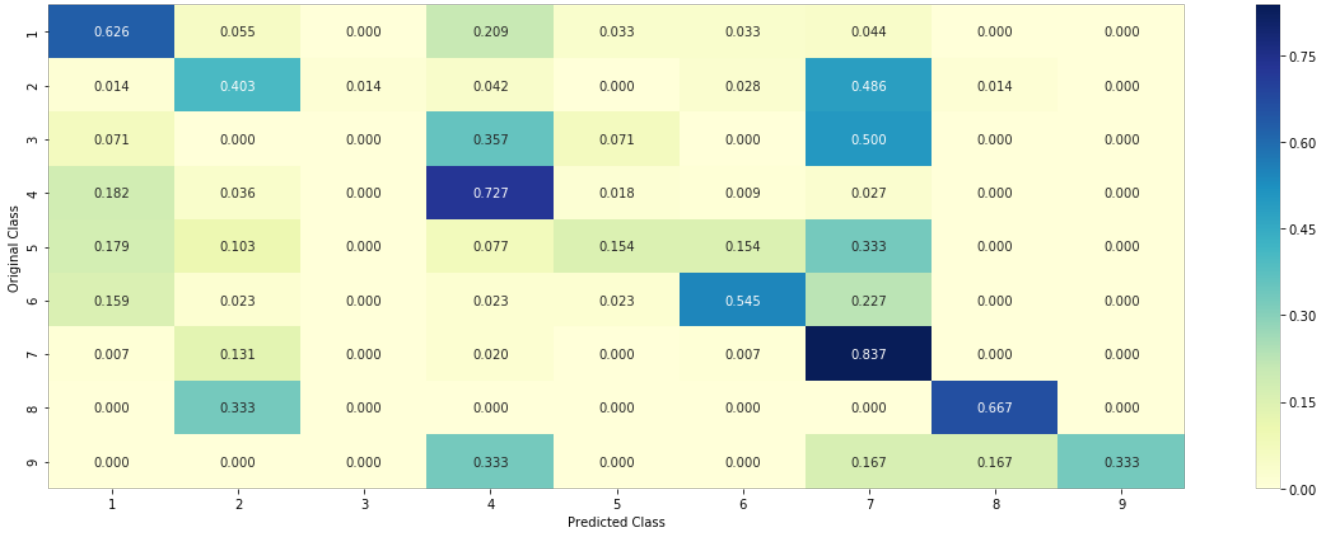



```
-------------------- Precision matrix (Column Sum=1) --------------------
```






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



Observations

- The best k value is 21.
- The test log loss by KNN is 1.07.
- Number of misclassified points are 0.3834.

Sample Query point -1

In [63]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))

print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])

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

```
Predicted Class : 7
Actual Class : 4
The  21  nearest neighbours of the test points belongs to classes [4 5 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
Fequency of nearest points : Counter({4: 19, 5: 2})
```

Sample Query Point-2

In [64]:

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

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])

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

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 21 and the nearest neighbours of the test points belongs to classes [4 4 4
4 4 4 4 4 4 4 1 4 4 4 4 4 1 1 4 1 1]
Fequency of nearest points : Counter({4: 16, 1: 5})
```

Logistic Regression with Class Balancing

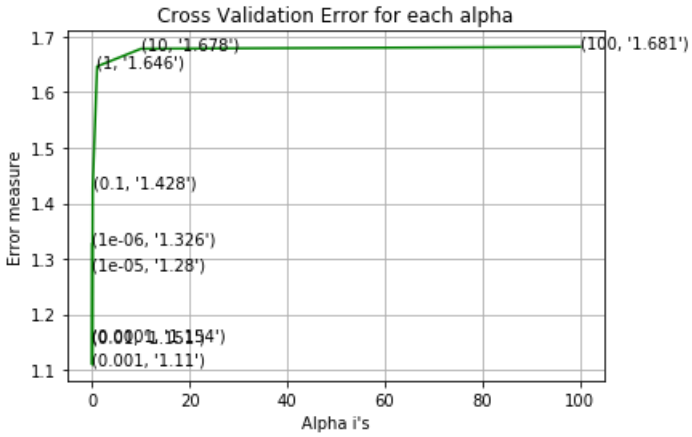
Hyper paramter tuning

In [65]:

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

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

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

for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))

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

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

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

```
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best 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 : 1.326432795336554
for alpha = 1e-05
Log Loss : 1.2798927652974355
for alpha = 0.0001
Log Loss : 1.1535032325228216
for alpha = 0.001
Log Loss : 1.1095056949197042
for alpha = 0.01
Log Loss : 1.1506878474795372
for alpha = 0.1
Log Loss : 1.4280019184853032
for alpha = 1
Log Loss : 1.6462182202823852
for alpha = 10
Log Loss : 1.6779805889841999
for alpha = 100
Log Loss : 1.6813973895784946
```



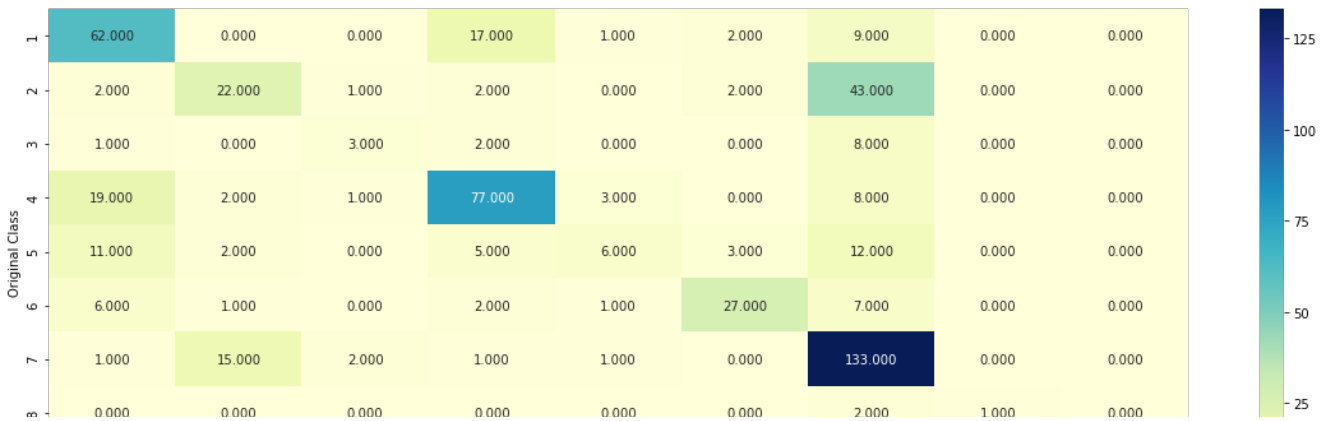

```
For values of best alpha =  0.001 The train log loss is: 0.5391990465545016
For values of best alpha =  0.001 The cross validation log loss is: 1.1095056949197042
For values of best alpha =  0.001 The test log loss is: 1.0431542345531812
```

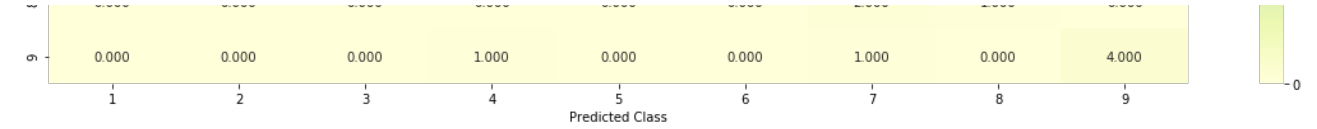
Testing the model with best hyper paramters

In [66]:

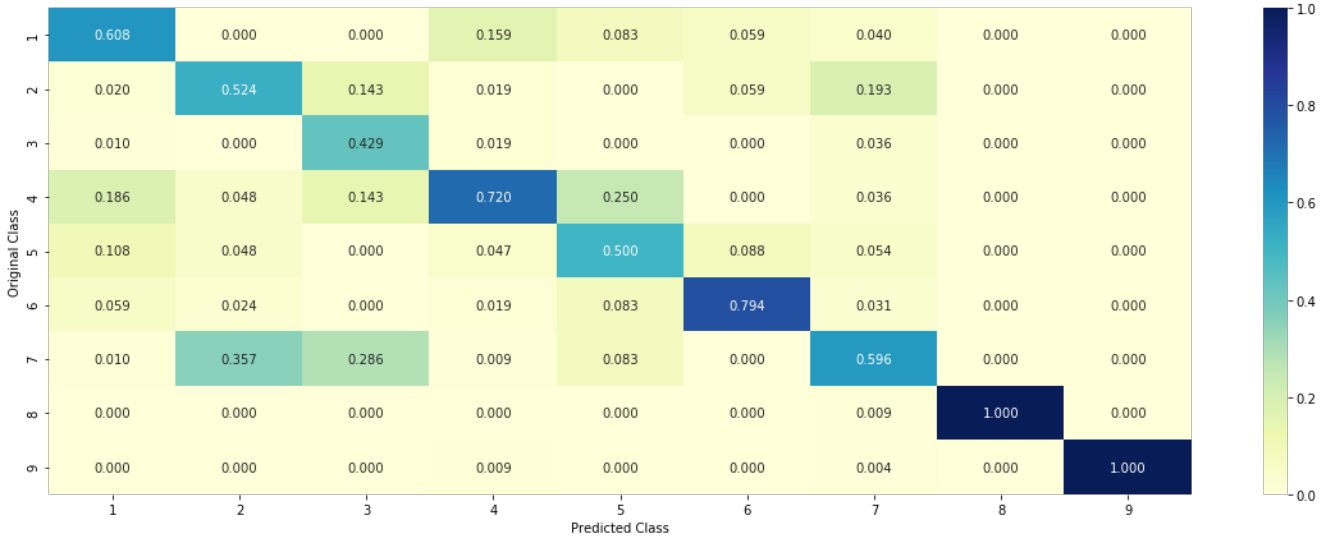
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.1095056949197042
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```

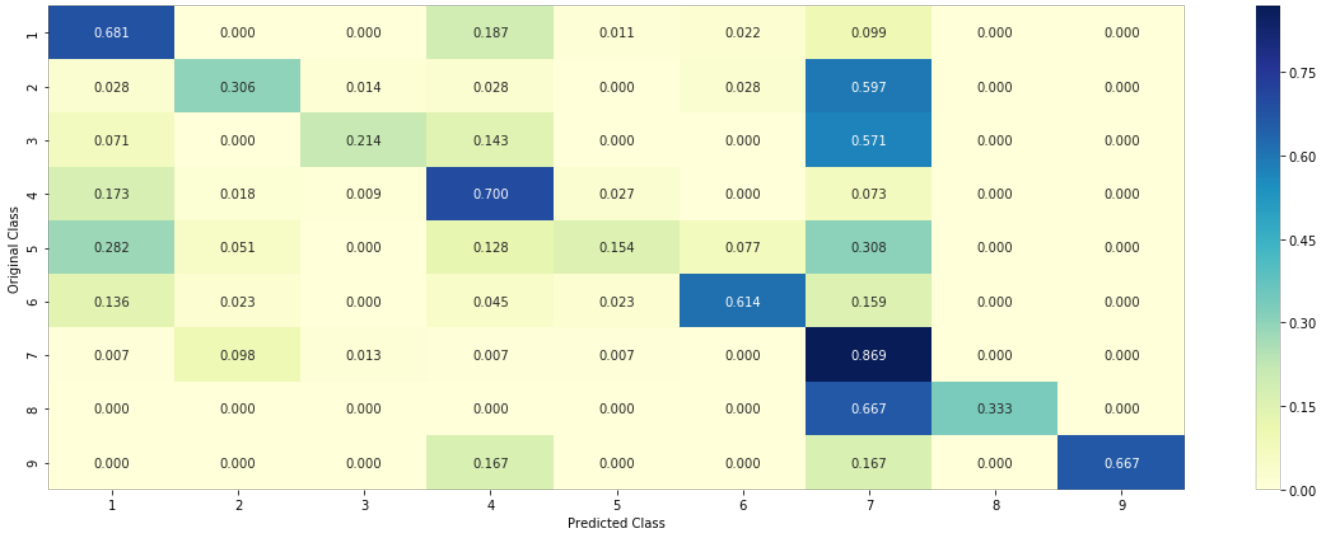




---------- Precision matrix (Columm Sum=1) ----------



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



Observations

- The best alpha value is 0.001.
- The test log loss by Logistic Regression with class balancing is 1.04.
- Number of misclassified points are 0.3703.

Feature Importance

In [67]:

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

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

Incorrectly Classified point

In [68]:

```
# from tabulate import tabulate

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0554 0.07   0.013  0.2576 0.5014 0.022  0.0665 0.007  0.0071]]
Actual Class : 4
--------------------------------------------------
252 Text feature [pyrimidines] present in test data point [True]
368 Text feature [s362l] present in test data point [True]
369 Text feature [p95s] present in test data point [True]
370 Text feature [ezviewtm] present in test data point [True]
371 Text feature [901g] present in test data point [True]
372 Text feature [cycloadenines] present in test data point [True]
373 Text feature [l1325f] present in test data point [True]
374 Text feature [f154l] present in test data point [True]
375 Text feature [d301n] present in test data point [True]
376 Text feature [pyrimidone] present in test data point [True]
377 Text feature [dermatoscopy] present in test data point [True]
378 Text feature [xp295be] present in test data point [True]
379 Text feature [y16x] present in test data point [True]
380 Text feature [292c] present in test data point [True]
381 Text feature [g230x] present in test data point [True]
382 Text feature [q110r] present in test data point [True]
389 Text feature [dipyrimidines] present in test data point [True]
Out of the top  500  features  17 are present in query point
```

Correctly Classified point

In [69]:

```
test_point_index = 100
no_feature = 500

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

```
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0163 0.0282 0.0065 0.8825 0.0146 0.0104 0.0335 0.0045 0.0036]]
Actual Class : 4
--------------------------------------------------
170 Text feature [activating] present in test data point [True]
188 Text feature [nonidet] present in test data point [True]
290 Text feature [destroyed] present in test data point [True]
329 Text feature [transforming] present in test data point [True]
380 Text feature [inhibitor] present in test data point [True]
382 Text feature [agar] present in test data point [True]
386 Text feature [potential] present in test data point [True]
476 Text feature [confer] present in test data point [True]
495 Text feature [constitutive] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

Logistic Regression Without Class balancing

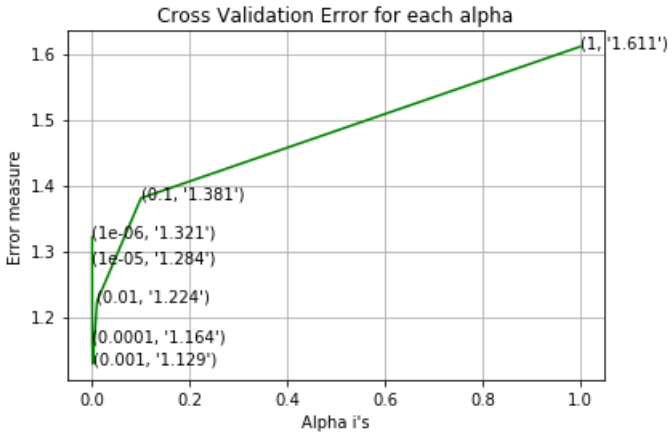
Hyper paramter tuning

In [70]:

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []

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

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

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best 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 : 1.3211661228932872
for alpha = 1e-05
Log Loss : 1.283642052262818
for alpha = 0.0001
```

```
Log Loss : 1.1636121845687155
for alpha = 0.001
Log Loss : 1.128608762507809
for alpha = 0.01
Log Loss : 1.223710593498229
for alpha = 0.1
Log Loss : 1.3805451890253693
for alpha = 1
Log Loss : 1.6107660073579224
```



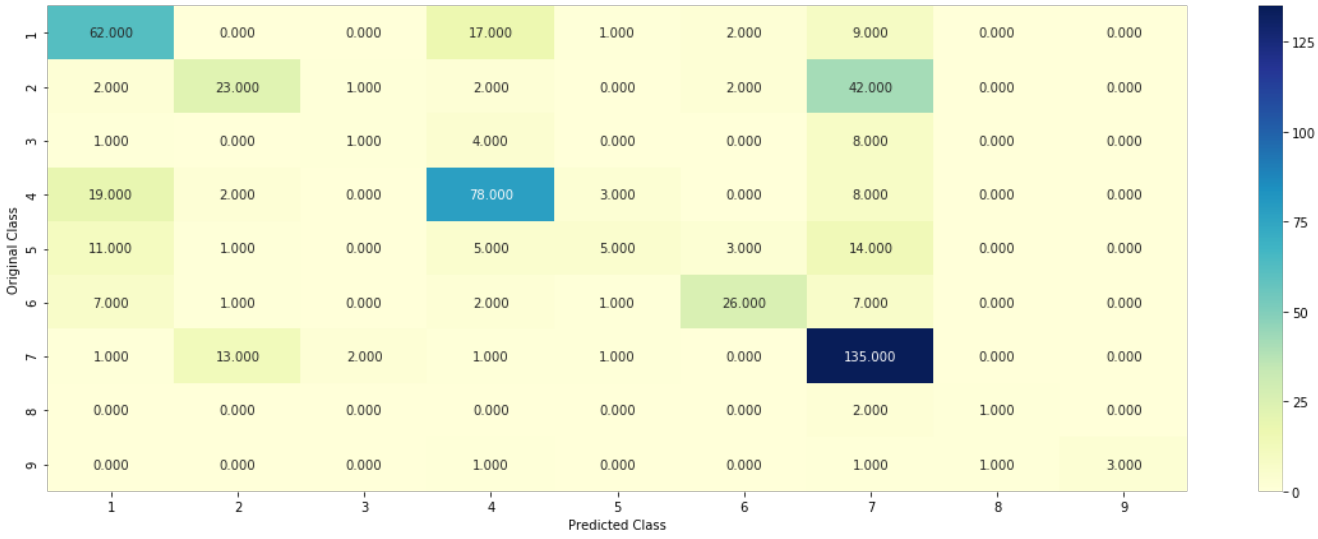

```
For values of best alpha =  0.001 The train log loss is: 0.5349039849019852
For values of best alpha =  0.001 The cross validation log loss is: 1.128608762507809
For values of best alpha =  0.001 The test log log loss is: 1.045002355161908
```

Testing model with best hyper parameters

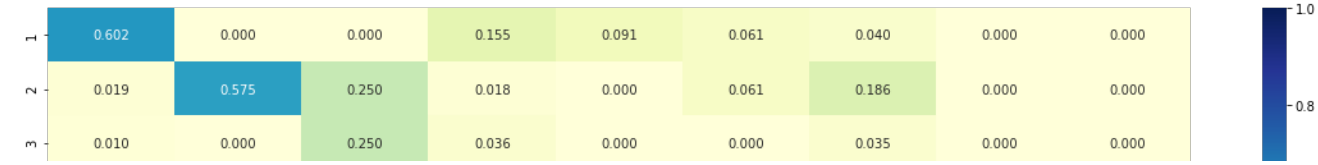
In [71]:

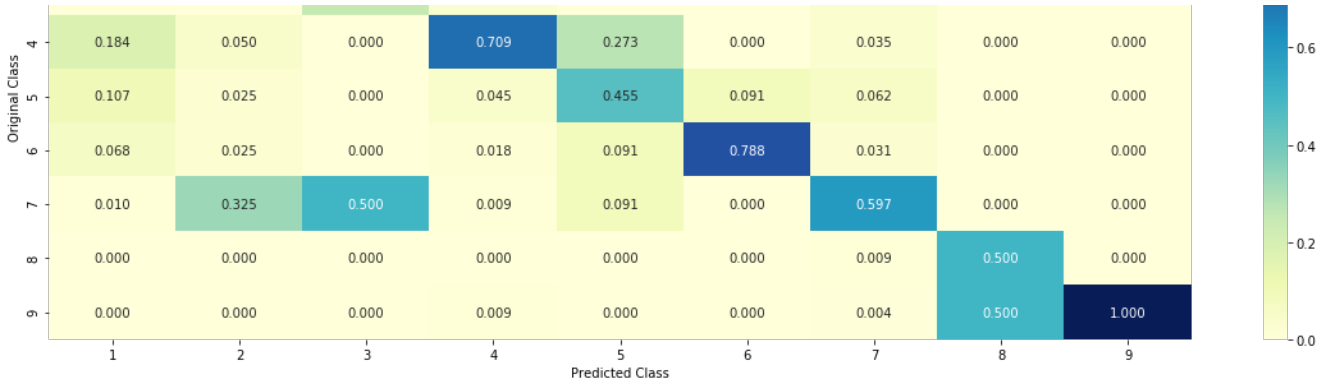
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.128608762507809
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

