In [ ]:
```
import os
```

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import re
import time
import warnings
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression

from nltk.corpus import stopwords
```

In [6]:
```
data = pd.read_csv('msk-redefining-cancer-treatment/training_variants')
print('NO. of data points in the dataset',data.shape[0])
print('No. of features',data.shape[1])
print('features',data.columns.values)
data.head()
```

```
NO. of data points in the dataset 3321
No. of features 4
features ['ID' 'Gene' 'Variation' 'Class']
```

Out[6]:

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

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

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

READING TEXT DATA

In [7]:
```
data_text = pd.read_csv('msk-redefining-cancer-treatment/training_text'
,sep = "\|\|",engine = 'python',names = ['ID','TEXT'],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[7]:

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

PREPROCESSING OF TEXT

In [8]:

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


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

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

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

In [9]:
```
from tqdm import tqdm
```

In [10]:
```
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_tim
e, "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 : 183.65417742300278 seconds
```

In [11]:
```
result = pd.merge(data,data_text,on='ID',how= 'left')
result.head()
```

Out[11]:

	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...

	ID	Gene	Variation	Class	TEXT
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

In [12]:
```
result[result.isnull().any(axis=1)]
```

Out[12]:

	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 [13]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +" "+result
['Variation']
```

In [14]:
```
result[result['ID']== 1109]
```

Out[14]:

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

Test, Train and Cross Validation Split

In [15]:
```
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, st
ratify=y_true, test_size=0.2)
```

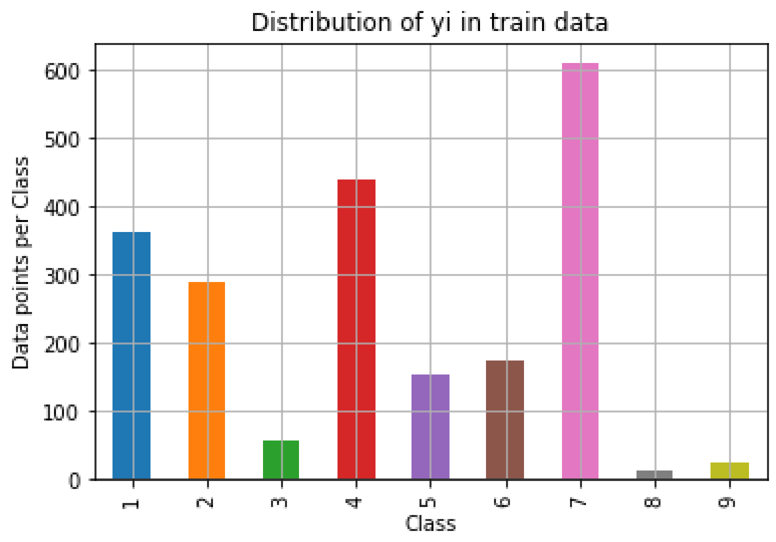
```
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, str
atify=y_train, test_size=0.2)
```

In [16]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
])
```

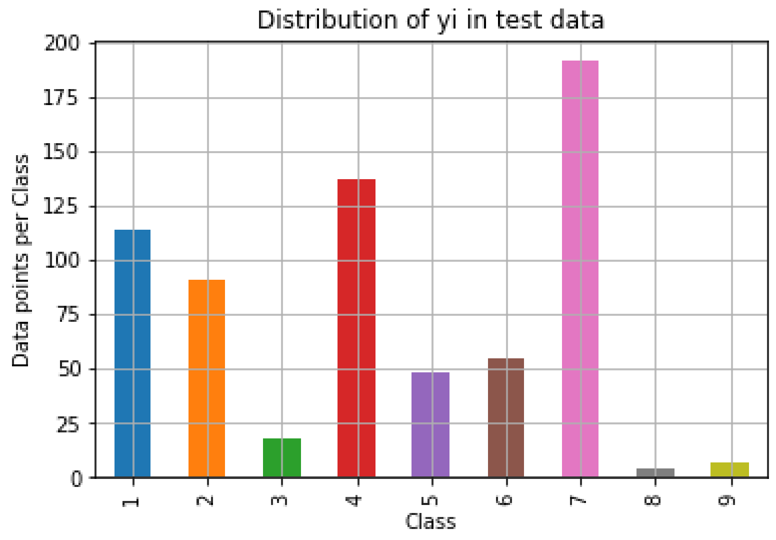
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

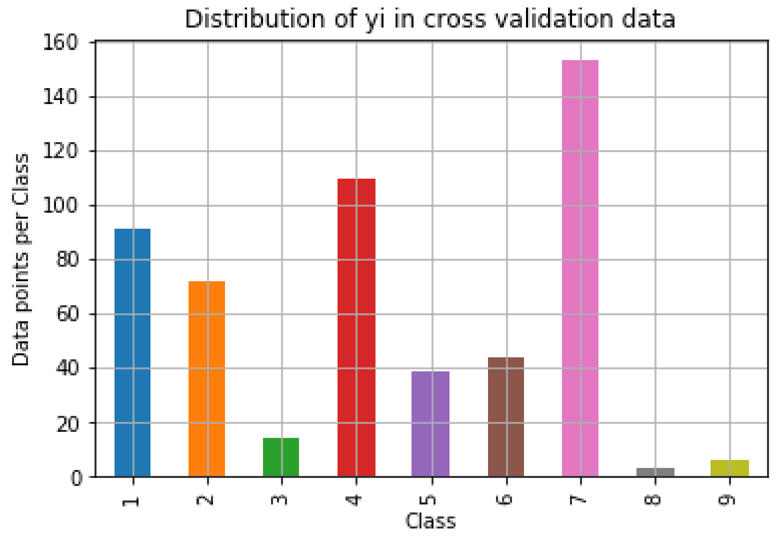
In [17]:
```
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()
my_colors = 'rgbkymc'
train_class_distribution.plot(kind = 'bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
sorted_yis = np.argsort(-train_class_distribution.values)
for i in sorted_yis:

    print('Number of data points in class', i+1, ':',train_class_distri
bution.values[i], '(', np.round((train_class_distribution.values[i]/tra
in_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/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distrib
ution.values[i], '(', np.round((test_class_distribution.values[i]/test_
df.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distribut
ion.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.sha
pe[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 %)
```