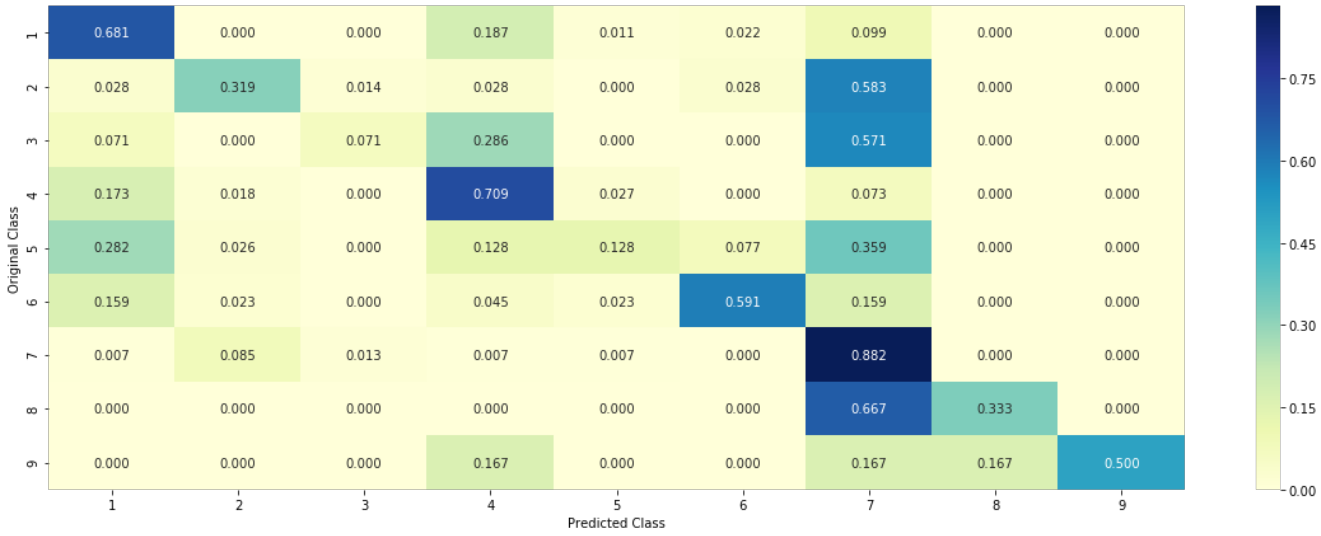



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Observations

- The best alpha value is 0.001.
- The test log loss by Logistic Regression without class balancing is 1.05.
- Number of misclassified points are 0.3721.
- As we can see that the difference between the test logloss and No. of misclassified points between LR with and without class balancing is not that much. Infact they are nearly the same!

Feature Importance and Incorrectly Classified point

In [72]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0557 0.0708 0.0149 0.2956 0.4639 0.0223 0.0653 0.0059 0.0057]]
Actual Class : 4
--------------------------------------------------
```

```
262 Text feature [pyrimidines] present in test data point [True]
375 Text feature [identified] present in test data point [True]
411 Text feature [identification] present in test data point [True]
429 Text feature [ezviewtm] present in test data point [True]
430 Text feature [q110r] present in test data point [True]
431 Text feature [292c] present in test data point [True]
432 Text feature [p95s] present in test data point [True]
433 Text feature [f154l] present in test data point [True]
434 Text feature [pyrimidone] present in test data point [True]
435 Text feature [g230x] present in test data point [True]
436 Text feature [cycloadenines] present in test data point [True]
437 Text feature [dermatoscopy] present in test data point [True]
438 Text feature [901g] present in test data point [True]
439 Text feature [y16x] present in test data point [True]
440 Text feature [l325f] present in test data point [True]
441 Text feature [d301n] present in test data point [True]
442 Text feature [xp295be] present in test data point [True]
443 Text feature [s362l] present in test data point [True]
462 Text feature [dipyrimidines] present in test data point [True]
467 Text feature [1a] present in test data point [True]
Out of the top  500  features  20 are present in query point
```

Feature Importance and Correctly Classified point

In [73]:

```
test_point_index = 100
no_feature = 500

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0168 0.0287 0.0066 0.8819 0.0147 0.0101 0.0355 0.0036 0.0022]]
Actual Class : 4
--------------------------------------------------
310 Text feature [activating] present in test data point [True]
315 Text feature [nonidet] present in test data point [True]
389 Text feature [destroyed] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

Linear Support Vector Machines

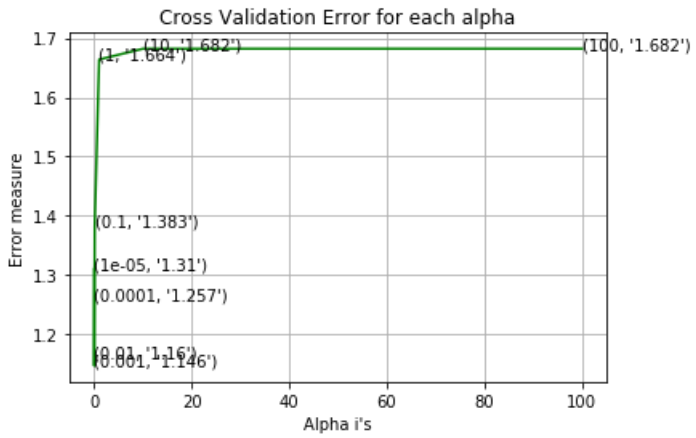
Hyper paramter tuning

In [74]:

```
alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []

for i in alpha:
    print("for C =", i)
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state
=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
```

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

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best 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 : 1.309993141366402
for C = 0.0001
Log Loss : 1.2573809143397818
for C = 0.001
Log Loss : 1.146413204671599
for C = 0.01
Log Loss : 1.1602749625278517
for C = 0.1
Log Loss : 1.382856169130563
for C = 1
Log Loss : 1.6637737731226172
for C = 10
Log Loss : 1.6819950886866328
for C = 100
Log Loss : 1.6819944440896757
```



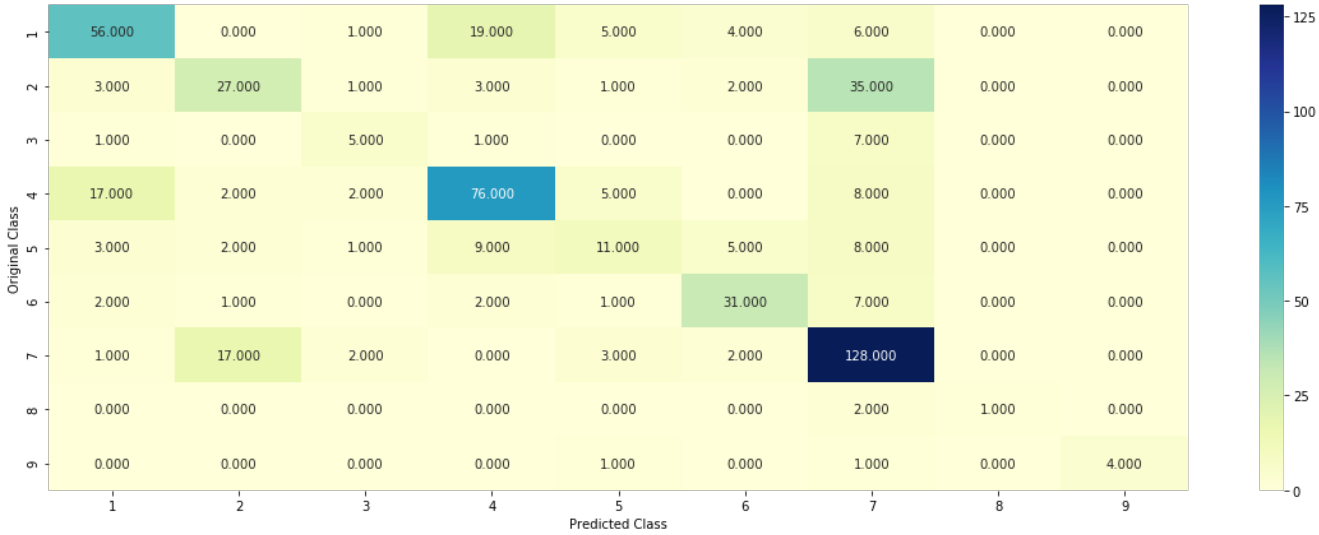

```
For values of best alpha =  0.001 The train log loss is: 0.5624840214507419
For values of best alpha =  0.001 The cross validation log loss is: 1.146413204671599
For values of best alpha =  0.001 The test log loss is: 1.1191231771652232
```

Testing model with best hyper parameters

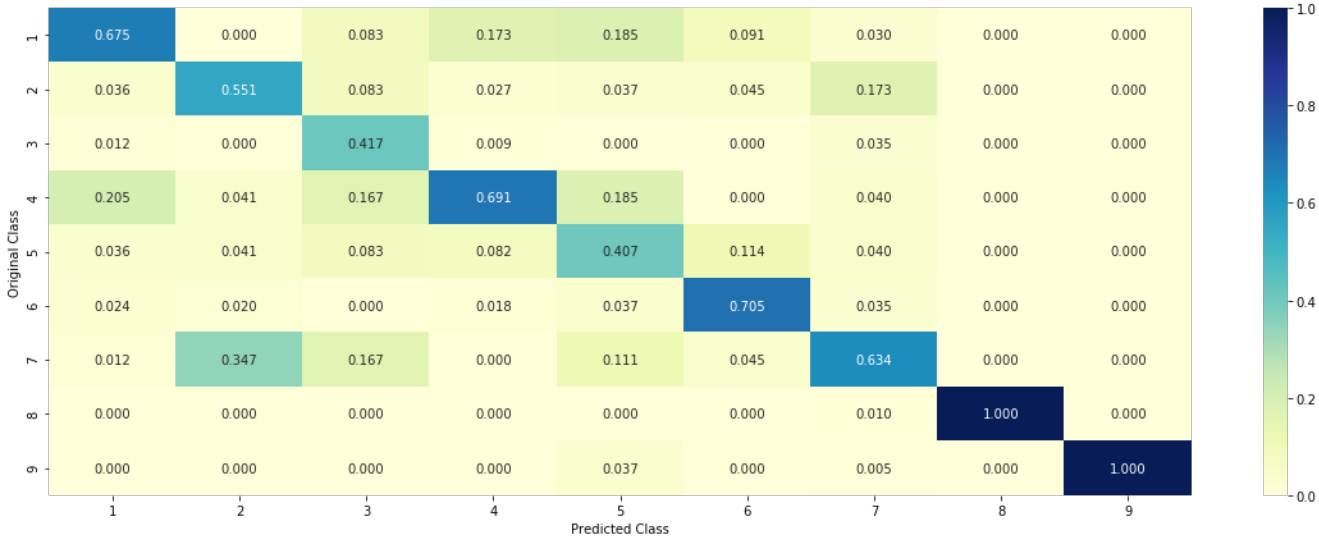
In [75]:

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

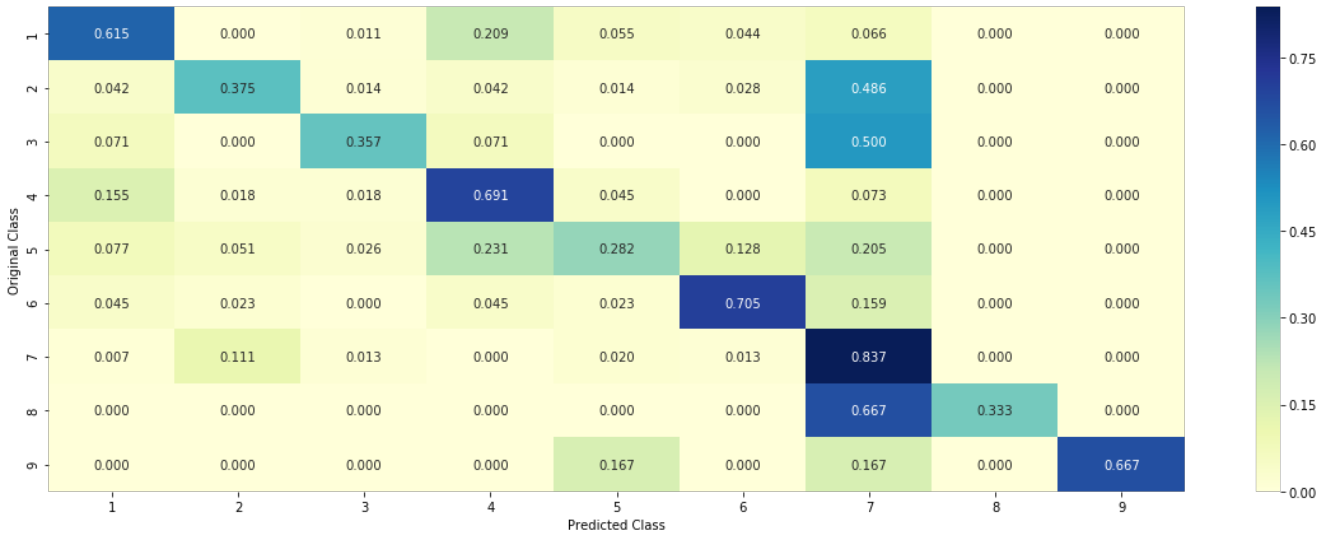
```
Log loss : 1.146413204671599
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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




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



Observations

- The best alpha value is 0.001.
- The test log loss by Linear SVM is 1.11.
- Number of misclassified points are 0.3627.

Feature Importance

For Incorrectly classified point

In [76]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.104  0.0858 0.0074 0.0491 0.5812 0.0362 0.123  0.0059 0.0073]]
Actual Class : 4
--------------------------------------------------
396 Text feature [pyrimidines] present in test data point [True]
483 Text feature [f1541] present in test data point [True]
484 Text feature [g230x] present in test data point [True]
485 Text feature [q110r] present in test data point [True]
486 Text feature [ezviewtm] present in test data point [True]
487 Text feature [d301n] present in test data point [True]
488 Text feature [pyrimidone] present in test data point [True]
489 Text feature [dermatoscopy] present in test data point [True]
490 Text feature [292c] present in test data point [True]
491 Text feature [y16x] present in test data point [True]
492 Text feature [p95s] present in test data point [True]
493 Text feature [xp295be] present in test data point [True]
494 Text feature [901g] present in test data point [True]
495 Text feature [cycloadenines] present in test data point [True]
496 Text feature [l325f] present in test data point [True]
497 Text feature [s362l] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

For Correctly classified point

In [77]:

```
test_point_index = 100
no_feature = 500

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

```
Predicted Class : 4
```

```
Predicted Class Probabilities: [[0.0508 0.0503 0.0093 0.7377 0.0297 0.0192 0.0943 0.0046 0.0042]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

Random Forest Classifier

Hyper paramter tuning (With One hot Encoding)

In [78]:

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

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

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

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

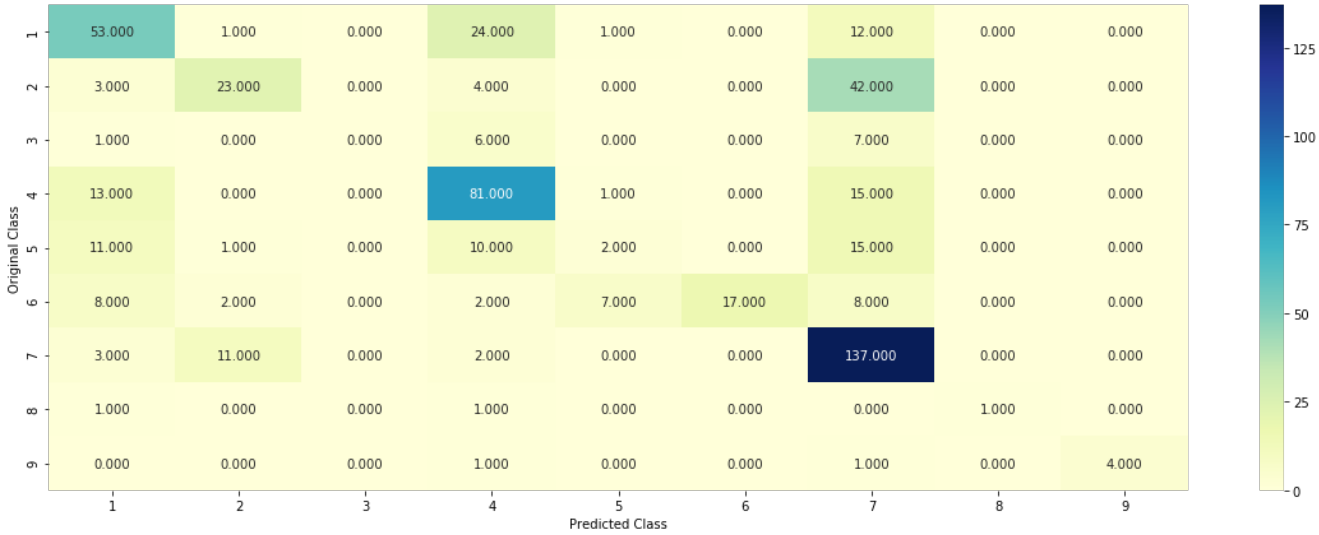
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2614781130529913
for n_estimators = 100 and max depth =  10
Log Loss : 1.1876944571609138
for n_estimators = 200 and max depth =  5
Log Loss : 1.2482958045197232
for n_estimators = 200 and max depth =  10
Log Loss : 1.1769129889657957
for n_estimators = 500 and max depth =  5
Log Loss : 1.245557387176137
for n_estimators = 500 and max depth =  10
Log Loss : 1.1756940381917431
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2406928496150462
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1758533165036986
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2399040470536569
for n_estimators = 2000 and max depth =  10
Log Loss : 1.174706260788384
For values of best estimator =  2000 The train log loss is: 0.7048815084029313
For values of best estimator =  2000 The cross validation log loss is: 1.174706260788384
For values of best estimator =  2000 The test log loss is: 1.147870913318269
```

Testing model with best hyper parameters (One Hot Encoding)

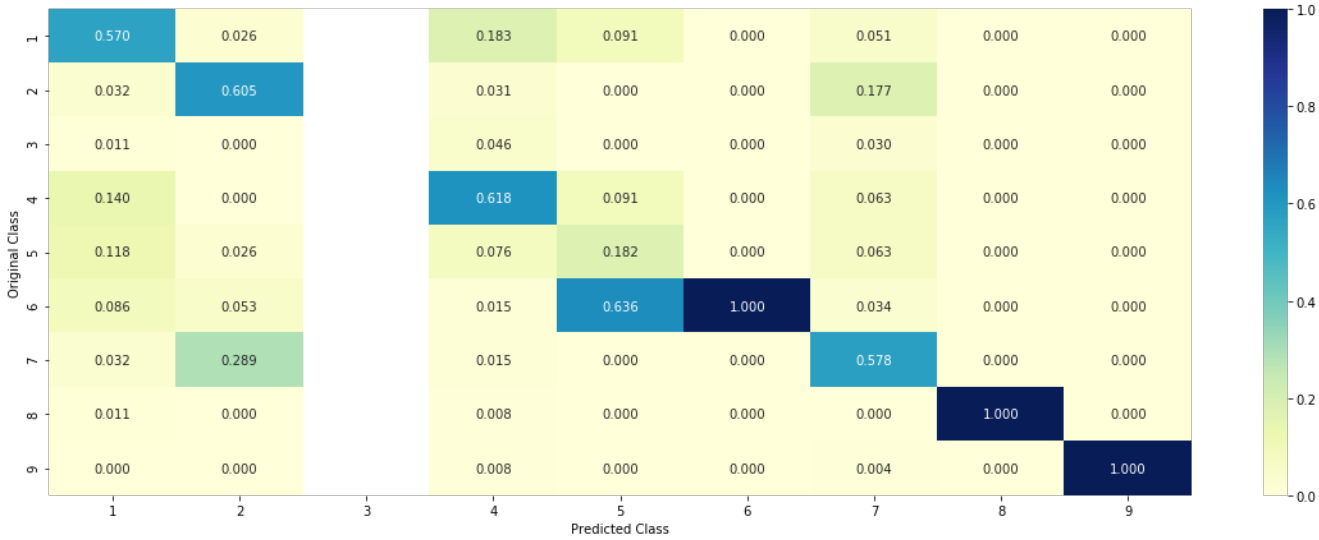
In [79]:

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

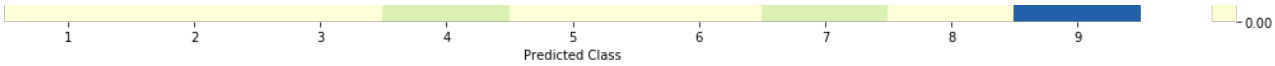
```
Log loss : 1.174706260788384
Number of mis-classified points : 0.40225563909774437
-------------------- Confusion matrix --------------------
```




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




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



Observations

- The best alpha value is 2000.
- The test log loss by Random Forest with one hot encoding is 1.14.
- Number of misclassified points are 0.4022.

Feature Importance

Correctly Classified point

In [80]:

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

test_point_index = 1
no_feature = 100

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2443 0.0892 0.0267 0.3931 0.0837 0.0551 0.0895 0.0079 0.0105]]
Actual Class : 4
--------------------------------------------------
5 Text feature [activation] present in test data point [True]
6 Text feature [missense] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
10 Text feature [nonsense] present in test data point [True]
11 Text feature [function] present in test data point [True]
13 Text feature [signaling] present in test data point [True]
14 Text feature [suppressor] present in test data point [True]
17 Text feature [akt] present in test data point [True]
18 Text feature [cells] present in test data point [True]
20 Text feature [functional] present in test data point [True]
24 Text feature [downstream] present in test data point [True]
25 Text feature [activate] present in test data point [True]
27 Text feature [loss] present in test data point [True]
49 Text feature [treated] present in test data point [True]
51 Text feature [patients] present in test data point [True]
53 Text feature [defective] present in test data point [True]
58 Text feature [protein] present in test data point [True]
60 Text feature [oncogene] present in test data point [True]
65 Text feature [clinical] present in test data point [True]
68 Text feature [cell] present in test data point [True]
70 Text feature [repair] present in test data point [True]
71 Text feature [pten] present in test data point [True]
72 Text feature [p53] present in test data point [True]
78 Text feature [dna] present in test data point [True]
88 Text feature [kit] present in test data point [True]
95 Text feature [expression] present in test data point [True]
99 Text feature [lines] present in test data point [True]
Out of the top  100  features  27 are present in query point
```

Inorrectly Classified point

Incorrectly Classified point

In [81]:

```
test_point_index = 100
no_feature = 100

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0909 0.075  0.0234 0.5344 0.0501 0.0436 0.1675 0.0067 0.0084]]
Actuall Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
4 Text feature [constitutive] present in test data point [True]
5 Text feature [activation] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
11 Text feature [function] present in test data point [True]
13 Text feature [signaling] present in test data point [True]
15 Text feature [oncogenic] present in test data point [True]
17 Text feature [akt] present in test data point [True]
18 Text feature [cells] present in test data point [True]
19 Text feature [growth] present in test data point [True]
24 Text feature [downstream] present in test data point [True]
25 Text feature [activate] present in test data point [True]
32 Text feature [expressing] present in test data point [True]
49 Text feature [treated] present in test data point [True]
50 Text feature [transforming] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
58 Text feature [protein] present in test data point [True]
60 Text feature [oncogene] present in test data point [True]
67 Text feature [phospho] present in test data point [True]
68 Text feature [cell] present in test data point [True]
73 Text feature [proteins] present in test data point [True]
76 Text feature [phosphoinositide] present in test data point [True]
78 Text feature [dna] present in test data point [True]
80 Text feature [amplification] present in test data point [True]
88 Text feature [kit] present in test data point [True]
89 Text feature [proliferation] present in test data point [True]
93 Text feature [factor] present in test data point [True]
95 Text feature [expression] present in test data point [True]
97 Text feature [inhibited] present in test data point [True]
Out of the top  100  features  32 are present in query point
```

Hyper paramter tuning (With Response Coding)

In [82]:

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

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
_train, 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 : 2.0951197412987494
for n_estimators = 10 and max depth =  3
Log Loss : 1.8459073329166769
for n_estimators = 10 and max depth =  5
Log Loss : 1.4640726302908331
for n_estimators = 10 and max depth =  10
Log Loss : 1.8671169699107457
for n_estimators = 50 and max depth =  2
Log Loss : 1.7601527365056566
for n_estimators = 50 and max depth =  3
Log Loss : 1.4298518180852928
for n_estimators = 50 and max depth =  5
Log Loss : 1.3878405035939285
for n_estimators = 50 and max depth =  10
Log Loss : 1.6547657461129928
for n_estimators = 100 and max depth =  2
Log Loss : 1.5988238052400199
for n_estimators = 100 and max depth =  3
Log Loss : 1.4238119244704797
for n_estimators = 100 and max depth =  5
Log Loss : 1.4040224746019536
for n_estimators = 100 and max depth =  10
Log Loss : 1.6637290441571593
for n_estimators = 200 and max depth =  2
Log Loss : 1.616428709630442
for n_estimators = 200 and max depth =  3
Log Loss : 1.4257517937300865
for n_estimators = 200 and max depth =  5
Log Loss : 1.417880261650661
for n_estimators = 200 and max depth =  10
Log Loss : 1.6501071319844138
for n_estimators = 500 and max depth =  2
Log Loss : 1.6473034354032272
for n_estimators = 500 and max depth =  3
Log Loss : 1.5141617201463209
for n_estimators = 500 and max depth =  5
Log Loss : 1.4292331077292517
for n_estimators = 500 and max depth =  10
Log Loss : 1.677653639773243
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6230434883253733
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4939009614542664
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4102777720831627
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6426298891905176
For values of best alpha =  50 The train log loss is: 0.08277014412854723
For values of best alpha =  50 The cross validation log loss is: 1.3878405035939287
For values of best alpha =  50 The test log loss is: 1.3066910375527085
```

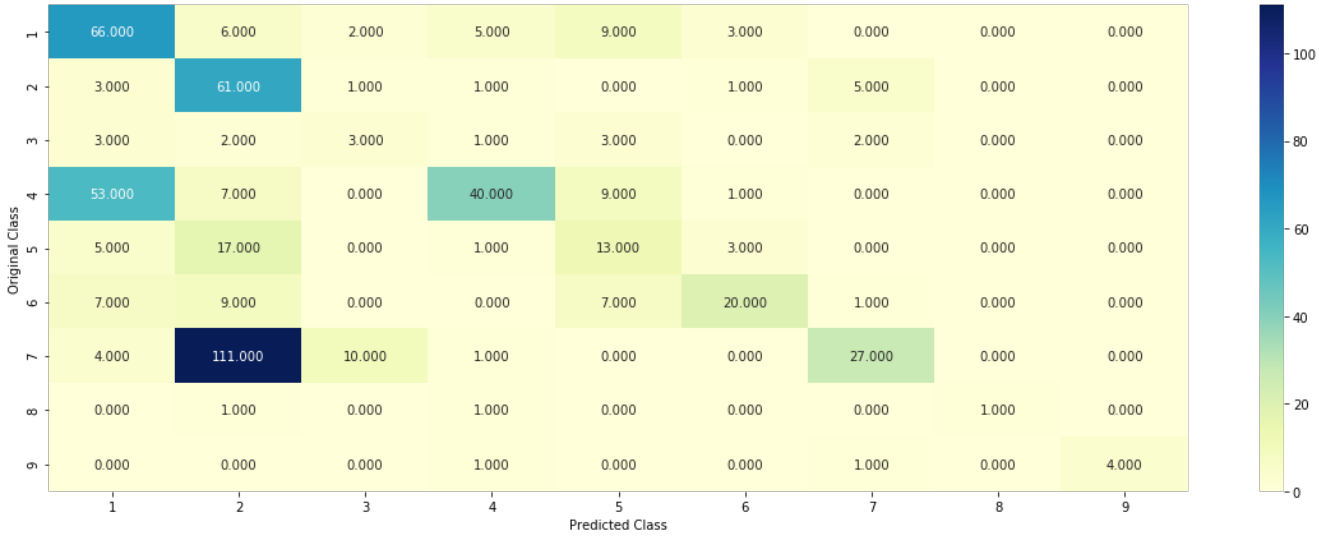
Testing model with best hyper parameters (Response Coding)

In [83]:

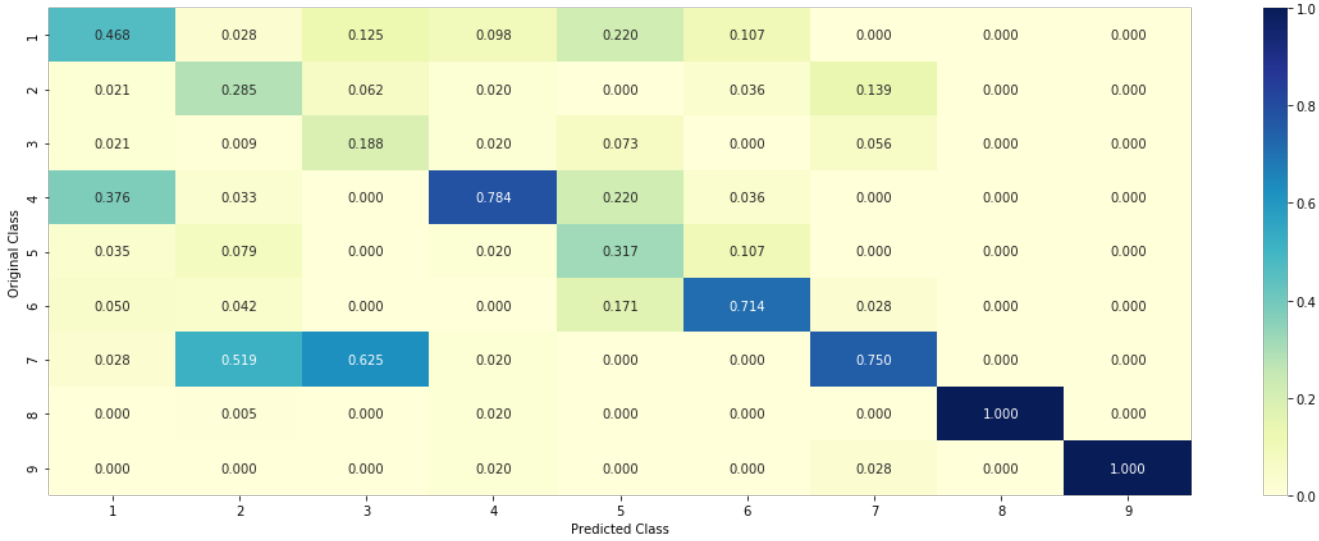
```
```

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

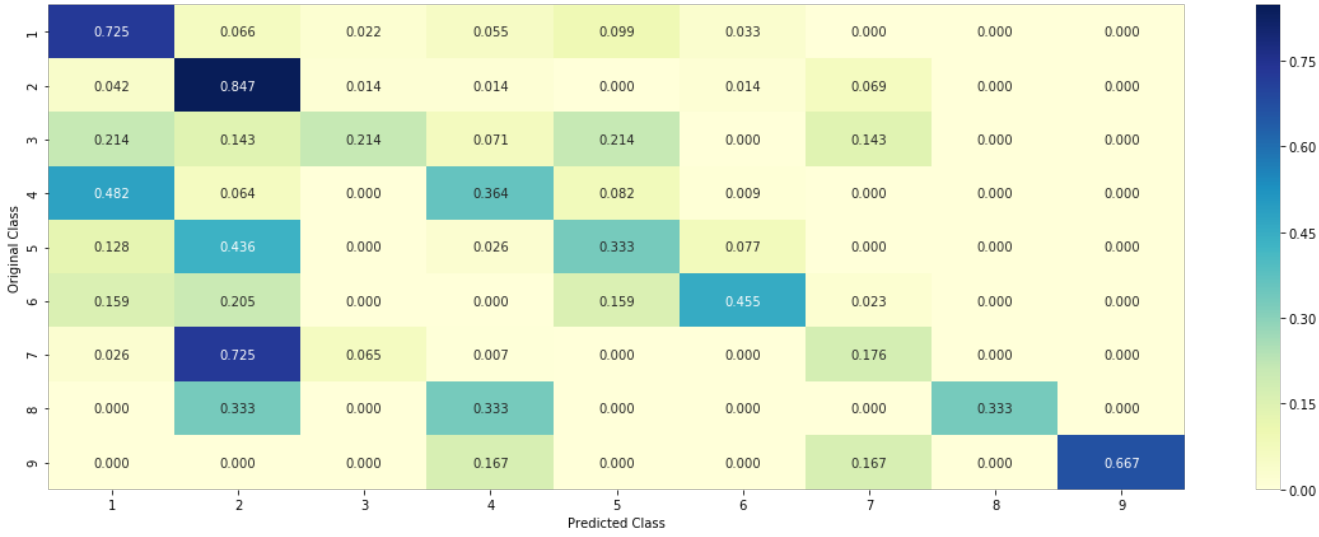
```
Log loss : 1.3878405035939287
Number of mis-classified points : 0.5582706766917294
-------------------- Confusion matrix --------------------
```




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




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



Observations

- The best alpha value is 50.
- The test log loss by Random Forest with response coding is 1.30.
- Number of misclassified points are 0.5582.

Feature Importance

Correctly Classified point

In [84]:

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

test_point_index = 1
no_feature = 27

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)

for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2577 0.0338 0.1334 0.3951 0.0638 0.0278 0.012  0.0372 0.0393]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation 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
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
```

Incorrectly Classified point

In [85]:

```
test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)

for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3015 0.0584 0.128  0.3823 0.0357 0.0274 0.0151 0.0251 0.0265]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation 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
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
```

Stack the models

Testing with hyper parameter tuning

In [86]:

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

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

clf3 = MultinomialNB(alpha=0.001)
```

```
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))

sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))

sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)

alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999

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

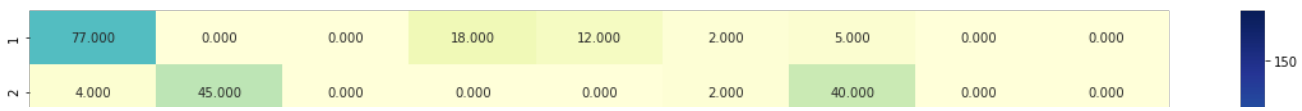
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.66
Naive Bayes : Log Loss: 1.28
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.819
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.730
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.360
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.226
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.509
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.861
```

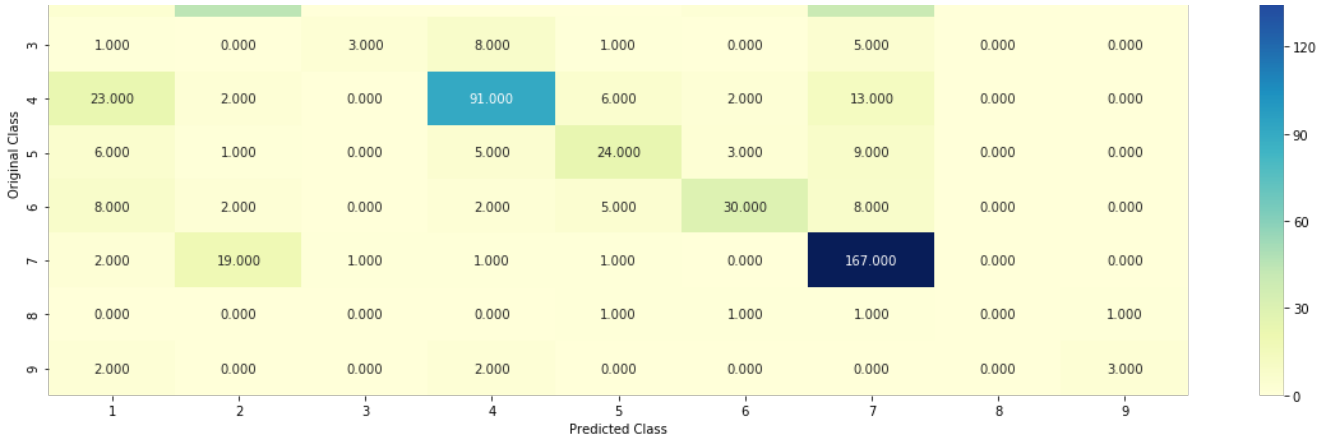
Testing the model with the best hyper parameters

In [87]:

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

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

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

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

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

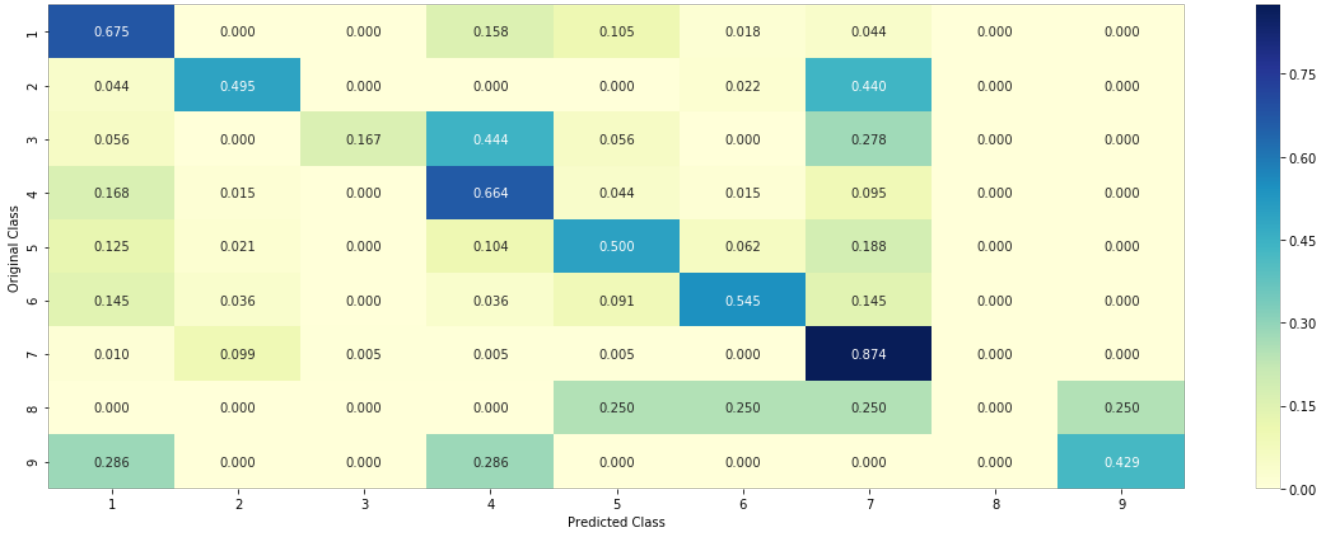
```
Log loss (train) on the stacking classifier : 0.5129146629931494
Log loss (CV) on the stacking classifier : 1.2259620599474286
Log loss (test) on the stacking classifier : 1.1225525737461446
Number of missclassified point : 0.3383458646616541
-------------------- Confusion matrix --------------------
```





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

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



Observations

- The best alpha value is 0.1.
- The test log loss by Stacking classifier is 1.12.
- Number of misclassified points are 0.3383.

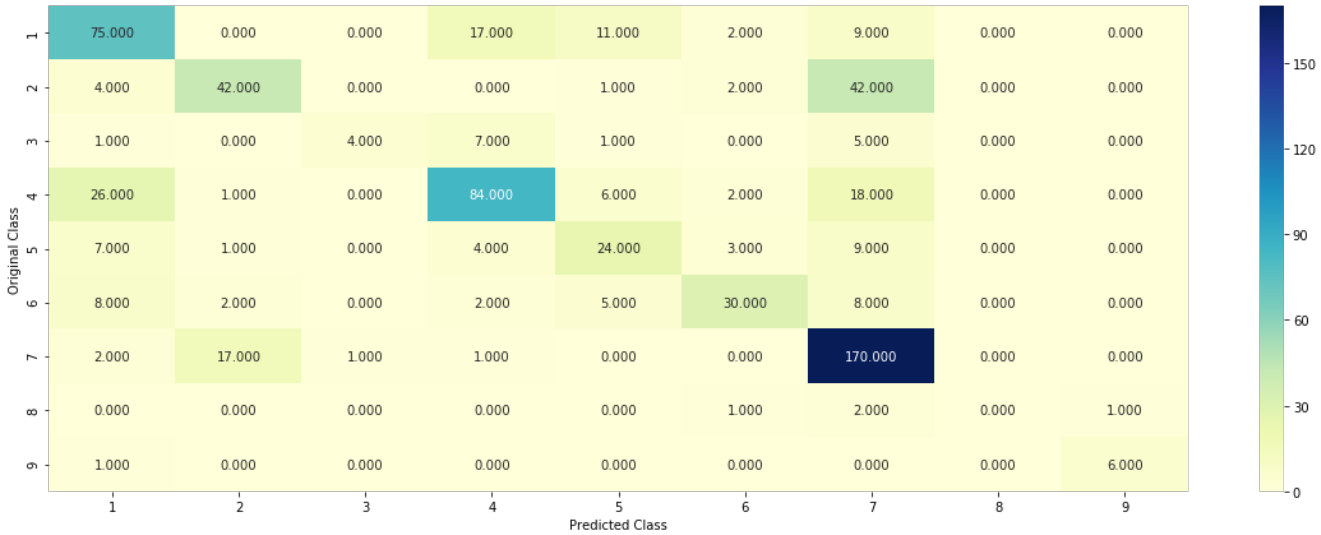
Maximum Voting classifier

Maximum Voting classifier

In [88]:

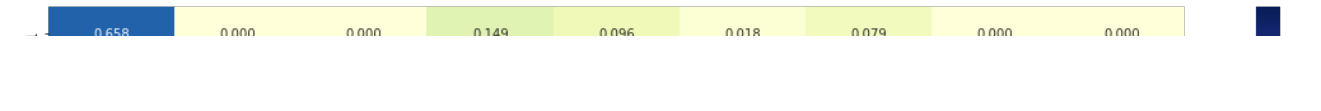
```
from sklearn.ensemble import VotingClassifier

vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)

print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y,
vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)-
test_y))/test_y.shape[0])

plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

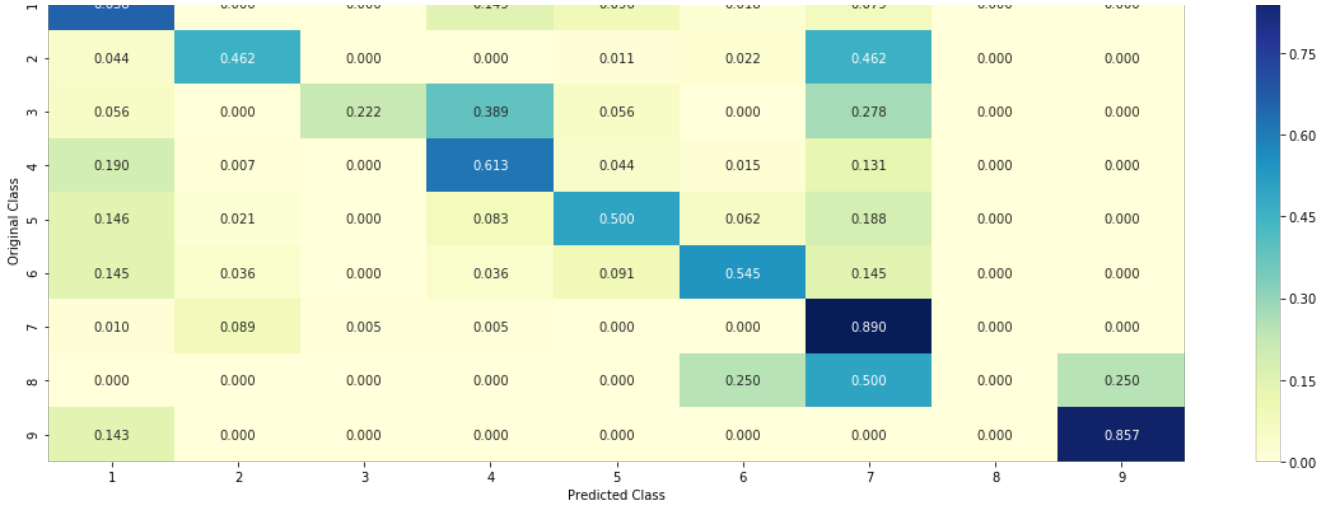
```
Log loss (train) on the VotingClassifier : 0.875427852890204
Log loss (CV) on the VotingClassifier : 1.1992683082135023
Log loss (test) on the VotingClassifier : 1.1752829761856285
Number of missclassified point : 0.3458646616541353
-------------------- Confusion matrix --------------------
```




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


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





Observations

- The test log loss by Majority voting classifier is 1.17.
- Number of misclassified points are 0.3458.

Task - 1

Using Tf-idf features

In [109]:

```
text_vectorizer = TfidfVectorizer(min_df=3)

train_text_feature_tfidf = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_features = text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_tfidf.sum(axis=0).A1

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 : 53760
```

In [110]:

```
# Normalizing

train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)

test_text_feature_tfidf = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_tfidf = normalize(test_text_feature_tfidf, axis=0)

cv_text_feature_tfidf = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_tfidf = normalize(cv_text_feature_tfidf, axis=0)
```

Merging all the features

In [111]:

```
# merging gene, variance and text features

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

```
train_x_tfidf = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf)).tocsr()
train_y = np.array(list(train_df['Class']))

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

cv_x_tfidf = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [112]:

```
# Checking total datapoints in Train, Test and CV

print("(number of data points * number of features) in train data = ", train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", test_x_tfidf.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_tfidf.shape)
```

```
(number of data points * number of features) in train data =  (2124, 55945)
(number of data points * number of features) in test data =  (665, 55945)
(number of data points * number of features) in cross validation data = (532, 55945)
```

Applying Machine Learning models

Naive Bayes

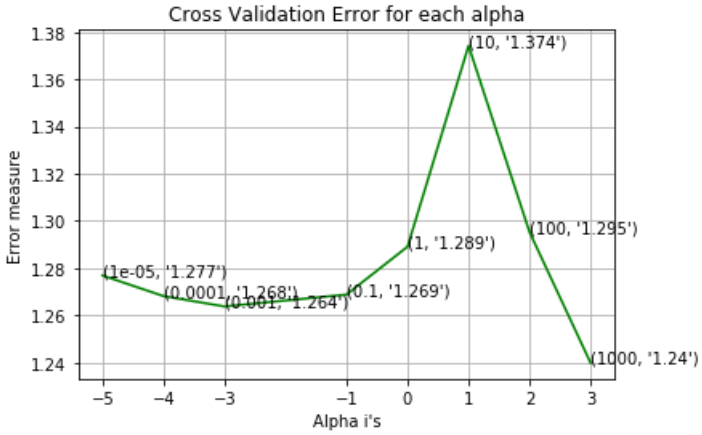
Hyper parameter tuning

In [113]:

```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []

for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha = i)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')

for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))

plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
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_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
```

```
predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.2769085836078728
for alpha = 0.0001
Log Loss : 1.2681795946092136
for alpha = 0.001
Log Loss : 1.2637916533089544
for alpha = 0.1
Log Loss : 1.2688212002951393
for alpha = 1
Log Loss : 1.2891207339825774
for alpha = 10
Log Loss : 1.3742185132101814
for alpha = 100
Log Loss : 1.2952490659818559
for alpha = 1000
Log Loss : 1.2399020375061882
```



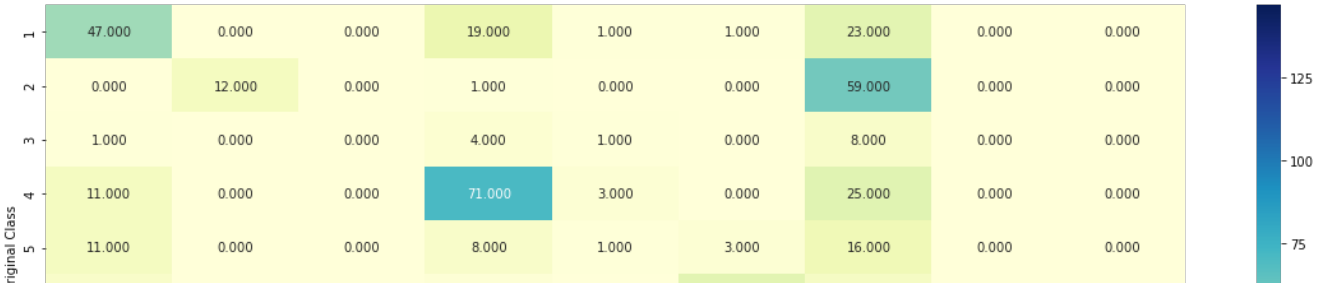

```
For values of best alpha =  1000 The train log loss is: 0.9220250222743688
For values of best alpha =  1000 The cross validation log loss is: 1.2399020375061882
For values of best alpha =  1000 The test log loss is: 1.1883733203351687
```

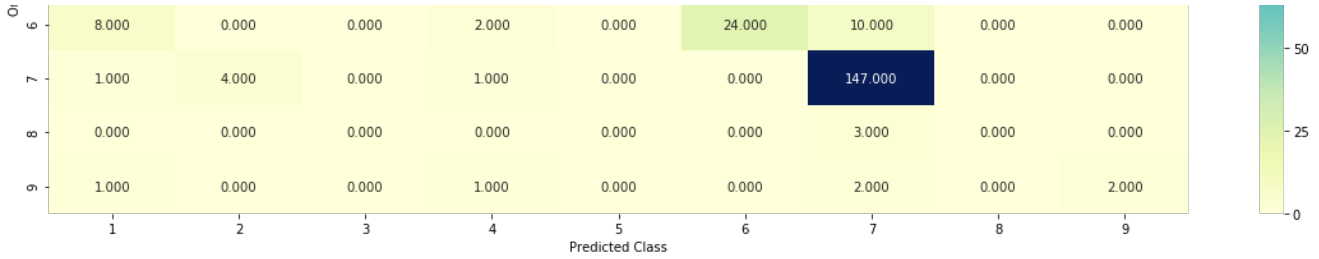
Testing with best Hyper parameter

In [114]:

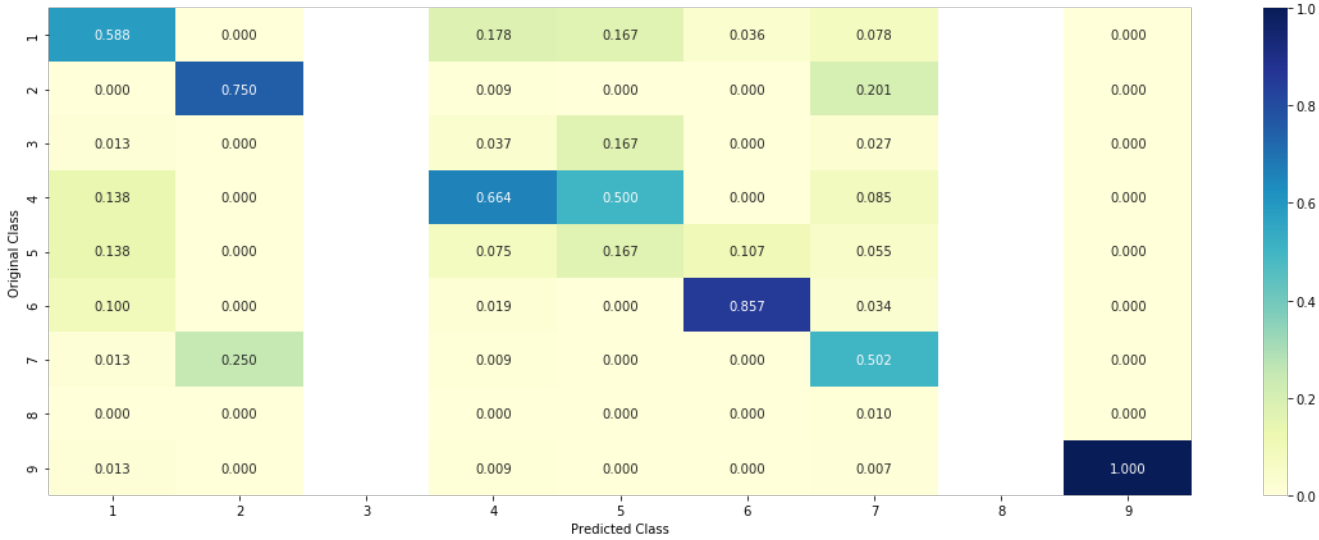
```
clf = MultinomialNB(alpha = alpha[best_alpha])
clf.fit(train_x_tfidf, train_y)

sig_clf = CalibratedClassifierCV(clf, method = "sigmoid")
sig_clf.fit(train_x_tfidf, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)

print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfidf)- cv_y))/cv
_y.shape[0])

plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf.toarray()))
```

```
Log Loss : 1.2399020375061882
Number of missclassified point : 0.42857142857142855
-------------------- Confusion matrix --------------------
```

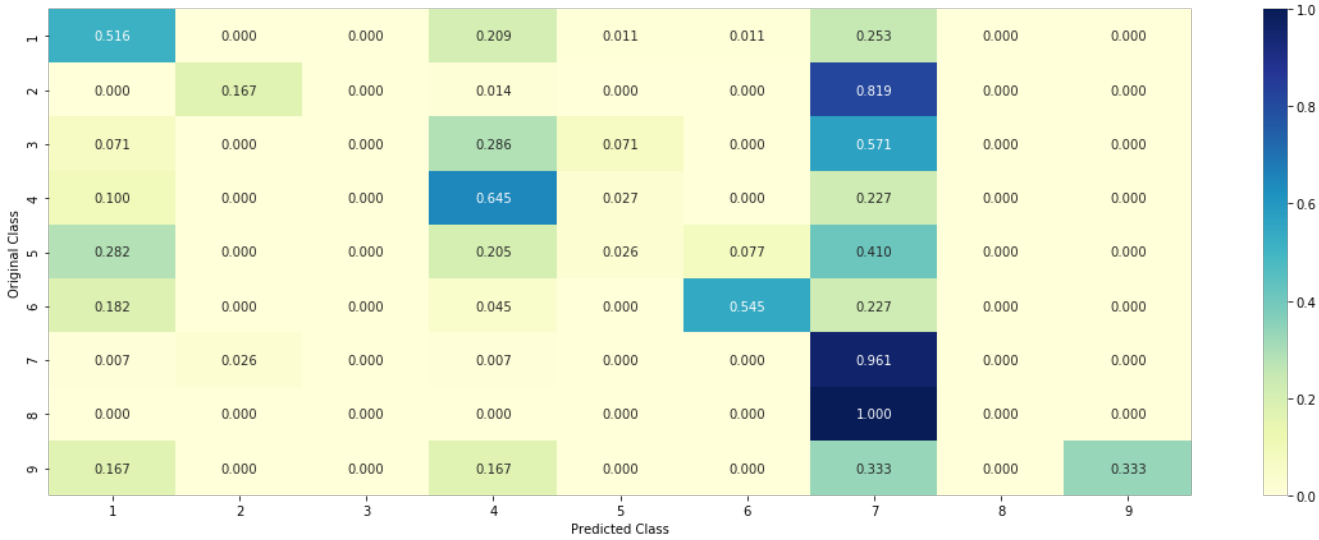




---------- Precision matrix (Columm Sum=1) ----------



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



Observations

- The best alpha value is 1000.
- The test log loss by naive Bayes is 1.18.
- Number of misclassified points are 0.4285.

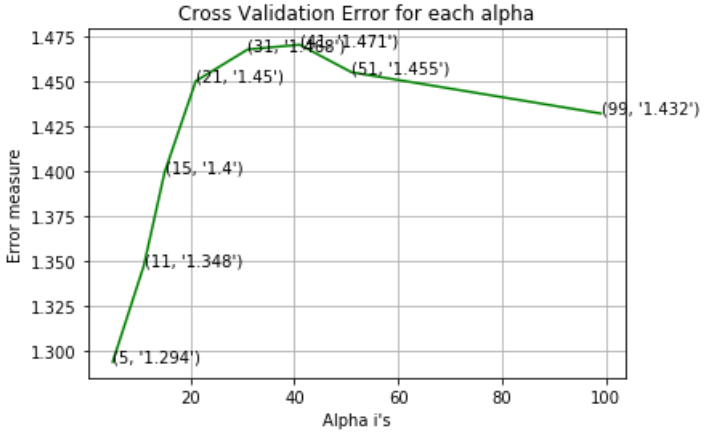
KNN

Hyper parameter tuning

In [115]:

```
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []

for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
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_x_tfidf)
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 = 5
Log Loss : 1.2941910205458698
for alpha = 11
Log Loss : 1.3477740079698028
for alpha = 15
Log Loss : 1.399721597832882
for alpha = 21
Log Loss : 1.45040678429071
for alpha = 31
Log Loss : 1.467997131093865
for alpha = 41
Log Loss : 1.4706001117753373
for alpha = 51
Log Loss : 1.4552817439550196
for alpha = 99
Log Loss : 1.4323727052622992
```



For values of best alpha = 5 The train log loss is: 0.06242000021102139

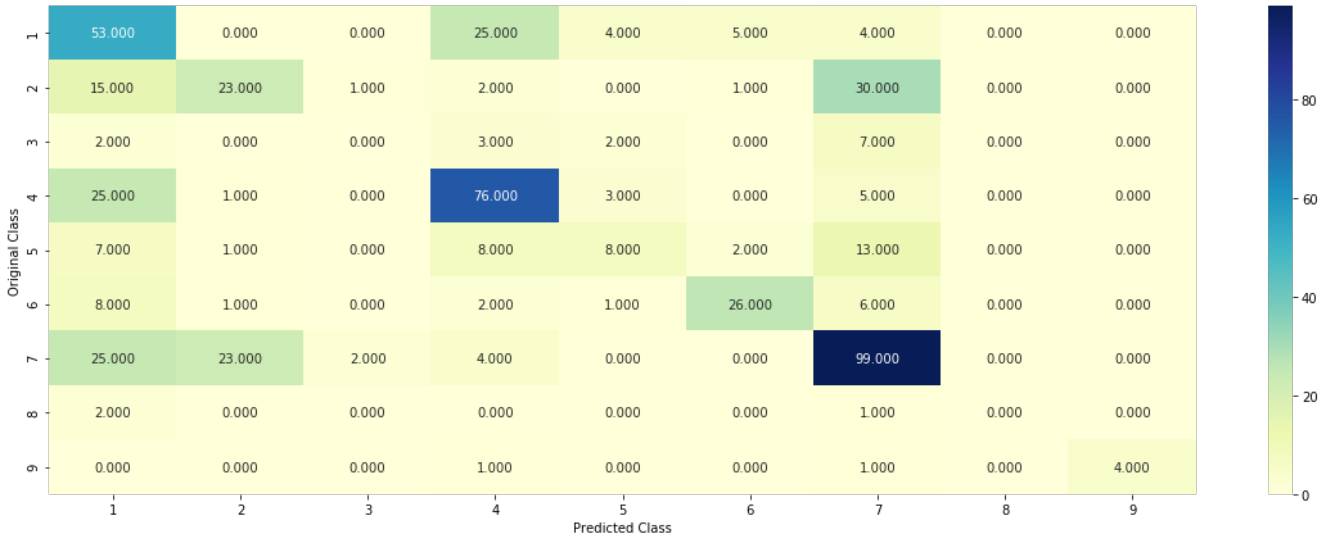
```
For values of best alpha =  5 The train log loss is: 0.9634390893102138
For values of best alpha =  5 The cross validation log loss is: 1.2941910205458698
For values of best alpha =  5 The test log loss is: 1.2638645736069705
```