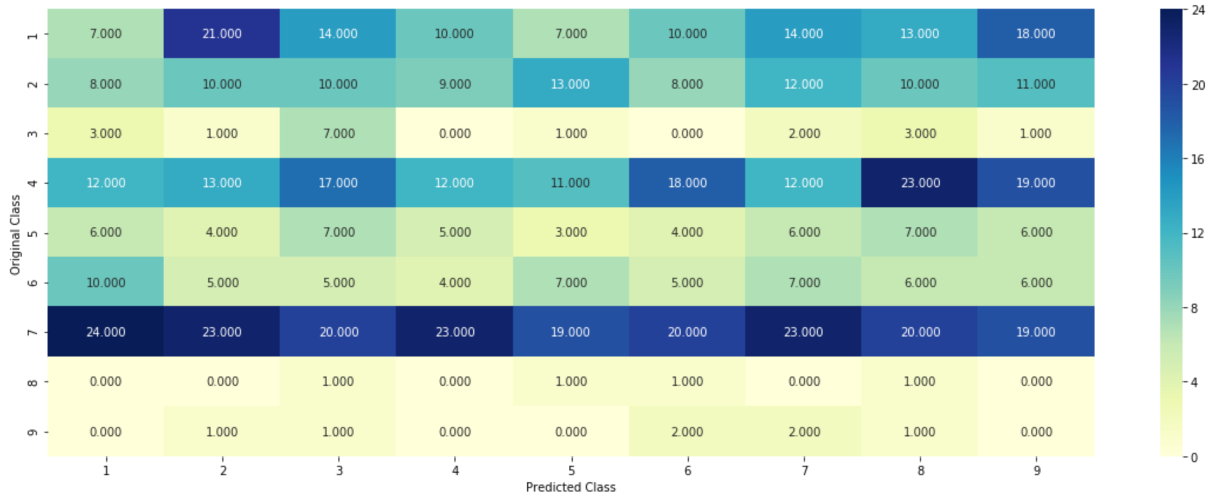
In [18]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y,predict_y)
    A = (((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize = (20,7))
    sns.heatmap(C,annot = True,cmap= "YlGnBu",fmt=".3f", xticklabels=la
bels, yticklabels=labels)
```

```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

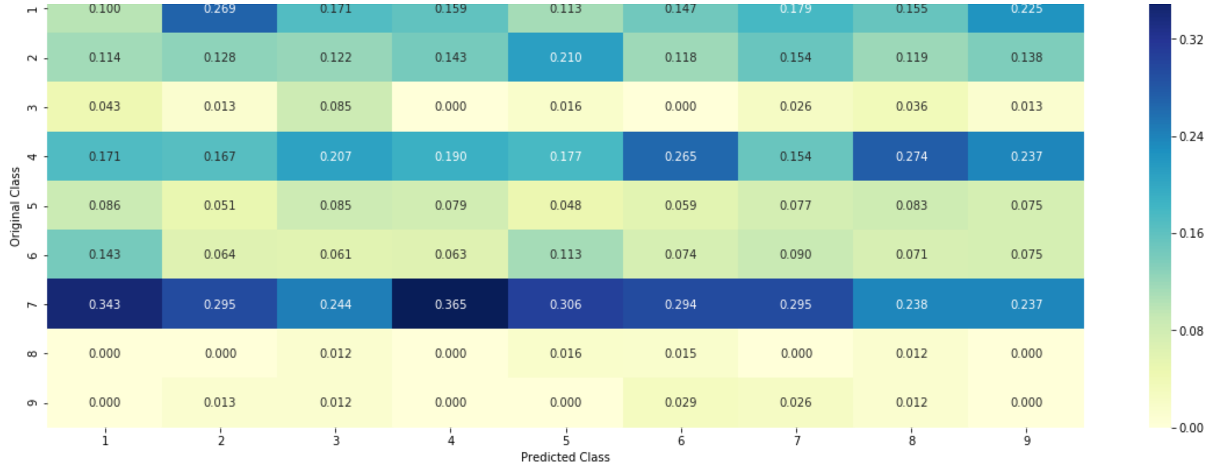
In [19]:

```
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_p
redicted_y, eps=1e-15))
predicted_y = np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

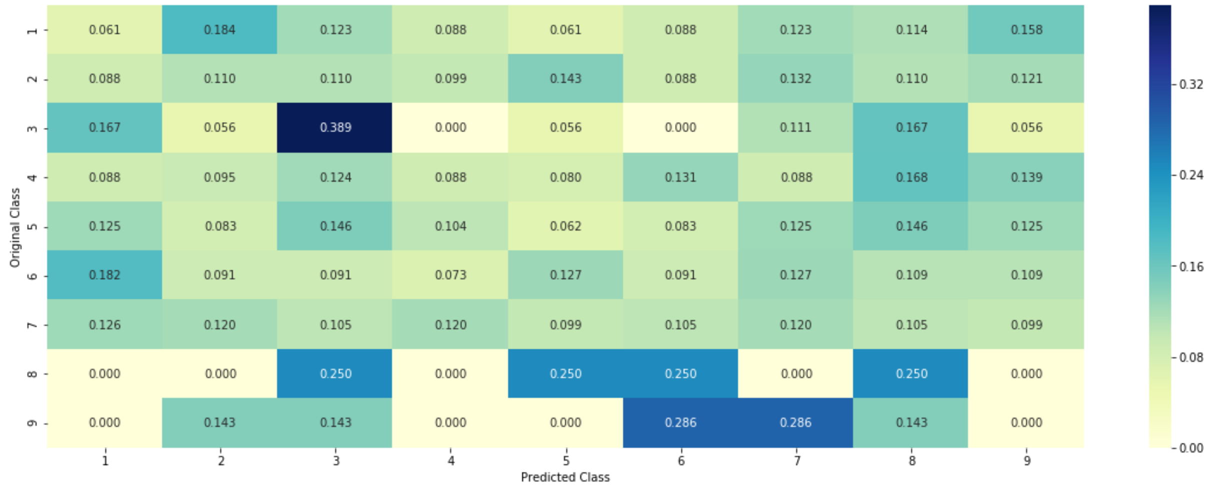
```
Log loss on Cross Validation Data using Random Model 2.5031563378690067
Log loss on Test Data using Random Model 2.4707563896965534
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [20]:
```
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
```

```
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability arr
ay for each gene/variation
    gv_dict = dict()

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.200757575757575757, 0.03787878787878788, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.068181818181818177, 0.068181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.13939393939393939394, 0.345454545454
```

```
54546, 0.060606060606060608, 0.060606060606060608, 0.06060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.06289308176100062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

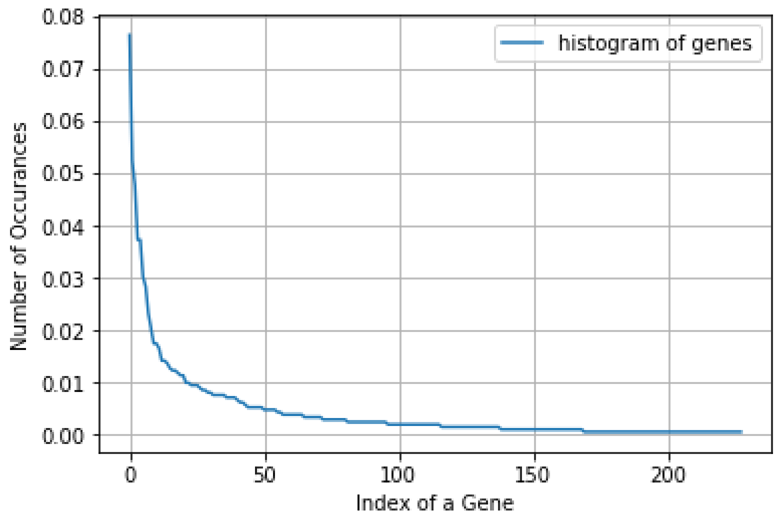
Ans. Gene is a categorical variable

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

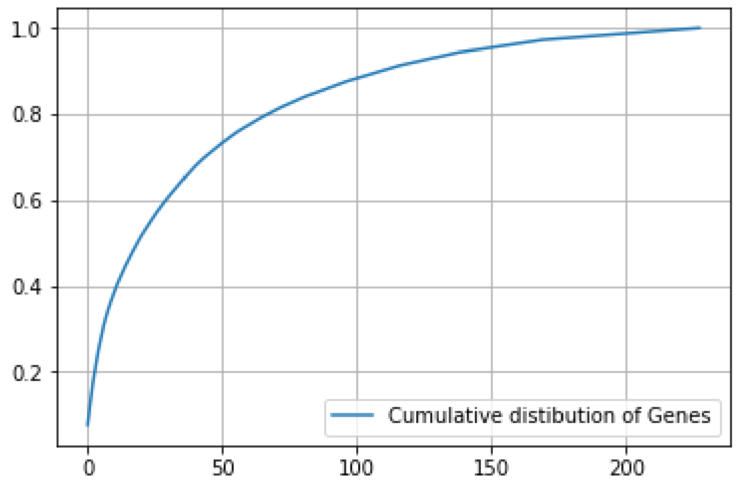
In [21]:
```
unique_genes = train_df['Gene'].value_counts()
print('No.of Unique Genes',unique_genes.shape[0])
print(unique_genes.head(10))
```