Testing the model with best hyper paramters

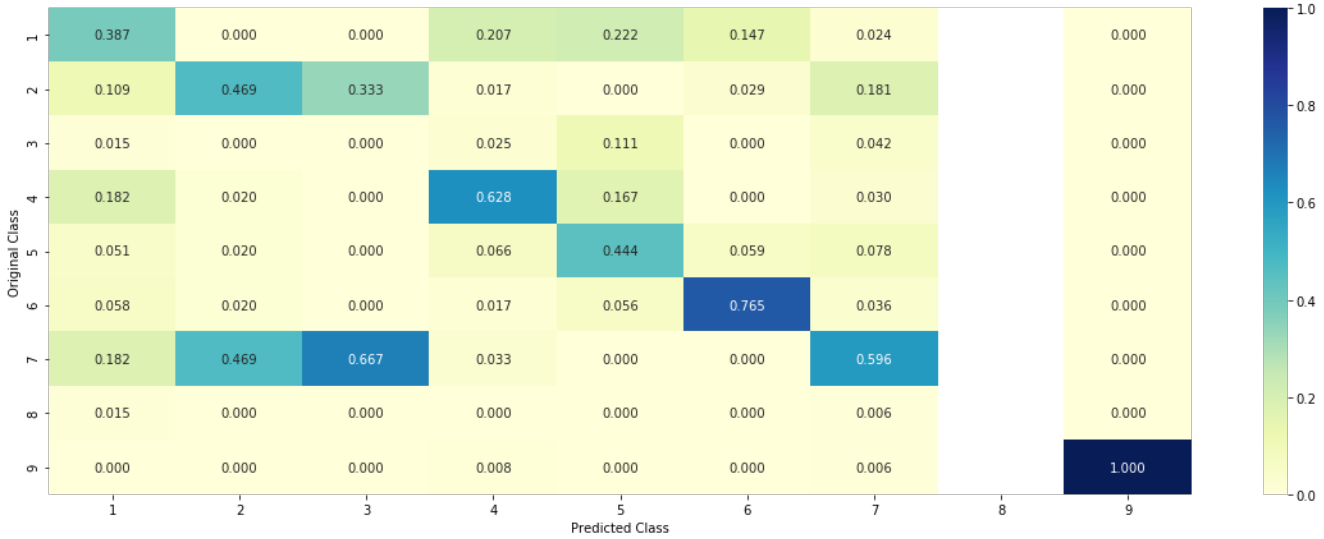
In [116]:

```
clf = KNeighborsClassifier(n_neighbors = alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

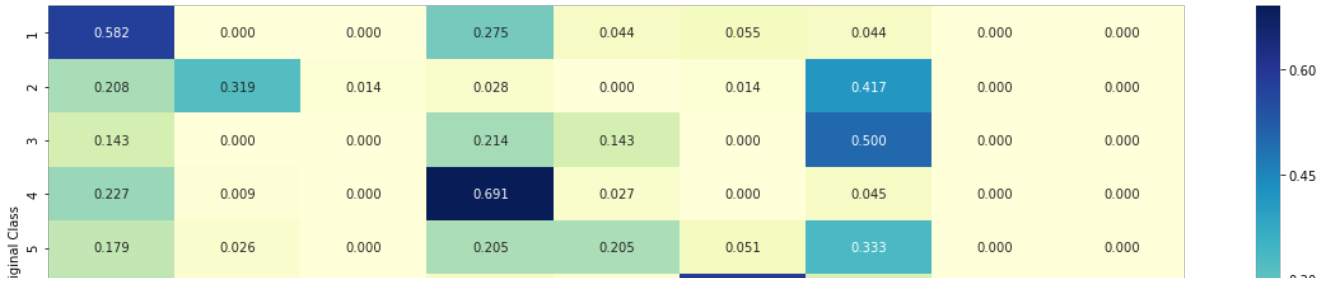
```
Log loss : 1.2941910205458698
Number of mis-classified points : 0.4567669172932331
-------------------- Confusion matrix --------------------
```

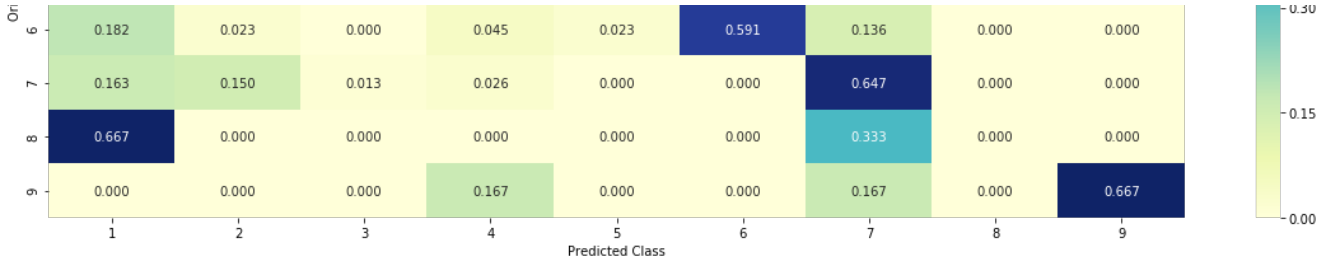



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





Observations

- The best k value is 5.
- The test log loss by KNN is 1.26.
- Number of misclassified points are 0.4567.

Logistic Regression with Class Balancing

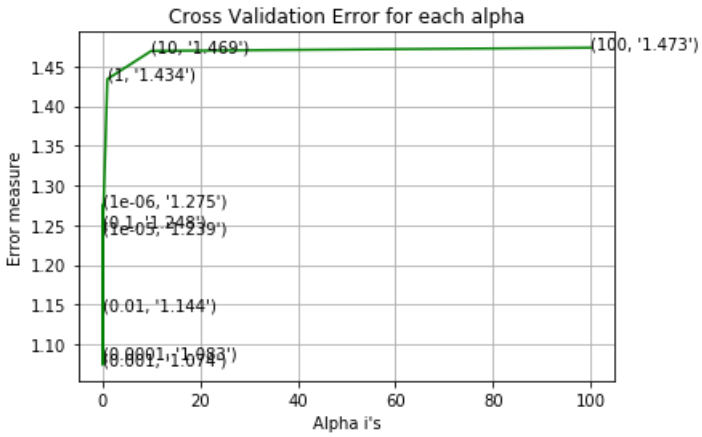
Hyper paramter tuning

In [117]:

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

for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))

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

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

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
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_x_tfidf)
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 : 1.2751024238615483
for alpha = 1e-05
Log Loss : 1.2393960476092876
```

```
for alpha = 0.0001
Log Loss : 1.0830510154512
for alpha = 0.001
Log Loss : 1.0739850377446598
for alpha = 0.01
Log Loss : 1.1438078247453767
for alpha = 0.1
Log Loss : 1.2481229489727215
for alpha = 1
Log Loss : 1.433919070546794
for alpha = 10
Log Loss : 1.4693911362074046
for alpha = 100
Log Loss : 1.4734067731654648
```



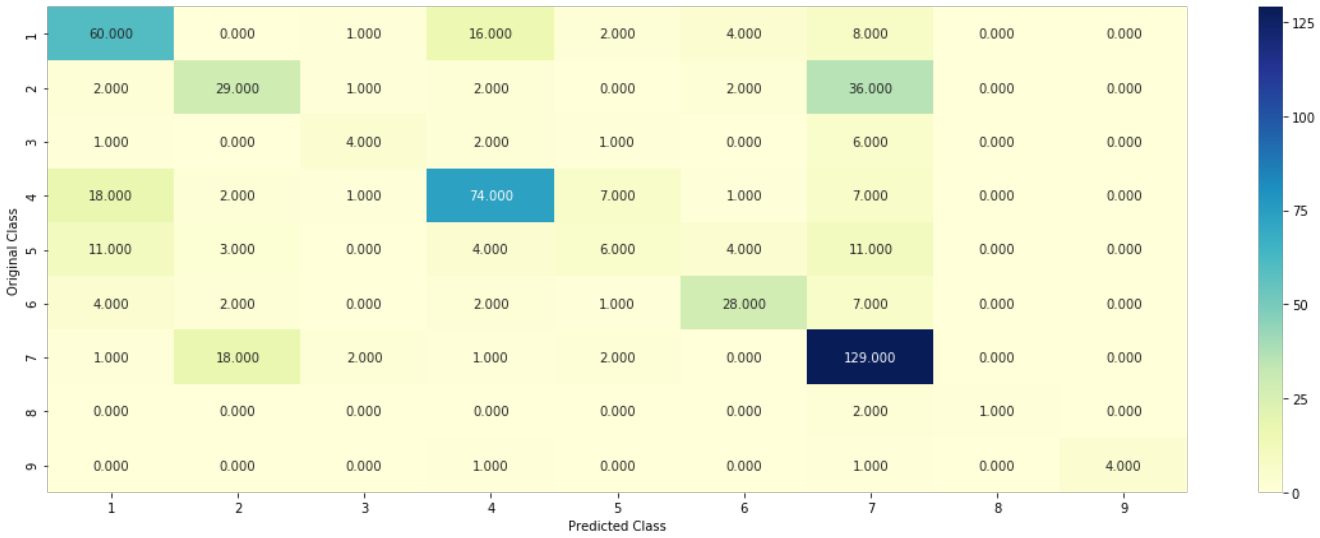

```
For values of best alpha =  0.001 The train log loss is: 0.5105594504248592
For values of best alpha =  0.001 The cross validation log loss is: 1.0739850377446598
For values of best alpha =  0.001 The test log loss is: 1.0113440638298266
```

Testing the model with best hyper paramters

In [118]:

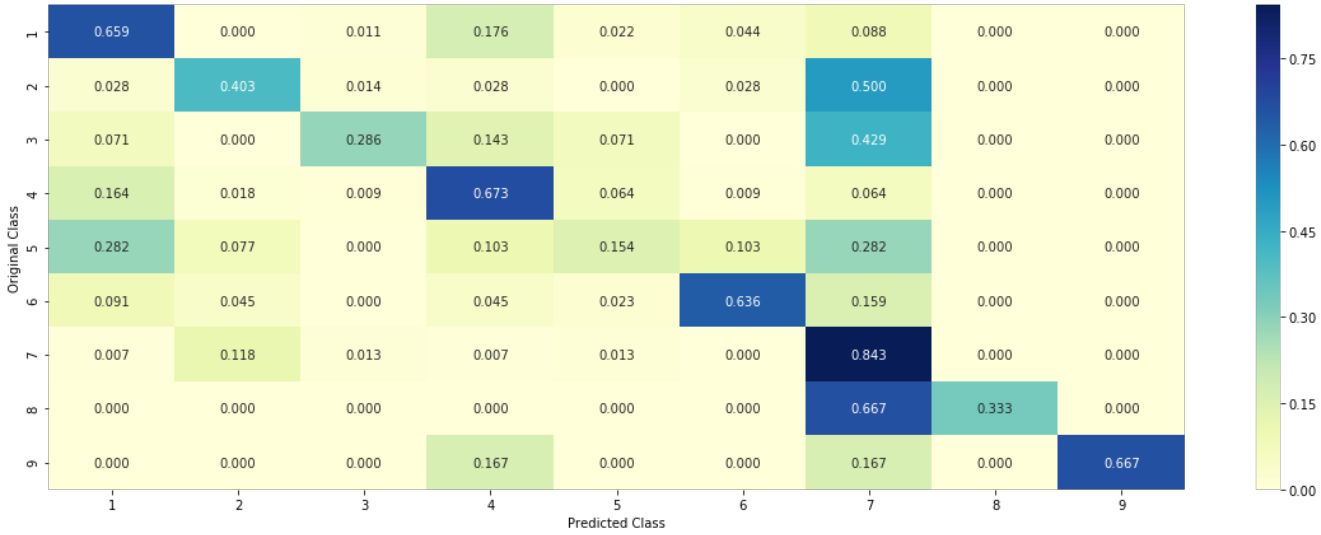
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

```
Log loss : 1.0739850377446598
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




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


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



Observations

- The best alpha value is 0.001.
- The test log loss by Logistic Regression with class balancing is 1.01.
- Number of misclassified points are 0.3703.

Logistic Regression Without Class balancing

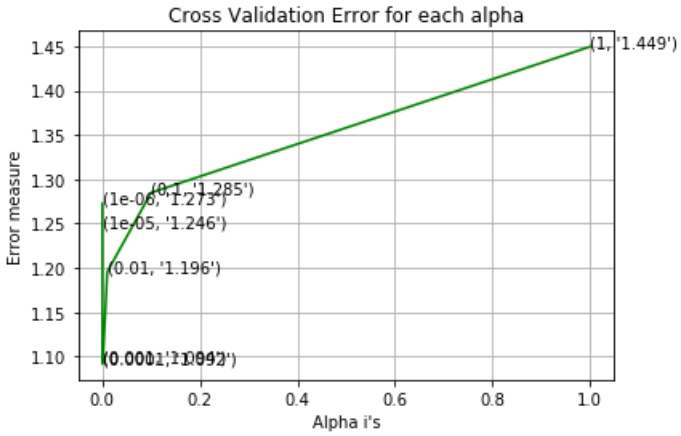
Hyper paramter tuning

In [119]:

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []

for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
```

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

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

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

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
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_x_tfidf)
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 : 1.272531065193417
for alpha = 1e-05
Log Loss : 1.2459461384968082
for alpha = 0.0001
Log Loss : 1.0916561932098423
for alpha = 0.001
Log Loss : 1.0939303709450758
for alpha = 0.01
Log Loss : 1.1955109406897726
for alpha = 0.1
Log Loss : 1.2847854874007825
for alpha = 1
Log Loss : 1.4494429449654622
```



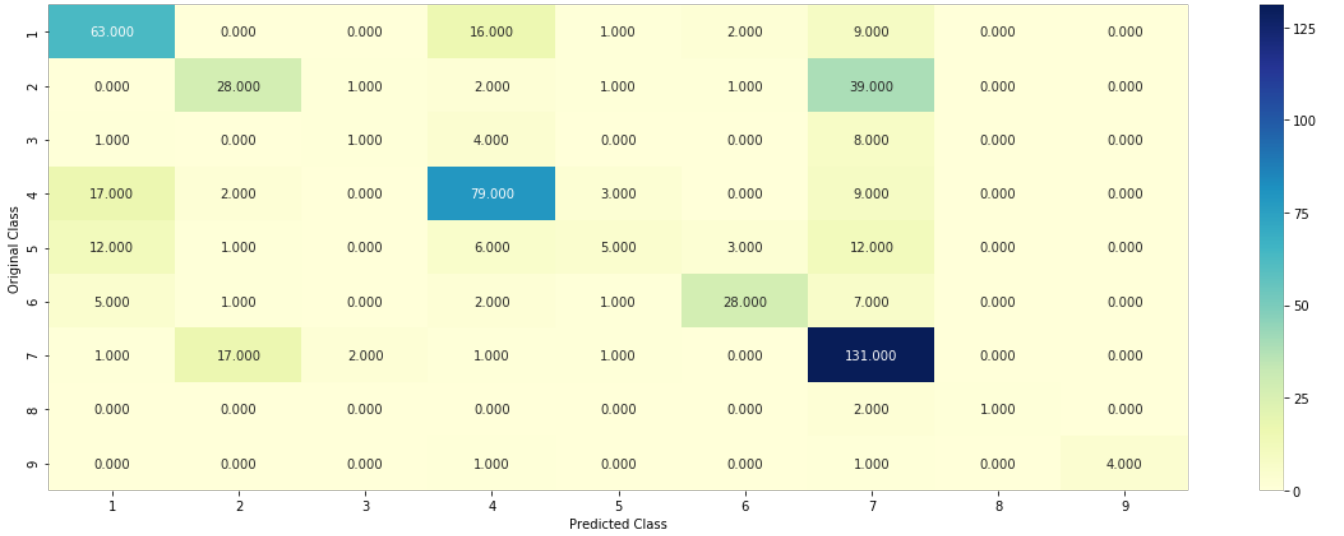

```
For values of best alpha =  0.0001 The train log loss is: 0.4759812532928805
For values of best alpha =  0.0001 The cross validation log loss is: 1.0916561932098423
For values of best alpha =  0.0001 The test log loss is: 1.0286615314703274
```

Testing model with best hyper parameters

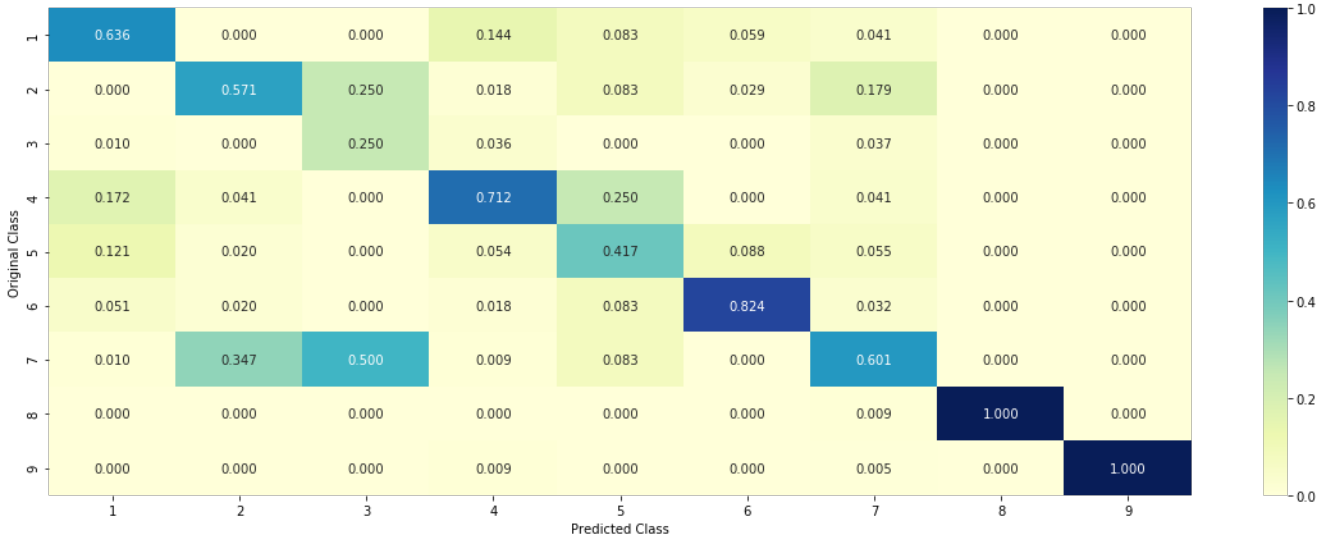
In [120]:

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

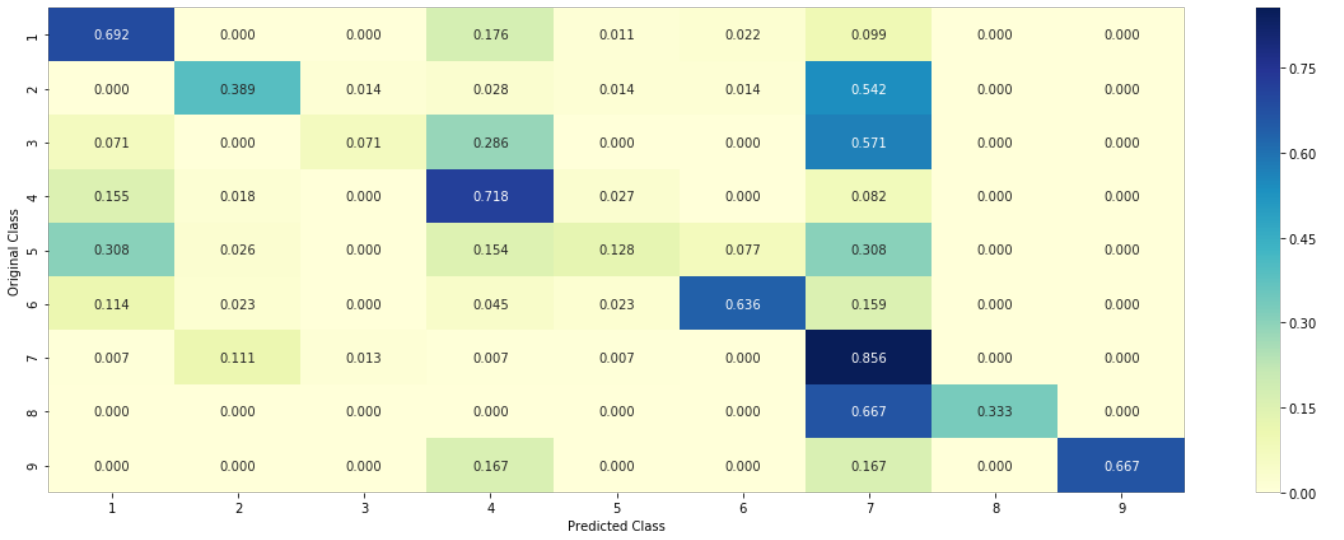
```
Log loss : 1.0916561932098423
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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



Observations

- The best alpha value is 0.0001.

- The test log loss by Logistic Regression without class balancing is 1.02.
- Number of misclassified points are 0.3609.

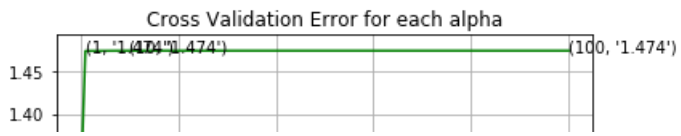
Linear Support Vector Machines

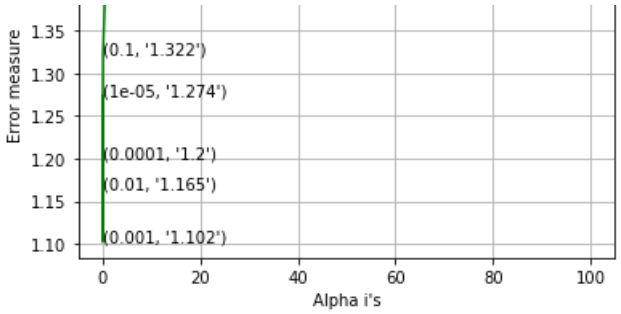
Hyper paramter tuning

In [121]:

```
alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []

for i in alpha:
    print("for C =", i)
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state
=42)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

predict_y = sig_clf.predict_proba(train_x_tfidf)
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_tfidf)
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_x_tfidf)
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 : 1.2735936067612563
for C = 0.0001
Log Loss : 1.2000328846604487
for C = 0.001
Log Loss : 1.1024487725898398
for C = 0.01
Log Loss : 1.164610559812309
for C = 0.1
Log Loss : 1.3220900093065169
for C = 1
Log Loss : 1.4738461055677436
for C = 10
Log Loss : 1.474100505711623
for C = 100
Log Loss : 1.474100108095067
```






```
For values of best alpha =  0.001 The train log loss is: 0.5533588417656476
For values of best alpha =  0.001 The cross validation log loss is: 1.1024487725898398
For values of best alpha =  0.001 The test log loss is: 1.073887013337954
```

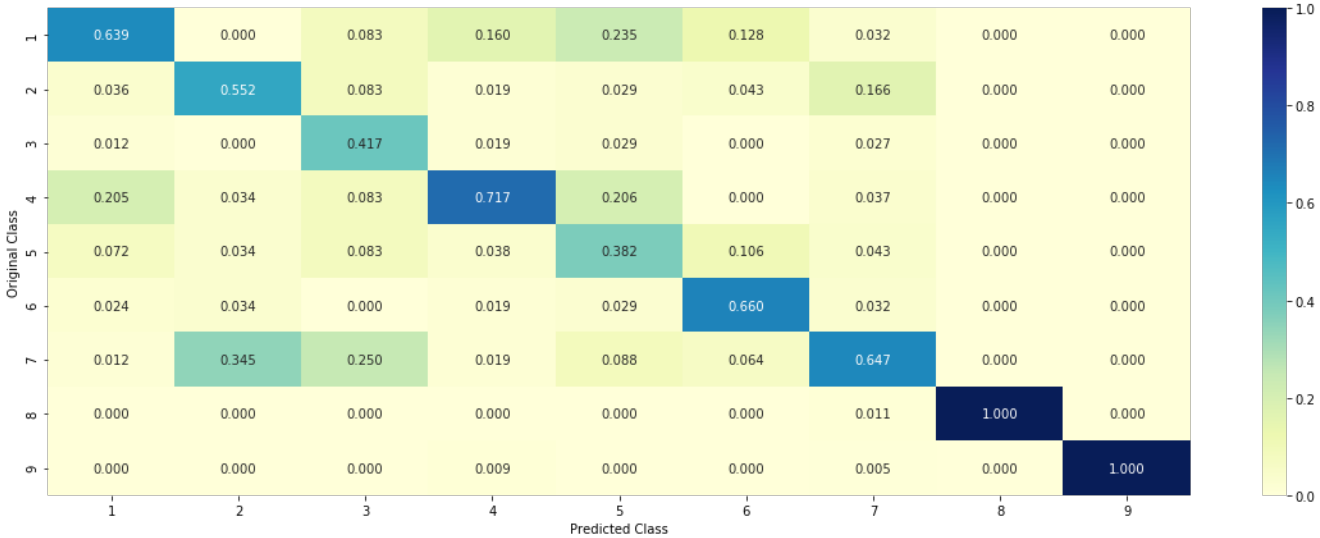
Testing model with best hyper parameters

In [122]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y, clf)
```

```
Log loss : 1.1024487725898398
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```


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



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

Observations

- The best alpha value is 0.001.
- The test log loss by Linear SVM is 1.07.
- Number of misclassified points are 0.3684.

Random Forest Classifier

Hyper parameter tuning

In [123]:

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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.268126835675337
for n_estimators = 100 and max depth =  10
Log Loss : 1.2005977274121666
for n_estimators = 200 and max depth =  5
Log Loss : 1.2556131580971834
for n_estimators = 200 and max depth =  10
Log Loss : 1.1916130891432741
for n_estimators = 500 and max depth =  5
Log Loss : 1.2511359532328394
for n_estimators = 500 and max depth =  10
Log Loss : 1.189059023548208
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2469193456463235
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1848191814169948
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2432647220946846
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1860459231997122
For values of best estimator =  1000 The train log loss is: 0.6583586380980803
For values of best estimator =  1000 The cross validation log loss is: 1.1848191814169948
For values of best estimator =  1000 The test log loss is: 1.1256824511651715
```

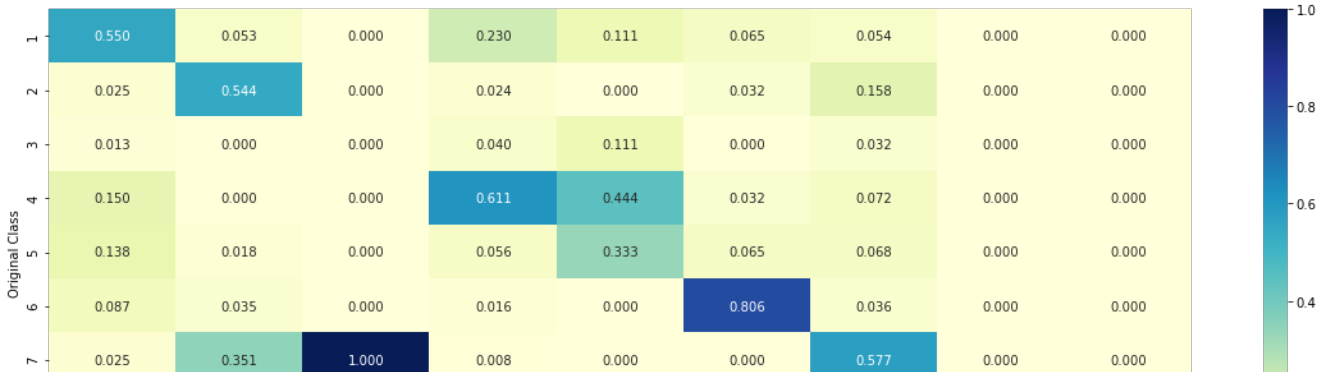
Testing model with best hyper parameters (One Hot Encoding)

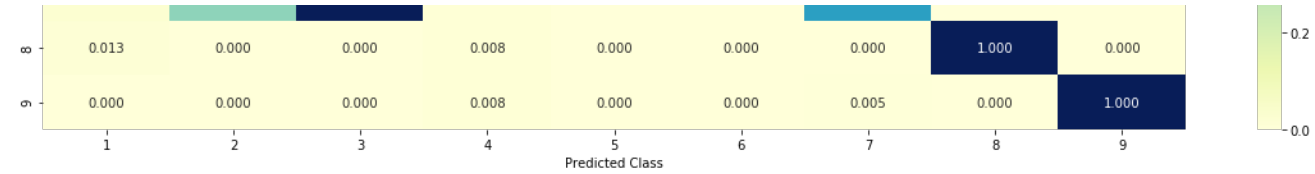
In [124]:

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

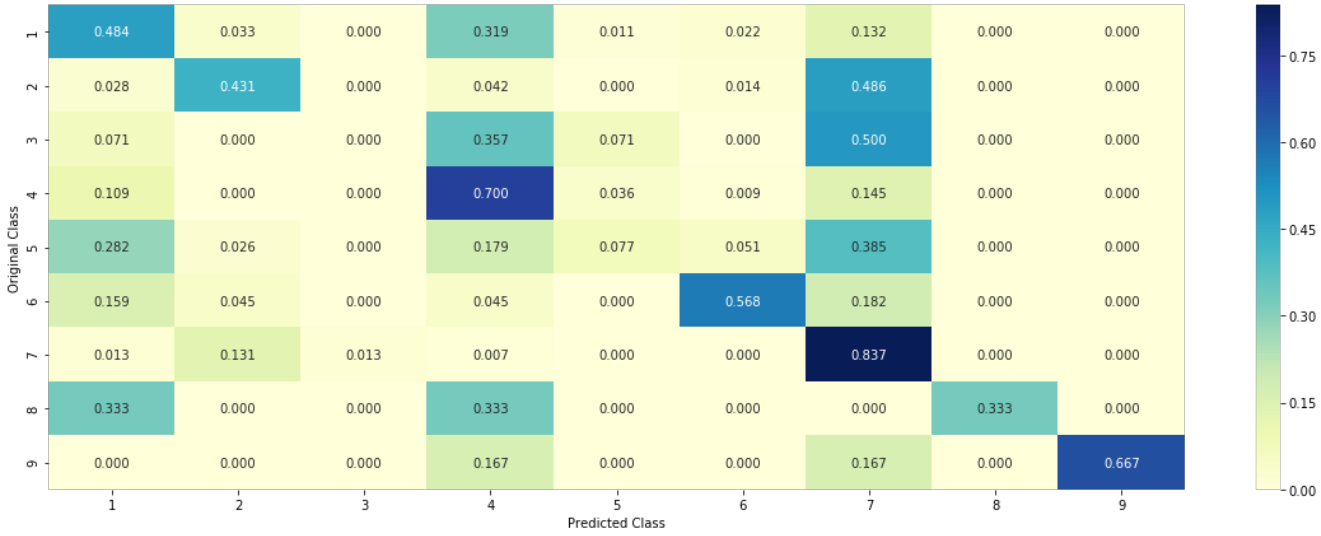
```
Log loss : 1.1848191814169946
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```


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






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



Observations

- The best alpha value is 1000.
- The test log loss by Random Forest is 1.12.
- Number of misclassified points are 0.4116.

Stack the models

In [125]:

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

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

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

sig_clf1.fit(train_x_tfidf, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_tfidf)
)))

sig_clf2.fit(train_x_tfidf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_tfidf))))

sig_clf3.fit(train_x_tfidf, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tfidf))))
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_p
robas=True)
    sclf.fit(train_x_tfidf, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_tfidf))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
    if best_alpha > log_error:
        best_alpha = log_error
```

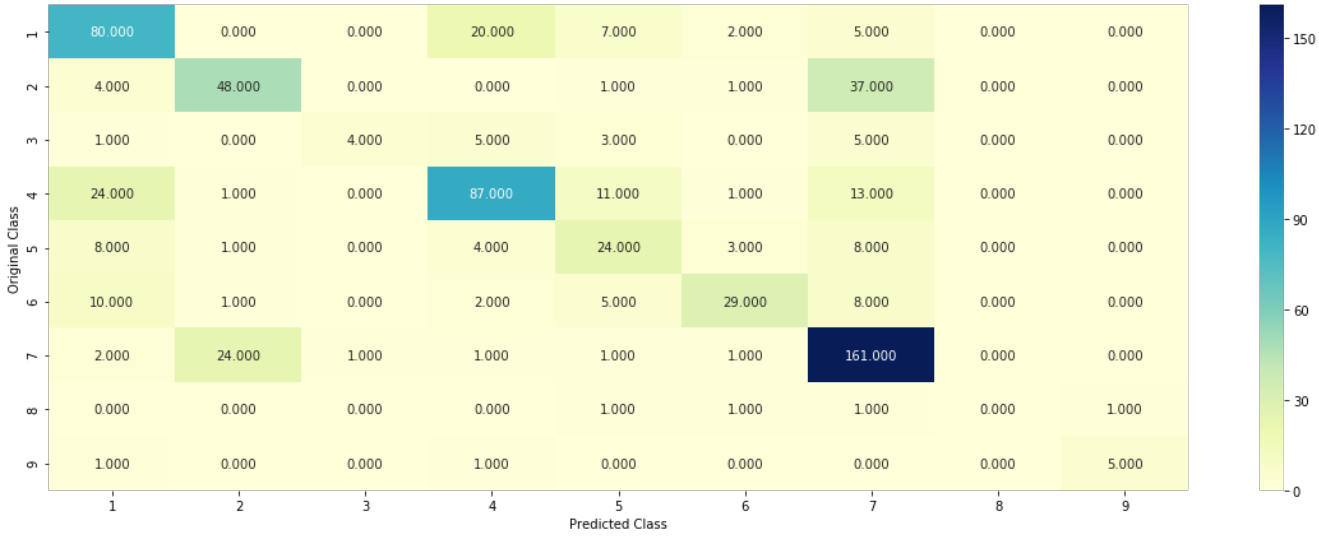
```
Logistic Regression :  Log Loss: 1.08
Support vector machines : Log Loss: 1.47
Naive Bayes : Log Loss: 1.26
---------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.818
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.718
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.325
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.233
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.531
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.924
```

Testing the model with the best hyper parameters

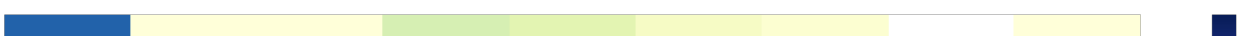
In [126]:

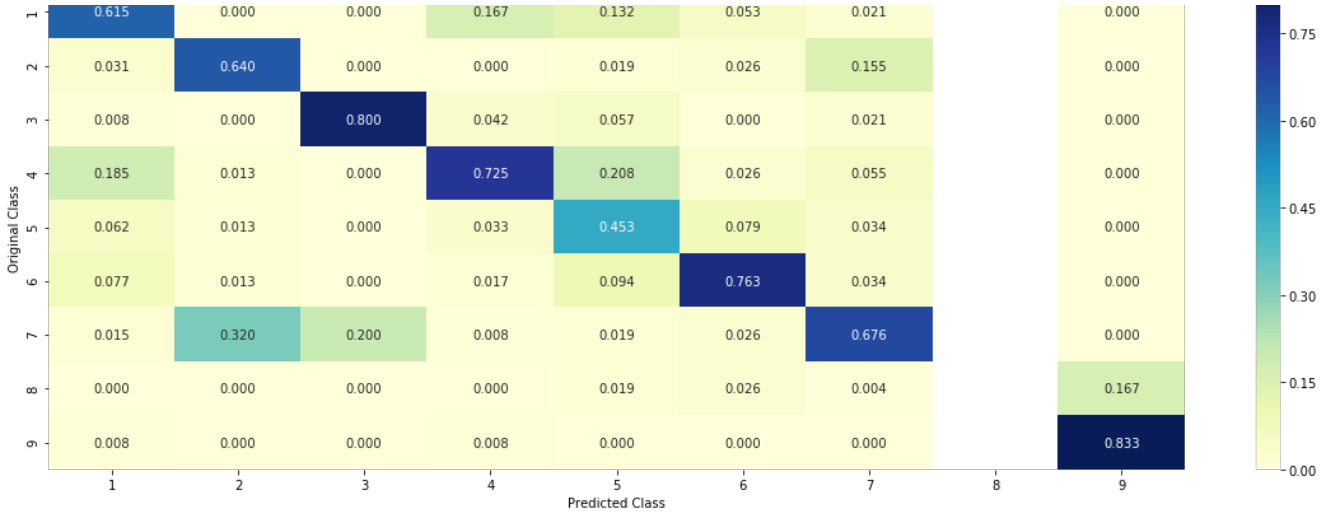
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_tfidf, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.4937010617579535
Log loss (CV) on the stacking classifier : 1.233320153124504
Log loss (test) on the stacking classifier : 1.1280079713846318
Number of missclassified point : 0.34135338345864663
-------------------- Confusion matrix --------------------
```

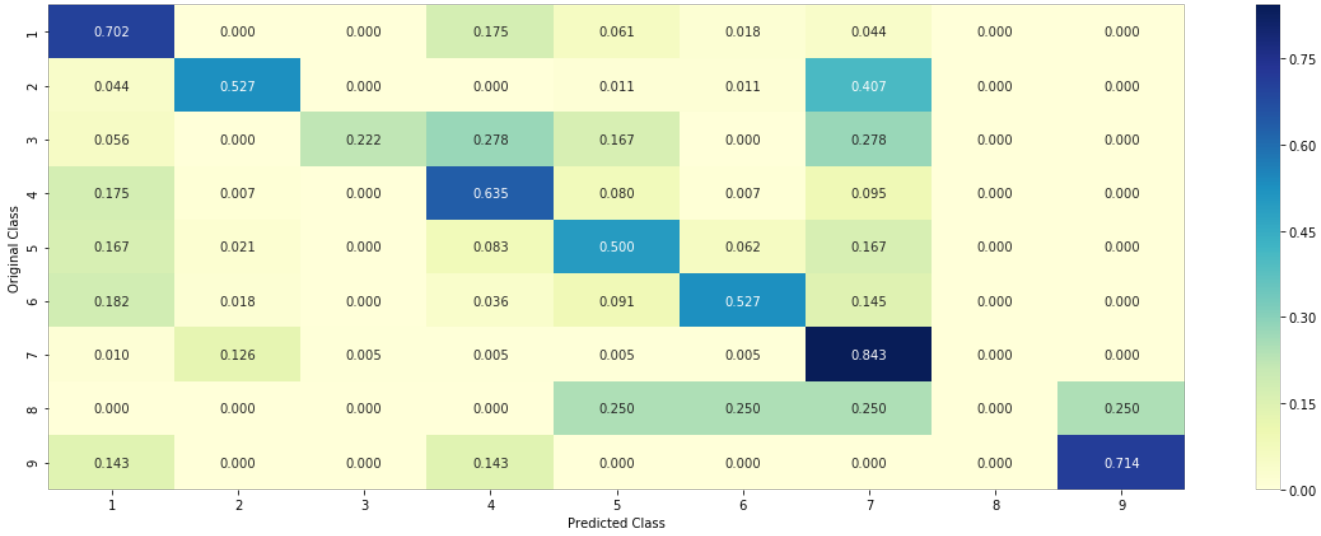



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Observations

- The best alpha value is 0.1.
- The test log loss by Stacking classifier is 1.12.
- Number of misclassified points are 0.3413.