```
No.of Unique Genes 228
BRCA1     162
TP53      111
EGFR      101
PTEN       79
BRCA2      79
BRAF       64
KIT        60
ALK        49
ERBB2      43
PDGFRA     37
Name: Gene, dtype: int64
```

In [22]:
```
s = sum(unique_genes.values)
h = unique_genes.values/s
plt.plot(h,label = "histogram of genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [23]:
```python
c = np.cumsum(h)
plt.plot(c,label = 'Cumulative distibution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

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

In [24]:
```
alpha = 1
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha,"Gen
e",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 [25]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature: (2124, 9)
```

In [30]:
```
gene_vectorizer = TfidfVectorizer(ngram_range=(1,1),max_features = 1000
)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [31]:
```
train_df['Gene'].head()
```

Out[31]:
```
155       EGFR
3028       KIT
3317     RUNX1
2365     STK11
981       ETV6
Name: Gene, dtype: object
```

In [32]:
```
train_gene_feature_onehotCoding
```

Out[32]:
```
<2124x227 sparse matrix of type '<class 'numpy.float64'>'
        with 2124 stored elements in Compressed Sparse Row format>
```

In [29]:
```
len(gene_vectorizer.get_feature_names())
```

Out[29]:
```
227
```

In [33]:
```
print("train_gene_feature_onehotCoding is converted feature using one-h
ot encoding method. The shape of gene feature:", train_gene_feature_one
hotCoding.shape)
```

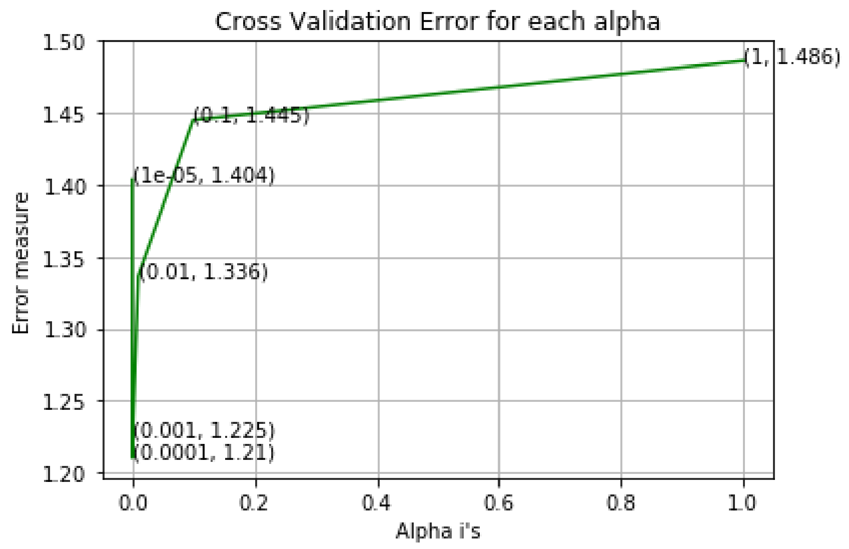
```
train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature: (2124, 227)
```

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 [34]:
```
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array = []
for i in alpha:
    clf = SGDClassifier(alpha = i ,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.clas
ses_,eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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 vali
dation log loss is:",log_loss(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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.403502216534151
For values of alpha =  0.0001 The log loss is: 1.2100040953883213
For values of alpha =  0.001 The log loss is: 1.2251820081190452
For values of alpha =  0.01 The log loss is: 1.3364242873399734
For values of alpha =  0.1 The log loss is: 1.4450994742490895
For values of alpha =  1 The log loss is: 1.4862833586858324
```



```
For values of best alpha =  0.0001 The train log loss is: 1.04282048645
1529
For values of best alpha =  0.0001 The cross validation log loss is: 1.
2100040953883213
For values of best alpha =  0.0001 The test log loss is: 1.201503071307
5001
```

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 [35]:
```
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']))))].shap
e[0]
```

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

```
Q6. How many data points in Test and CV datasets are covered by the  22
8  genes in train dataset?
Ans
1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 512 out of  532 : 96.2406015037594
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

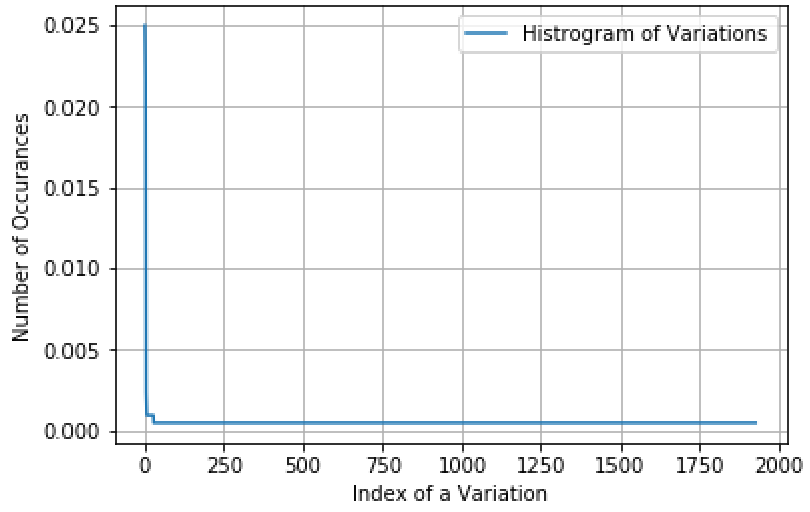
In [36]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1927
Truncating_Mutations    53
Deletion                50
Amplification           46
Fusions                 24
Overexpression           5
Q61H                     3
ETV6-NTRK3_Fusion        2
TMPRSS2-ETV1_Fusion      2
K117N                    2
T286A                    2
Name: Variation, dtype: int64
```

In [37]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

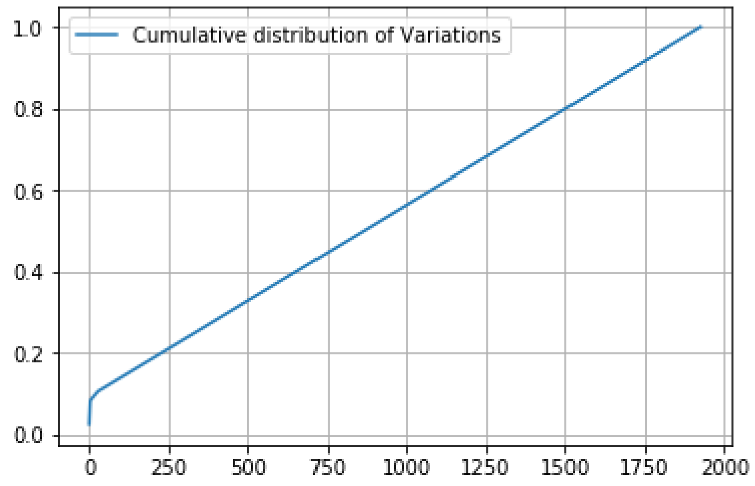
```
Ans: There are 1927 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [38]:
```
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 [39]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02495292 0.04849341 0.07015066 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

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

In [40]:
```
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
```

```
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [41]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [42]:
```
variation_vectorizer = TfidfVectorizer(ngram_range=(1,1),max_features =
 1000)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [43]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature: (2124, 1000)
```

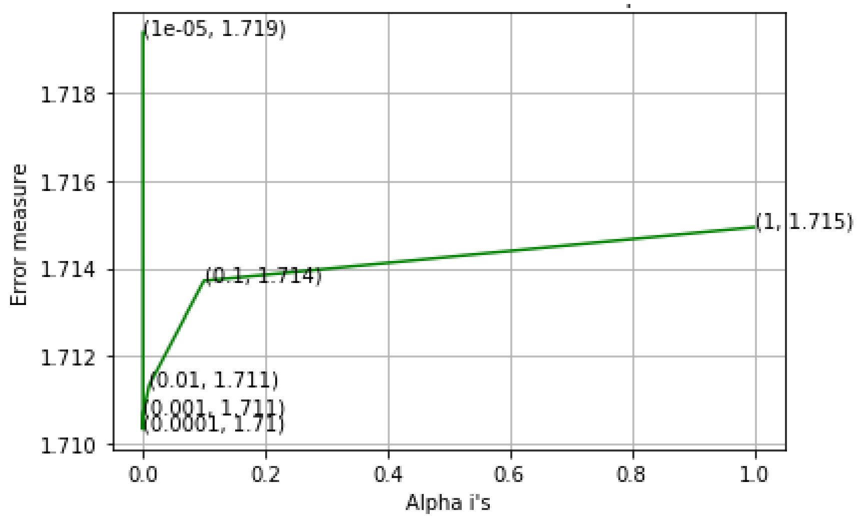
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [44]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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



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

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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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 vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7193557255651712
For values of alpha =  0.0001 The log loss is: 1.7103329698301422
For values of alpha =  0.001 The log loss is: 1.7107265487895786
For values of alpha =  0.01 The log loss is: 1.7113321358061389
For values of alpha =  0.1 The log loss is: 1.713712745763189
For values of alpha =  1 The log loss is: 1.714935005465305
```

Cross Validation Error for each alpha



```
For values of best alpha =  0.0001 The train log loss is: 1.21675153078
4613
For values of best alpha =  0.0001 The cross validation log loss is: 1.
7103329698301422
For values of best alpha =  0.0001 The test log loss is: 1.701362186409
7355
```

In [45]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
]))))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1927  genes in test and
cross validation data sets?
Ans
1. In test data 68 out of 665 : 10.225563909774436
2. In cross validation data 55 out of  532 : 10.338345864661653
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

In [46]:
```
def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index,rows in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word]+=1
    return dictionary
```

In [47]:
```
import math
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 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [49]:
```
text_vectorizer = TfidfVectorizer(min_df=3,max_features =1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
```

```
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_count
s))
print("Total number of unique words in train data :", len(train_text_fe
atures))
```

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

In [50]:
```
from collections import Counter, defaultdict