Maximum Voting classifier

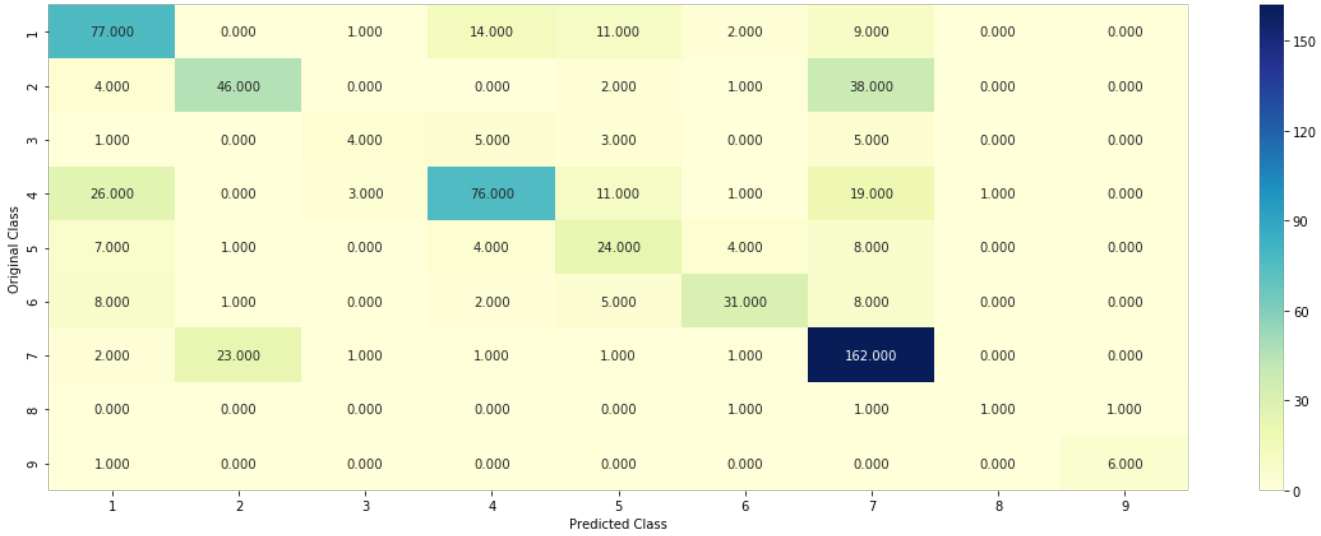
In [127]:

```
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_tfidf, train_y)

print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_tfidf)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_tfidf)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
vclf.predict_proba(test_x_tfidf)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfidf)- test_y))/t
est_y.shape[0])

plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidf))
```

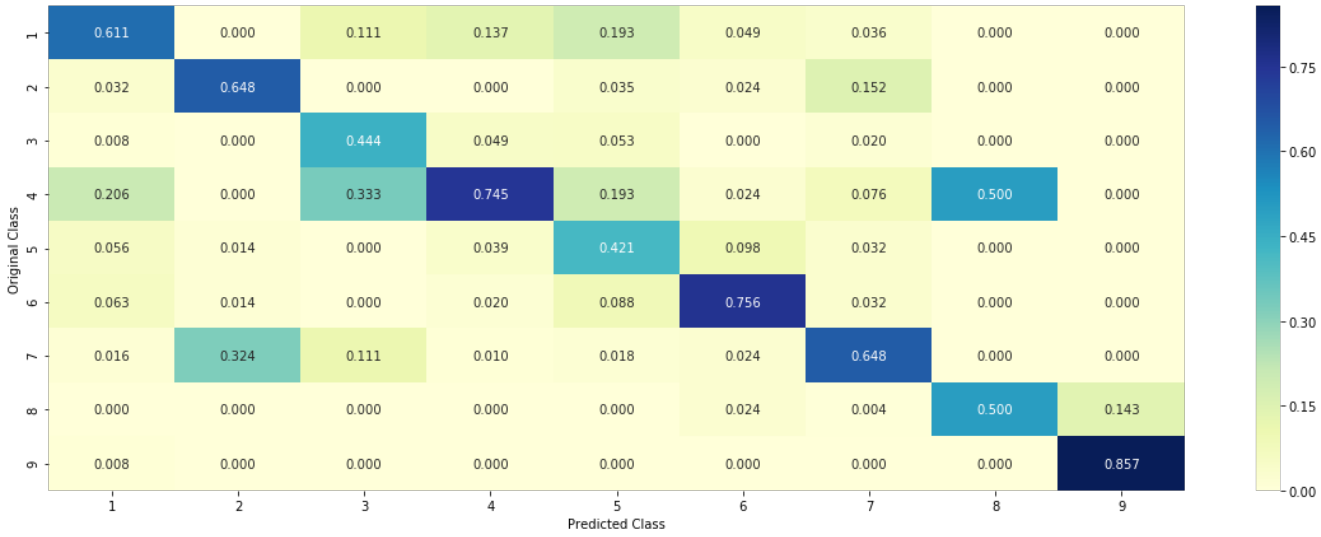
```
Log loss (train) on the VotingClassifier : 0.8162967469202848
Log loss (CV) on the VotingClassifier : 1.1367572286627716
Log loss (test) on the VotingClassifier : 1.1242854209145088
Number of missclassified point : 0.35789473684210527
```

Number of misclassified point : 0.5576947366421052

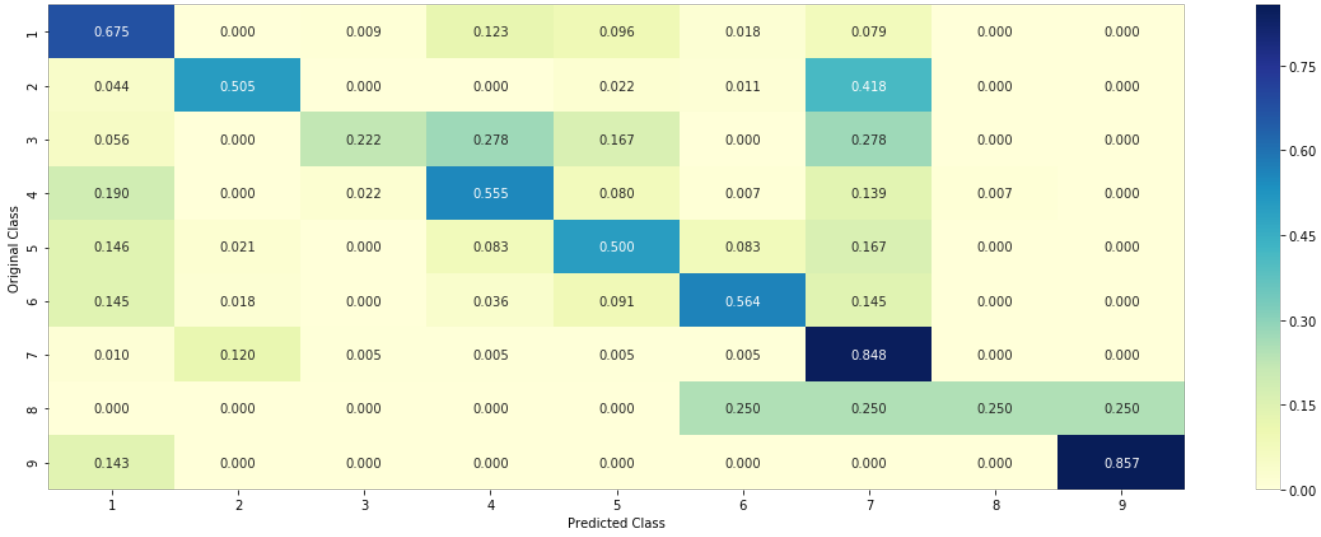
---------- Confusion matrix ----------



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



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



Observations

- The test log loss by Majority classifier is 1.12

- The test log loss by Majority classifier is 1.12.
- Number of misclassified points are 0.3578.

Task-2

Using only top 1000 fetures based on Tf-idf values

In [128]:

```
text_vectorizer = TfidfVectorizer(min_df=3, max_features = 1000)

train_text_feature_tfidf = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_features = text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_tfidf.sum(axis=0).A1

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 [129]:

```
# Normalizing

train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)

test_text_feature_tfidf = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_tfidf = normalize(test_text_feature_tfidf, axis=0)

cv_text_feature_tfidf = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_tfidf = normalize(cv_text_feature_tfidf, axis=0)
```

Merging all the features

In [130]:

```
train_x_tfidf = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf)).tocsr()
train_y = np.array(list(train_df['Class']))

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

cv_x_tfidf = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [131]:

```
# Checking total datapoints in Train, Test and CV

print("(number of data points * number of features) in train data = ", train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", test_x_tfidf.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_tfidf.shape)
```

```
(number of data points * number of features) in train data =  (2124, 3185)
(number of data points * number of features) in test data =  (665, 3185)
(number of data points * number of features) in cross validation data = (532, 3185)
```

Applying Random Forest model

Hyper parameter tuning

In [132]:

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

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

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

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

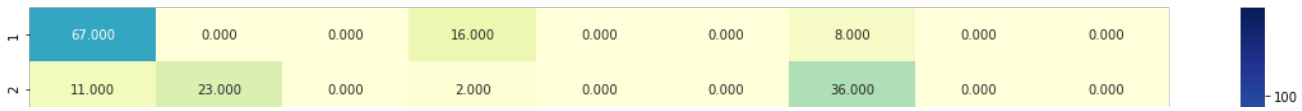
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.260828361133113
for n_estimators = 100 and max depth =  10
Log Loss : 1.266732748705189
for n_estimators = 200 and max depth =  5
Log Loss : 1.2533695262425446
for n_estimators = 200 and max depth =  10
Log Loss : 1.256368665301526
for n_estimators = 500 and max depth =  5
Log Loss : 1.251235630383988
for n_estimators = 500 and max depth =  10
Log Loss : 1.2547795011864793
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2490392126421068
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2532326284257604
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2471987654446224
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2523481923475053
For values of best estimator =  2000 The train log loss is: 0.8632800716702296
For values of best estimator =  2000 The cross validation log loss is: 1.2471987654446224
For values of best estimator =  2000 The test log loss is: 1.1856445832766112
```

Testing model with best hyper parameters (One Hot Encoding)

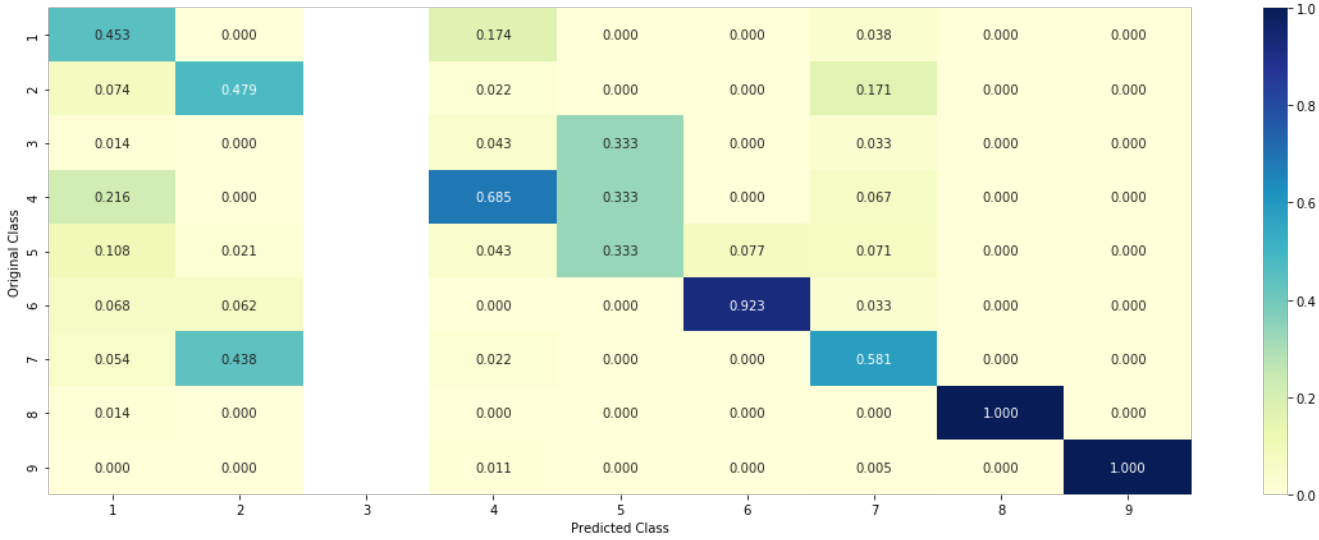
In [133]:

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

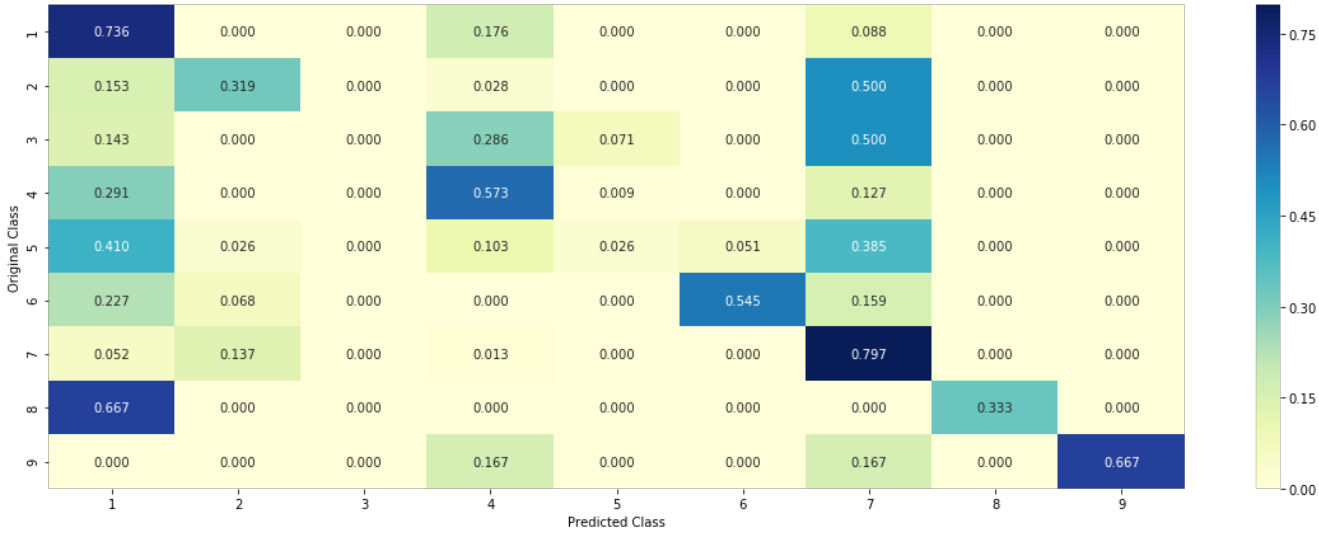
```
Log loss : 1.2471987654446224
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```



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



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



Observations

- The best alpha value is 2000.
- The test log loss by Random Forest is 1.18.
- Number of misclassified points are 0.4266.

Task-3

Task-3

Applying LR with both uni-grams and bi-grams

In [134]:

```
text_vectorizer = CountVectorizer(min_df=3, ngram_range=(1, 2))

train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_features= text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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 : 789389
```

In [135]:

```
# Normalizing

train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

Merging all the features

In [136]:

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

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

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

In [137]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 791574)
(number of data points * number of features) in test data =  (665, 791574)
(number of data points * number of features) in cross validation data = (532, 791574)
```

Applying Logistic Regression with Class Balancing

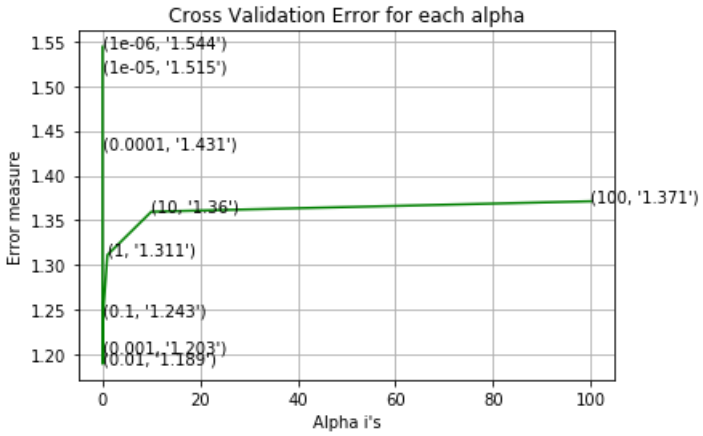
Hyper parameter tuning

In [138]:

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

for i in alpha:
    print("for alpha =", i)
```

```
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best 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 : 1.543939548813537
for alpha = 1e-05
Log Loss : 1.515462677813077
for alpha = 0.0001
Log Loss : 1.4308402415533696
for alpha = 0.001
Log Loss : 1.2025152836307598
for alpha = 0.01
Log Loss : 1.1894389676239001
for alpha = 0.1
Log Loss : 1.2432084375239036
for alpha = 1
Log Loss : 1.3111301978049443
for alpha = 10
Log Loss : 1.3595365979607825
for alpha = 100
Log Loss : 1.3711890568298912
```