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

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


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

In [51]:
```
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 [52]:
```
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_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [53]:
```
from sklearn.preprocessing import normalize

train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEX
T'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCodi
ng, axis=0)

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

In [54]:
```
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 [55]:
```
print(Counter(sorted_text_occur))
```

```
Counter({248.52427606909612: 1, 177.86722878750243: 1, 131.488853530085
46: 1, 129.5360673563795: 1, 126.81278421734666: 1, 116.0057907003641:
1, 115.22798752464622: 1, 115.2039365863168: 1, 109.30258617773265: 1,
107.95775329085768: 1, 105.91194712849071: 1, 91.89526232408748: 1, 89.
09406161826489: 1, 88.90102761036763: 1, 86.39684691421618: 1, 79.68458
481863323: 1, 79.06078288408965: 1, 78.25832894431927: 1, 78.0248185532
9442: 1, 77.49435317351258: 1, 76.21032363017056: 1, 73.41906921894395:
1, 69.84754657894561: 1, 69.72168896427023: 1, 67.35833156446581: 1, 6
```

```
7.3354140267106: 1, 65.83788989461704: 1, 64.46946335307234: 1, 62.7743
2957426946: 1, 62.56642619141566: 1, 62.257364508352936: 1, 61.95654168
702442: 1, 61.80617197287956: 1, 59.11522333470696: 1, 58.1312999295983
1: 1, 56.11310895686478: 1, 55.92662020616207: 1, 55.407941713931585:
1, 54.47624428287909: 1, 52.99921078575137: 1, 52.02454030500826: 1, 5
0.38318217783756: 1, 49.76723404586894: 1, 49.211439032332336: 1, 46.09
40104262259: 1, 45.89715396519147: 1, 45.71173094619685: 1, 44.99995974
418838: 1, 44.57835346697158: 1, 44.12983069536676: 1, 43.9684787374322
4: 1, 43.7165219766933: 1, 43.57229876830894: 1, 42.74515895030914: 1,
42.67237185052187: 1, 42.36812967450439: 1, 42.1022862978819 2: 1, 41.96
120240352003: 1, 41.746996933800055: 1, 41.32069244985428: 1, 41.124681
60422162: 1, 41.006627427996555: 1, 40.7801072027467: 1, 40.00112263655
4854: 1, 39.43664629395642: 1, 39.282200107658866: 1, 39.17943305174948
4: 1, 38.96274688279646: 1, 38.45626516994032: 1, 38.35923482663256: 1,
38.32045886894367: 1, 38.170400819962: 1, 37.93675712515078: 1, 37.4573
6286296036: 1, 37.39432665383861: 1, 37.13554341628058: 1, 36.990864087
91708: 1, 36.932441707156535: 1, 36.513803811001196: 1, 35.496958318638
576: 1, 35.376441387264364: 1, 35.07832141011283: 1, 35.06150934438142:
1, 34.989230088985245: 1, 34.92127200331075: 1, 34.70354129464218: 1, 3
4.53372303521881: 1, 34.44870988562202: 1, 33.572353542074055: 1, 33.50
815372343455: 1, 33.48449147305569: 1, 33.47560529089247: 1, 33.4234972
92277105: 1, 33.12961254310747: 1, 32.16638547386928: 1, 32.10970072292
97: 1, 32.05489320790616: 1, 31.986607637697986: 1, 31.94435149444447:
1, 31.934888803719765: 1, 31.876474677614233: 1, 31.441233668334664: 1,
31.417282859186297: 1, 31.407185421283256: 1, 31.37741902886808: 1, 31.
30477181587594: 1, 31.28398053016766: 1, 31.17143570607263: 1, 31.13242
498783721: 1, 31.015364031697825: 1, 30.814488059120603: 1, 30.77461928
3325087: 1, 30.52915949265452: 1, 30.368660990370895: 1, 30.25712686386
4556: 1, 30.226037534146446: 1, 30.01864210270758: 1, 29.77084220575673
6: 1, 29.659399210994273: 1, 29.61145676330718: 1, 29.611238971637313:
1, 29.580924007897874: 1, 29.326727281569802: 1, 29.190284758854588: 1,
29.02456245879258: 1, 28.98210220834235: 1, 28.958284438604405: 1, 28.8
6329917025676: 1, 28.636644413526977: 1, 28.426658216244807: 1, 28.3981
33136020846: 1, 28.382668158295473: 1, 28.139660710289547: 1, 28.129258
171406036: 1, 28.119837907718118: 1, 27.77097386709698: 1, 27.520821571
953064: 1, 27.49753235494914: 1, 27.40309301449611: 1, 27.1071526224887
34: 1, 26.92722189824348: 1, 26.695492321483016: 1, 26.554869006993886:
1, 26.336714751000187: 1, 26.27207725640616: 1, 26.207797455471738: 1,
26.1305532564752: 1, 26.06047124472428: 1, 26.03302759978919: 1, 25.930
```

```
545489587185: 1, 25.768459906364303: 1, 25.433514244361557: 1, 25.31884
412775549: 1, 25.17161924249155: 1, 25.10585630386131: 1, 25.095420362
358333: 1, 25.042045594862707: 1, 25.02644012484162: 1, 24.889446818018
88: 1, 24.788856336568625: 1, 24.76256089346469: 1, 24.75472954562874:
1, 24.506806229264818: 1, 24.483317634112694: 1, 24.468416928440398: 1,
24.461208592691147: 1, 24.45260188234402: 1, 24.423725882680852: 1, 24.
36533011186098: 1, 24.35104009972439: 1, 24.184803790002668: 1, 24.1248
39607667802: 1, 24.107846619071303: 1, 23.837942645498476: 1, 23.818008
41520302: 1, 23.813623552337834: 1, 23.765137613245933: 1, 23.674367297
492594: 1, 23.583984172362392: 1, 23.516588123546775: 1, 23.35957006492
48: 1, 23.260930051246785: 1, 23.249534155561225: 1, 23.17409987885061
5: 1, 22.973942947389997: 1, 22.942162725540562: 1, 22.91227944004784:
1, 22.90966957099252: 1, 22.893834113121155: 1, 22.658350492108283: 1,
22.59019682004605: 1, 22.55586048711108: 1, 22.442428460912183: 1, 22.4
16293842091058: 1, 22.326065088055646: 1, 22.324597432690382: 1, 22.312
26835770719: 1, 22.26120446461926: 1, 22.2529019129376: 1, 22.223043642
539825: 1, 22.199322030960623: 1, 22.101922621515747: 1, 22.09007868125
4184: 1, 22.07693178015118: 1, 22.074758943615407: 1, 21.95936926275198
8: 1, 21.93394695866355: 1, 21.90465614627163: 1, 21.86682811587659: 1,
21.861691041716547: 1, 21.70711398482642: 1, 21.642475886239236: 1, 2
1.59617101978728: 1, 21.556588929453916: 1, 21.535687916492716: 1, 21.5
16867918192712: 1, 21.485368658905212: 1, 21.365274248393433: 1, 21.347
313037926433: 1, 21.293698945478866: 1, 21.289245004997: 1, 21.22410356
7860116: 1, 21.13093090297478: 1, 21.052217451318725: 1, 21.04321032044
988: 1, 21.02384666709195: 1, 20.997242033486838: 1, 20.98151306720074
8: 1, 20.96836881634294: 1, 20.922807447754906: 1, 20.914085864156405:
1, 20.758189619319616: 1, 20.752314238987047: 1, 20.734222634103592: 1,
20.66823365235398: 1, 20.65973729304347: 1, 20.556513813331254: 1, 20.5
37272539084196: 1, 20.35120812554191: 1, 20.295404009774362: 1, 20.2323
50817341892: 1, 20.231932776107104: 1, 20.205564021888023: 1, 20.170223
252067476: 1, 20.162795547163867: 1, 20.035412759103362: 1, 20.01584133
8979087: 1, 19.97530533179047: 1, 19.848009832535375: 1, 19.82303022501
1998: 1, 19.728539069229047: 1, 19.724218163972907: 1, 19.6479824193165
56: 1, 19.60699036167022: 1, 19.379659202418885: 1, 19.353894732832664:
1, 19.33401330691549: 1, 19.324022193877287: 1, 19.290846425093356: 1,
19.227978077037083: 1, 19.194238309340342: 1, 19.15479270350682: 1, 19.
088892658773418: 1, 19.068940203924154: 1, 19.06649787041529: 1, 19.063
207867217802: 1, 19.04233910682987: 1, 18.976763719271325: 1, 18.911610
500107056: 1, 18.906663726170308: 1, 18.891108608024116: 1, 18.87054528
```

```
694546: 1, 18.841448797619044: 1, 18.839636940801064: 1, 18.81390108873
743: 1, 18.813766554131494: 1, 18.802768724676213: 1, 18.7879696756333
8: 1, 18.770292584590674: 1, 18.72240952774448: 1, 18.7222310735877:
1, 18.699422758123596: 1, 18.69062039473171: 1, 18.651679712735717: 1,
18.63888290218619: 1, 18.608476861626986: 1, 18.596859295664338: 1, 18.
595670422676704: 1, 18.49698264264268: 1, 18.452240332113185: 1, 18.405
980051107562: 1, 18.39696644883793: 1, 18.332766790438278: 1, 18.239705
184539467: 1, 18.146066959156563: 1, 18.10941381024801: 1, 18.109126701
46675: 1, 18.086371287989053: 1, 18.06299545046891: 1, 18.0073894360004
84: 1, 17.982194907919496: 1, 17.916937542667732: 1, 17.86813236290357
3: 1, 17.8611614014068: 1, 17.843290349568846: 1, 17.842841033505557:
1, 17.786561622855878: 1, 17.770406909781812: 1, 17.75560202803379: 1,
17.71272501707648: 1, 17.70317115712559: 1, 17.694950667299487: 1, 17.6
7450217153932: 1, 17.640993085573516: 1, 17.57940312026416: 1, 17.5752
19931854743: 1, 17.550456219403323: 1, 17.54267477260149: 1, 17.5358686
16005214: 1, 17.509269923742018: 1, 17.507468421340256: 1, 17.453862730
313272: 1, 17.387926686139696: 1, 17.38355464577684: 1, 17.362378333254
34: 1, 17.32463102200246: 1, 17.277106079542566: 1, 17.274572357359062:
1, 17.26710147704727: 1, 17.253753366037675: 1, 17.243631555211234: 1,
17.183696130097534: 1, 17.181003445800204: 1, 17.18052108679944: 1, 17.
172154816073977: 1, 17.143094086684815: 1, 17.134738793339828: 1, 17.10
7187520319947: 1, 17.066051200241326: 1, 17.05124732927413: 1, 16.98198
042397616: 1, 16.91139511350747: 1, 16.907594797067286: 1, 16.901067871
06189: 1, 16.888065070093997: 1, 16.885729132567214: 1, 16.879805040333
99: 1, 16.870329850061264: 1, 16.827533917597826: 1, 16.81846143980814:
1, 16.74733425853835: 1, 16.701109147379817: 1, 16.694457359560435: 1,
16.682644710418288: 1, 16.673142300515806: 1, 16.641908376864496: 1, 1
6.577860834773794: 1, 16.515691659117476: 1, 16.420480802498236: 1, 16.
36082654811647: 1, 16.359144130002612: 1, 16.34942139963106: 1, 16.3045
1398964353: 1, 16.29828627332934: 1, 16.26264707103789: 1, 16.227391580
38653: 1, 16.198531364512665: 1, 16.167865450705648: 1, 16.165568378488
047: 1, 16.16506590034605: 1, 16.096605472536737: 1, 16.09572540370659
5: 1, 16.09204922181253: 1, 16.05991682738286: 1, 16.01066447359151: 1,
15.98975940335687: 1, 15.98075207534722: 1, 15.979895978704844: 1, 15.9
0135442731551: 1, 15.899160615360376: 1, 15.888878640483782: 1, 15.8584
18127817034: 1, 15.80400430058694: 1, 15.798131199164068: 1, 15.7797471
39069359: 1, 15.66114118225803: 1, 15.64996484228942: 1, 15.64466159416
3184: 1, 15.616464656414617: 1, 15.585080098969009: 1, 15.5807022266459
49: 1, 15.534564643886089: 1, 15.469760490102434: 1, 15.46974382640175
```

```
9: 1, 15.405883916195029: 1, 15.348500202810492: 1, 15.34589188831375:
1, 15.324108990349648: 1, 15.31914554801107: 1, 15.307196351569033: 1,
15.302158912706542: 1, 15.278009109300912: 1, 15.272520918654196: 1, 1
5.250533572913877: 1, 15.244448875244238: 1, 15.227206883700102: 1, 15.
174491204630067: 1, 15.15434459276512: 1, 15.12158572812026: 1, 15.1145
50198092523: 1, 15.095940379929218: 1, 15.070147019554197: 1, 15.056945
821117854: 1, 15.012468414689256: 1, 15.012106796997005: 1, 14.99922078
3544734: 1, 14.929539916065844: 1, 14.926902266670233: 1, 14.8917732966
55089: 1, 14.8881508388509: 1, 14.871492567202308: 1, 14.84113513483416
6: 1, 14.836869804107943: 1, 14.826819692971501: 1, 14.816961430125202:
1, 14.815054579996652: 1, 14.798141824663487: 1, 14.786480393966322: 1,
14.781778407192252: 1, 14.771254288397955: 1, 14.760308392142568: 1, 1
4.728484700445884: 1, 14.705274007318685: 1, 14.666793541794041: 1, 14.
664065734144794: 1, 14.661466446889952: 1, 14.638720214863598: 1, 14.63
864728378549: 1, 14.598562719125662: 1, 14.567596872850705: 1, 14.55489
8269008826: 1, 14.535557249011553: 1, 14.487582467584115: 1, 14.4746788
59874993: 1, 14.46035034776475: 1, 14.442341698291902: 1, 14.4398358715
67233: 1, 14.43971255845006: 1, 14.434460769440747: 1, 14.4293275648377
7: 1, 14.423696777934413: 1, 14.360348627973519: 1, 14.34568563460529:
1, 14.338296373488316: 1, 14.306842497227102: 1, 14.298962473853667: 1,
14.283497109365277: 1, 14.275660746197861: 1, 14.258593927547057: 1, 1
4.245063224150961: 1, 14.156073508353664: 1, 14.154823658824116: 1, 14.
102331685761515: 1, 14.08148234693988: 1, 14.079539671401074: 1, 13.965
849674853526: 1, 13.96080922245713: 1, 13.936659193733517: 1, 13.919867
16130453: 1, 13.913495099947797: 1, 13.913183129491038: 1, 13.901520674
145763: 1, 13.885010002711889: 1, 13.85852256799749: 1, 13.844041766329
271: 1, 13.80659675565497: 1, 13.782620875836566: 1, 13.78086719380269
5: 1, 13.76147127370592: 1, 13.74552740523466: 1, 13.646919937817634:
1, 13.642208784592352: 1, 13.630710438463138: 1, 13.621268567084154: 1,
13.597582787562736: 1, 13.58712390645743: 1, 13.55152114950087: 1, 13.4
8775664185955: 1, 13.463618236146052: 1, 13.456761753499437: 1, 13.4538
56909324115: 1, 13.374055885052837: 1, 13.318745882456756: 1, 13.317786
883511124: 1, 13.25611351549946: 1, 13.255509640817: 1, 13.248906420282