```
For values of best alpha =  0.01 The train log loss is: 0.6730755587777723
```

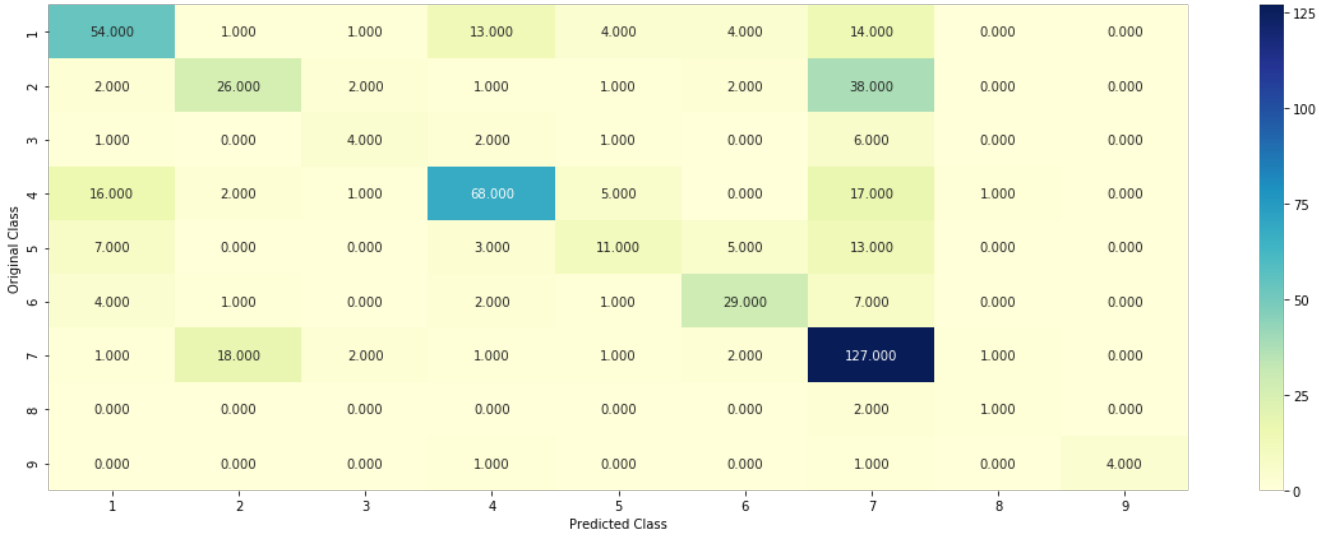
```
For values of best alpha =  0.01 The cross validation log loss is: 1.1894389676239001
For values of best alpha =  0.01 The test log loss is: 1.1352643045663067
```

Testing the model with best hyper paramters

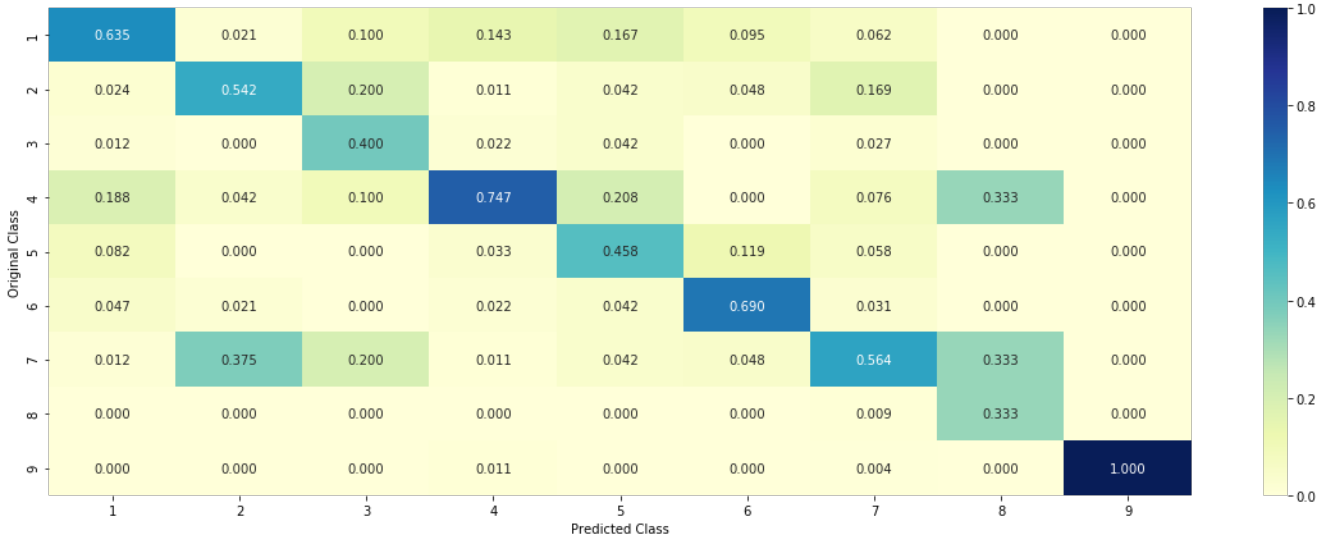
In [139]:

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

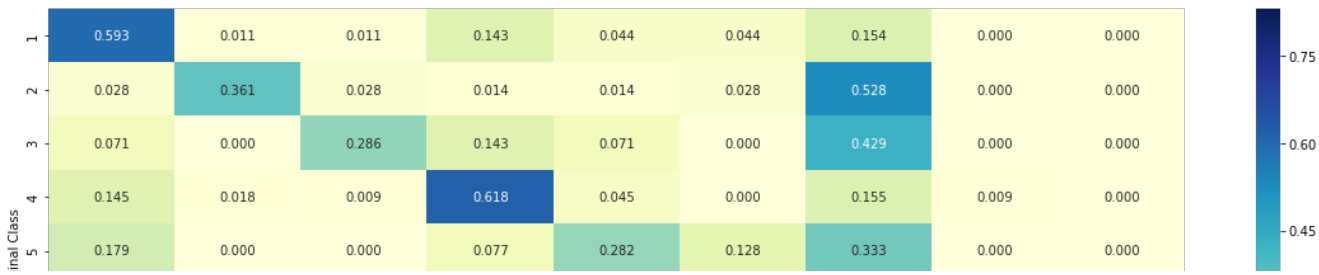
```
Log loss : 1.1894389676239001
Number of mis-classified points : 0.39097744360902253
-------------------- Confusion matrix --------------------
```

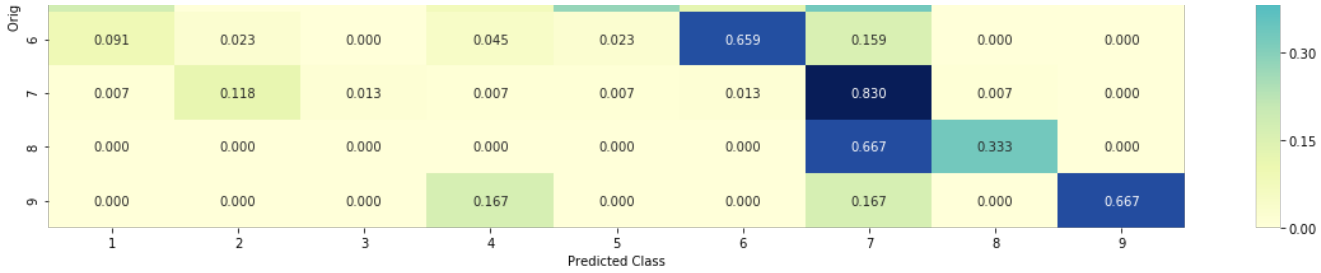



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





Observations

- The best alpha value is 0.01.
- The test log loss by Logistic Regression is 1.13.
- Number of misclassified points are 0.3909.

Task 4

Using any feature engineering technique to reduce the log-loss

In [149]:

```
# At first I have have tried Topic extraction for feature engineering but I didn't get
satisfactory results
# https://medium.com/analytics-vidhya/personalized-cancer-diagnosis-3d6f09a6b8c9

text_vectorizer = CountVectorizer(min_df=3, ngram_range=(1, 4), max_features=2000)

train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_features= text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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 : 2000
```

In [150]:

```
# Normalizing

train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

In [151]:

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

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

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

In [152]:

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

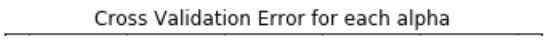
```
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding
.shape)
```

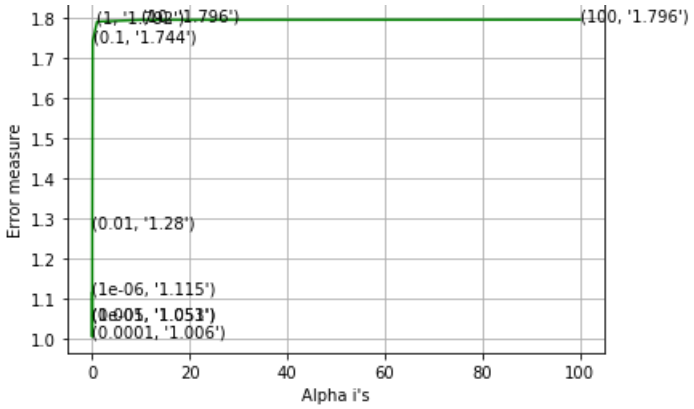
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 4185)
(number of data points * number of features) in test data =  (665, 4185)
(number of data points * number of features) in cross validation data = (532, 4185)
```

In [153]:

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

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

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

for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best 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 : 1.1153880194959058
for alpha = 1e-05
Log Loss : 1.0514235281766164
for alpha = 0.0001
Log Loss : 1.0058467708299583
for alpha = 0.001
Log Loss : 1.0525565939054393
for alpha = 0.01
Log Loss : 1.2795530875185093
for alpha = 0.1
Log Loss : 1.7438073084429748
for alpha = 1
Log Loss : 1.7916924305648578
for alpha = 10
Log Loss : 1.7958845811502266
for alpha = 100
Log Loss : 1.7963241371379632
```



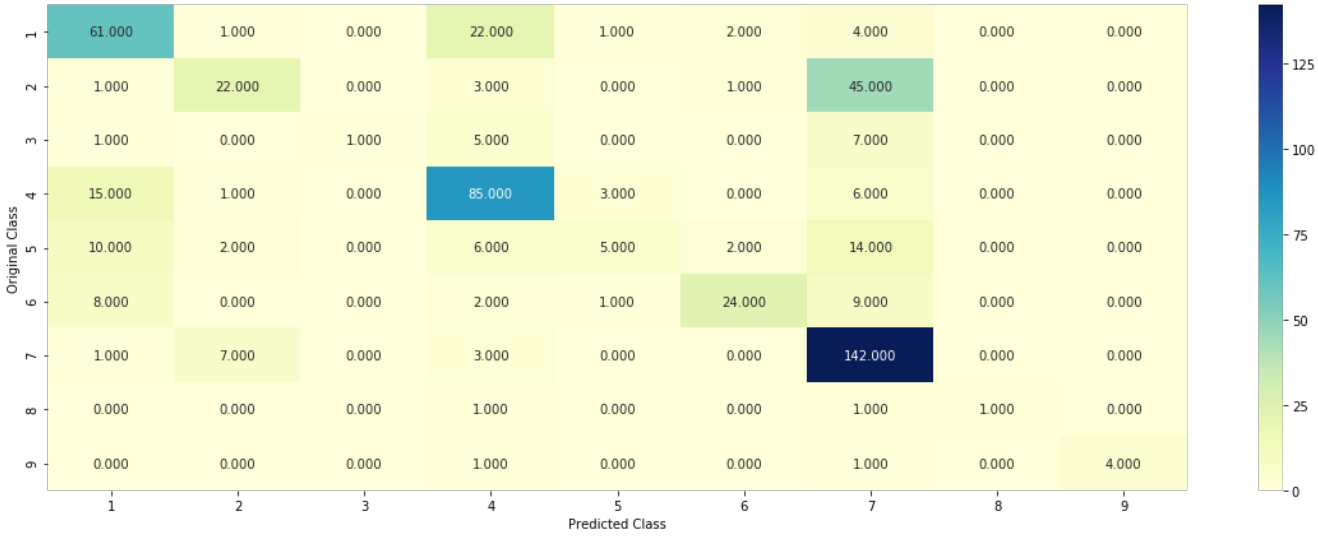



```
For values of best alpha =  0.0001 The train log loss is: 0.39124874063290827
For values of best alpha =  0.0001 The cross validation log loss is: 1.0058467708299583
For values of best alpha =  0.0001 The test log loss is: 0.965515596120808
```

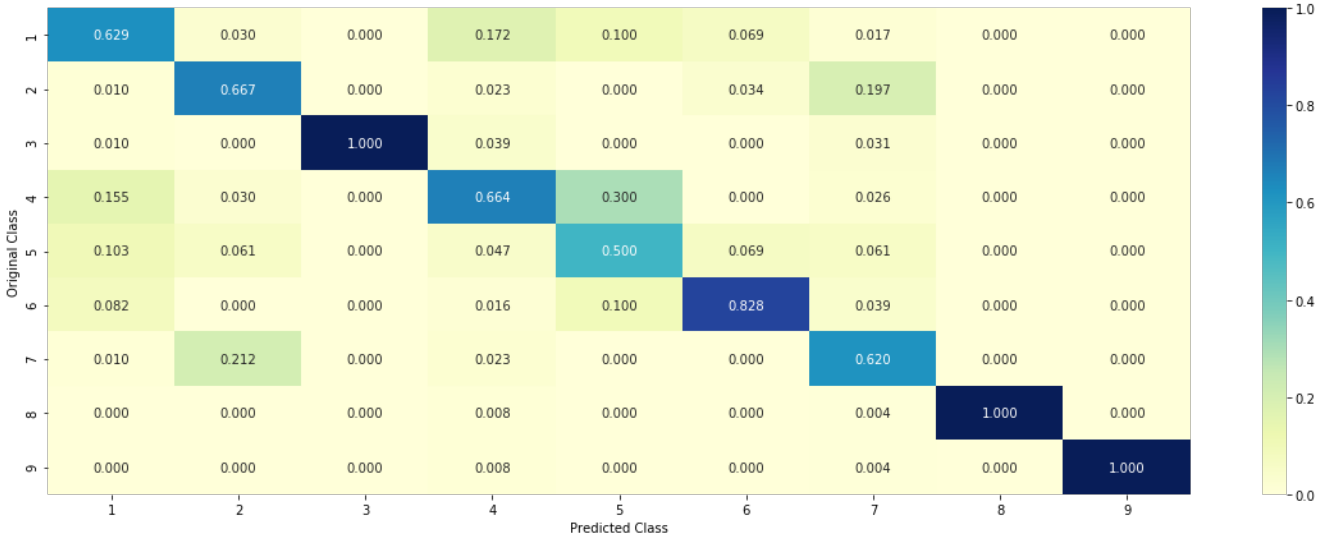
In [154]:

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

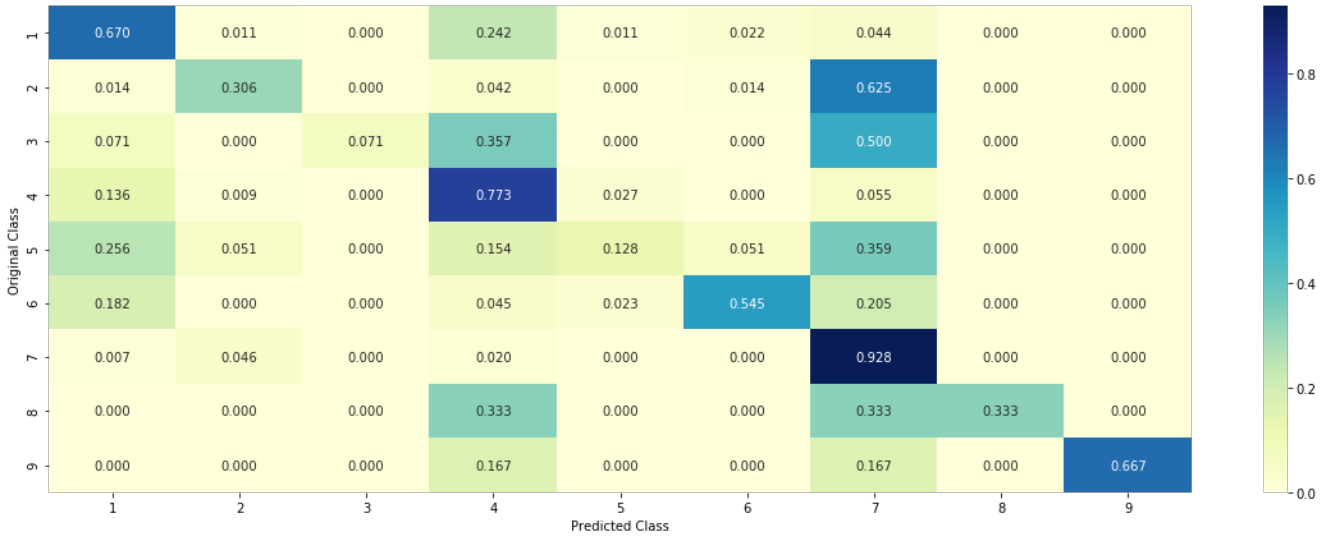
```
Log loss : 1.0058467708299583
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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



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



Observations

- The best alpha value is 0.0001.
- The test log loss by Logistic Regression is 0.965.
- Number of misclassified points are 0.3515.

Showing results with the Preety table

In [2]:

```
from prettytable import PrettyTable

pt = PrettyTable()
pt.field_names = ['No.', 'Model Name', 'Text Technique', 'Test Log-Loss', 'Misclassified Points']
pt.add_row(["0","Random", "BoW", "2.47", "-"])
pt.add_row(["1","Naive Bayes", "BoW", "1.27", "39.28%"])
pt.add_row(["2","KNN", "BoW", "1.07", "38.34%"])
pt.add_row(["3","LR with class balancing", "BoW", "1.04", "37.03%"])
pt.add_row(["4","LR without class balancing", "BoW", "1.05", "37.21%"])
pt.add_row(["5","Linear SVM", "BoW", "1.11", "36.27%"])
pt.add_row(["6","RF with onehot encoding", "BoW", "1.14", "40.22%"])
pt.add_row(["7","RF with response coding", "BoW", "1.30", "55.82%"])
pt.add_row(["8","Stacking Classifier", "BoW", "1.12", "33.83%"])
pt.add_row(["9","Maximum voting Classifier", "BoW", "1.17", "34.58%"])

pt.add_row(["\n","\n", "\n", "\n", "\n"])

pt.add_row(["10","Naive Bayes", "Tf-Idf", "1.18", "42.85%"])
pt.add_row(["11","KNN", "Tf-Idf", "1.26", "45.67%"])
pt.add_row(["12","LR with class balancing", "Tf-Idf", "1.01", "37.03%"])
pt.add_row(["13","LR without class balancing", "Tf-Idf", "1.02", "36.09%"])
pt.add_row(["14","Linear SVM", "Tf-Idf", "1.07", "36.84%"])
pt.add_row(["15","RF with onehot encoding", "Tf-Idf", "1.12", "41.16%"])
pt.add_row(["16","Stacking Classifier", "Tf-Idf", "1.12", "34.13%"])
pt.add_row(["17","Maximum voting Classifier", "Tf-Idf", "1.12", "35.78%"])

pt.add_row(["\n","\n", "\n", "\n", "\n"])

pt.add_row(["18","RF", "Tf-Idf with top 1000 ft", "1.18", "42.66%"])
pt.add_row(["19","LR with unigram & Bigram", "BoW", "1.13", "39.09%"])
pt.add_row(["20","LR", "Feature Engineering", "0.965", "35.15%"])

print(pt)
```

```
+-----+---------------------------+-------------------------+---------------+---------------------
| No. |         Model Name        |       Text Technique    | Test Log-Loss | Misclassified Points
```

No.	Model Name	Text Technique	Test Log Loss	Misclassified Points
0	Random	BoW	2.47	-
1	Naive Bayes	BoW	1.27	39.28%
2	KNN	BoW	1.07	38.34%
3	LR with class balancing	BoW	1.04	37.03%
4	LR without class balancing	BoW	1.05	37.21%
5	Linear SVM	BoW	1.11	36.27%
6	RF with onehot encoding	BoW	1.14	40.22%
7	RF with response coding	BoW	1.30	55.82%
8	Stacking Classifier	BoW	1.12	33.83%
9	Maximum voting Classifier	BoW	1.17	34.58%
10	Naive Bayes	Tf-Idf	1.18	42.85%
11	KNN	Tf-Idf	1.26	45.67%
12	LR with class balancing	Tf-Idf	1.01	37.03%
13	LR without class balancing	Tf-Idf	1.02	36.09%
14	Linear SVM	Tf-Idf	1.07	36.84%
15	RF with onehot encoding	Tf-Idf	1.12	41.16%
16	Stacking Classifier	Tf-Idf	1.12	34.13%
17	Maximum voting Classifier	Tf-Idf	1.12	35.78%
18	RF	Tf-Idf with top 1000 ft	1.18	42.66%
19	LR with unigram & Bigram	BoW	1.13	39.09%
20	LR	Feature Engineering	0.965	35.15%

Precedure Followed

Problem Statement:

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

Step 1 : We have 2 files, one file is having ID, Gene, Variation, Class and the other file is having ID, text. So first I have cleaned the text data in the second file so that I can conbine both the files based on ID. Based on these features we need to predict in which class a particular datapoint belongs to.

Step 2 : After Merging both the files now we have a final file which contains ID, Gene, Variation, Class, Text. Now I have checked the type of my dataset like is it balanced data or Imbalanced data. I have checked that through checking the distribution of classes. I get to know that the dataset we have is imbalanced dataset. dataset have Classes 7, 4, 1 and 2 as majority classes and classes 3, 4, 8, 9 as minority classes.

Step 3 : Before procedding further we need to check the worst log loss that our model can have. We can check the worst log loss by building a Random model. The worst logloss achieved on Random model is 2.47.

Step 4 : Now since we have very few features I have performed Univariate Analysis on each feature to know that how each feature perform in predicting the class Label. According to the results that I got Gene and Text features are very important features because by using them only we can see the significant decrease in Log loss. variation feature is usefull but not as much as compare to other features.

Step 5 : After doing Univariate Analysis I have applied different Machine learning algorithms such as Naive Bayes, KNN, Logistic Regression, Random Forest, etc. While applying models I have checked the multiclass log loss that I have achieved by that model

and along with that I am also printing the No. of misclassified points. To get better understanding I have also plotted Confusion Matrix, Recall Matrix, Precision Matrix.

Step 6 : Among all the models that I have tried I have got best results in Logistic Regression (Multiclass log loss - 1.04)

Step 7 : We also need to take care of the Interpretability, So while building machine learning model I have also showed why my model is classifing the particular point to that class. In this problem interpretability matters a lot because the researcher can get to know why our model is predicting this specific output.

Step 8 : Now After applying BoW we have tried to use the Tf-Idf Vectorizer instead of BagOfWords to see the changes we get in our outcome. From the above Preetytabel we can see that the Models that I have built using Tf-Idf Vectorizer have better log loss as compare to model with BoW. But In Tf-Idf models the No. of misclassified points are slightly more than BoW models.

Step 9 : To reduce the log loss I have tried to use only top 1000 features based on their IDF values then I have also tried to use both Uni-grams and Bi-grams to see what results I am getting. But the results are not quite satisfactory.

Step 10 : At last I needed to reduce the log loss less than 1.0. At first I have tried Topic Extraction but I dod not get log loss below 1.0. So to achieve log loss less than 1 I have set the ngram range to (1, 4) which means it will consider Uni-gram, Bi-gram, Tri-gram, 4-gram. If we increase the number of words while converting text into matrix then by increasing we might get more insight that Uni-gram. And I have also selected top 2000 features. Finally I got the Log loss of 0.965.