022: 1, 13.244659542026762: 1, 13.23776199293127: 1, 13.22037820253770
2: 1, 13.179184747044019: 1, 13.156397794281334: 1, 13.11335572853403:
1, 13.086492862853826: 1, 13.085150223106009: 1, 13.065603108322685: 1,
13.05908765894016: 1, 13.043651848209183: 1, 13.034793609583584: 1, 13.
02011767011801: 1, 13.019884440461844: 1, 13.017742623424105: 1, 12.984
11111392521: 1, 12.958721661331497: 1, 12.954166017499851: 1, 12.925581
```

```
426875766: 1, 12.908277089697659: 1, 12.902963029431898: 1, 12.87821500
4585243: 1, 12.85903928306183: 1, 12.832802652418872: 1, 12.79315375286
8749: 1, 12.789147788457177: 1, 12.779971627831115: 1, 12.7100204690606
7: 1, 12.705891156048686: 1, 12.69038015899689: 1, 12.687826900682388:
1, 12.652303162353347: 1, 12.617595099124468: 1, 12.573600754171794: 1,
12.545461715688537: 1, 12.499761796271494: 1, 12.476436474449944: 1, 1
2.449455010704856: 1, 12.425310721451915: 1, 12.412748823388412: 1, 12.
401198677521277: 1, 12.368611153704821: 1, 12.35766143547632: 1, 12.354
797202212069: 1, 12.347411110434978: 1, 12.33464107596166: 1, 12.330169
895100138: 1, 12.306784426672152: 1, 12.294772705201265: 1, 12.29162643
087986: 1, 12.288836748414145: 1, 12.27779144531277: 1, 12.276321654110
749: 1, 12.268566386882926: 1, 12.259986405288975: 1, 12.25697229298104
3: 1, 12.251713439243478: 1, 12.25056712237543: 1, 12.245422355934977:
1, 12.238066770489976: 1, 12.193424057243552: 1, 12.185365046015159: 1,
12.18242686868844: 1, 12.168559568836043: 1, 12.122988079380645: 1, 12.
09601690401812: 1, 12.075854424668206: 1, 12.073131234775976: 1, 12.029
799761686004: 1, 12.020834791722153: 1, 12.009136662589349: 1, 12.00727
9657503675: 1, 12.007265893002797: 1, 11.976386111633687: 1, 11.9637459
8881555: 1, 11.947858395147854: 1, 11.936591155061324: 1, 11.9320539916
1506: 1, 11.925364853095827: 1, 11.891149954100072: 1, 11.8758307934767
9: 1, 11.84074064476196: 1, 11.79964136888288: 1, 11.79730912462491: 1,
11.796422031297098: 1, 11.791793931940811: 1, 11.78403376915169: 1, 11.
781421877075518: 1, 11.752746383406834: 1, 11.748311947886098: 1, 11.73
3446868612537: 1, 11.726260486092391: 1, 11.687089303308243: 1, 11.6704
53376639005: 1, 11.661857722258295: 1, 11.646113534886016: 1, 11.606190
180060505: 1, 11.596762502716217: 1, 11.58294535045739: 1, 11.565605923
962798: 1, 11.565481984022087: 1, 11.558874069203705: 1, 11.55256028116
2522: 1, 11.548070216547888: 1, 11.52855894578669: 1, 11.4961486260199
2: 1, 11.485802534617338: 1, 11.464446426950987: 1, 11.435635251486373:
1, 11.378275006515995: 1, 11.37018899046207: 1, 11.36572169178425: 1, 1
1.345777090764473: 1, 11.322569242506736: 1, 11.27783814461811: 1, 11.2
77063116932464: 1, 11.275655881847946: 1, 11.2629786550939: 1, 11.24787
1718238548: 1, 11.247563370619053: 1, 11.236902772046035: 1, 11.2288494
57771698: 1, 11.215295449840776: 1, 11.203314058415943: 1, 11.186150022
967198: 1, 11.156616747334446: 1, 11.15179879077316: 1, 11.116072555873
822: 1, 11.110346953957496: 1, 11.085922558761867: 1, 11.05100835214500
6: 1, 11.049170489579133: 1, 11.038822006426763: 1, 11.037585989769966:
1, 11.029882687491959: 1, 11.025177765373359: 1, 11.024902845534605: 1,
11.019800680751768: 1, 10.997483599841578: 1, 10.990774613219898: 1, 1
```

```
0.984637867609397: 1, 10.93375273717659: 1, 10.931314179718637: 1, 10.9
16355590854815: 1, 10.906401336306036: 1, 10.863418202132774: 1, 10.856
835949994668: 1, 10.854863271618305: 1, 10.833525805342877: 1, 10.82953
0266400225: 1, 10.81763851693931: 1, 10.785298373504334: 1, 10.77516502
4323996: 1, 10.774684799681134: 1, 10.765942290564144: 1, 10.7532266834
18045: 1, 10.751530486913184: 1, 10.746428456917439: 1, 10.745834501539
5: 1, 10.71625885219281: 1, 10.695616937762907: 1, 10.693275444031492:
1, 10.687157993825334: 1, 10.677668232712893: 1, 10.67519908817194: 1,
10.665320188981253: 1, 10.662066375142155: 1, 10.65394581201275: 1, 10.
644044806394461: 1, 10.639946488455612: 1, 10.626486648376556: 1, 10.61
2006779963552: 1, 10.610167872735426: 1, 10.597632488926344: 1, 10.5865
64166441455: 1, 10.582981705565606: 1, 10.582495469411858: 1, 10.580324
844081352: 1, 10.56924360002415: 1, 10.568759228788894: 1, 10.565460978
677143: 1, 10.551221456359595: 1, 10.547763484532988: 1, 10.52968178376
6572: 1, 10.514320215744153: 1, 10.507488281854002: 1, 10.4977425490396
96: 1, 10.49221787829345: 1, 10.484390453907986: 1, 10.465914093314117:
1, 10.460428707068687: 1, 10.444836407744932: 1, 10.422493469470107: 1,
10.417160025033315: 1, 10.412578472005084: 1, 10.399522332606278: 1, 1
0.391598252085108: 1, 10.346957630426731: 1, 10.344100296748378: 1, 10.
343261777419764: 1, 10.331180875475978: 1, 10.328407897708633: 1, 10.29
976293332519: 1, 10.288704927897113: 1, 10.27641391060141: 1, 10.236154
599796993: 1, 10.224464479980282: 1, 10.223807926147094: 1, 10.21615649
852254: 1, 10.173099608571185: 1, 10.158227784763818: 1, 10.14468429100
6818: 1, 10.141337440999754: 1, 10.105201485635904: 1, 10.0627592864127
37: 1, 10.062151070796164: 1, 10.038026469745342: 1, 10.03308955410513
7: 1, 10.025995948169266: 1, 10.014546177767746: 1, 10.006760750610841:
1, 9.992629117247303: 1, 9.987250595418299: 1, 9.985400741223973: 1, 9.
983220423053686: 1, 9.979230059391059: 1, 9.974089244267985: 1, 9.96638
4131433246: 1, 9.965816137285126: 1, 9.957070317491535: 1, 9.9509476209
381: 1, 9.943382878551587: 1, 9.92688058703856: 1, 9.911743590126523:
1, 9.896588418744681: 1, 9.892274478770204: 1, 9.886360873107806: 1, 9.
876718634674292: 1, 9.872168937753722: 1, 9.855493401529309: 1, 9.84682
442022308: 1, 9.827115796336594: 1, 9.81712198755362: 1, 9.808794168966
57: 1, 9.807773882906753: 1, 9.785597446307527: 1, 9.776482496998705:
1, 9.763053803719368: 1, 9.74110005853616: 1, 9.721003524510436: 1, 9.7
11879781882182: 1, 9.708197275969264: 1, 9.69553456462026: 1, 9.6952680
910457: 1, 9.680647689772798: 1, 9.67680387926528: 1, 9.65534820581281
6: 1, 9.647058176294287: 1, 9.627613761936855: 1, 9.62707858445834: 1,
9.578815993790474: 1, 9.572173969622066: 1, 9.571490285445865: 1, 9.569
```

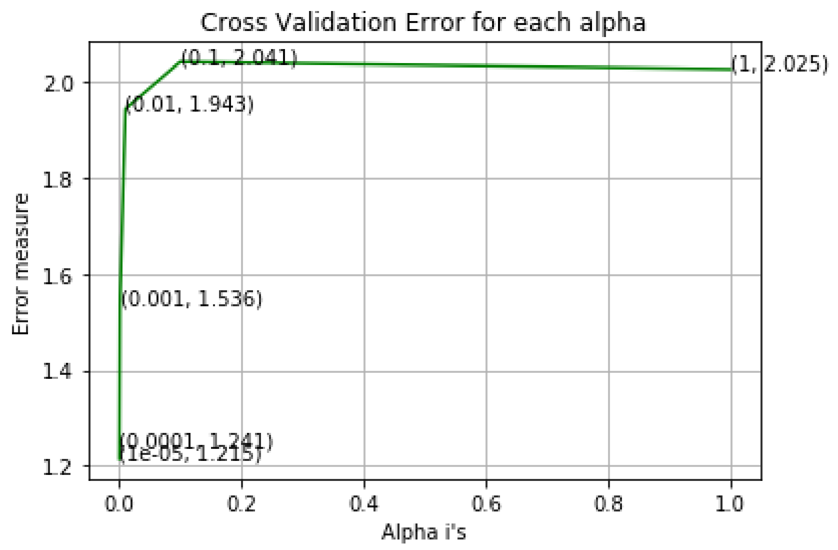
```
88358728841: 1, 9.56922830490767: 1, 9.562399165827546: 1, 9.5592396884
38383: 1, 9.553936587726197: 1, 9.533322535941561: 1, 9.52698035573507
2: 1, 9.51537158944235: 1, 9.508383780125298: 1, 9.493840076720963: 1,
9.49191104732251: 1, 9.487835667540178: 1, 9.485520745301658: 1, 9.4831
89994527514: 1, 9.482092484474618: 1, 9.479771319851888: 1, 9.474552498
365597: 1, 9.468115225160508: 1, 9.45354760138001: 1, 9.43733597830717:
1, 9.429957532408888: 1, 9.424061710454362: 1, 9.42276959388311: 1, 9.4
19930649935226: 1, 9.417679851456327: 1, 9.379917278076878: 1, 9.374862
441791525: 1, 9.371448764798803: 1, 9.35105185477756: 1, 9.337613607018
671: 1, 9.331367761259818: 1, 9.330516362307385: 1, 9.324774867539121:
1, 9.30500521875323: 1, 9.280542793363212: 1, 9.212745091471508: 1, 9.1
98962913627202: 1, 9.18480319477128: 1, 9.180514932637461: 1, 9.1726794
73023713: 1, 9.170911007609938: 1, 9.160557519923085: 1, 9.160381296638
672: 1, 9.143447355883573: 1, 9.137650606526154: 1, 9.11854373983682:
1, 9.113859546198434: 1, 9.105311376629079: 1, 9.105212629535867: 1, 9.
098615143550784: 1, 9.095494860181892: 1, 9.086424845540412: 1, 9.08330
4206591539: 1, 9.082365079719365: 1, 9.081834295517396: 1, 9.0719110573
4661: 1, 9.071350206755076: 1, 9.059515780048391: 1, 9.043012463147617:
1, 9.042538851928011: 1, 9.037593358375112: 1, 9.03403748754432: 1, 9.0
30988874316506: 1, 9.027437589605164: 1, 9.01351338843263: 1, 8.9934627
53526495: 1, 8.980852701876238: 1, 8.962160868563041: 1, 8.956003966526
044: 1, 8.955091294929813: 1, 8.945470197842988: 1, 8.915286075008881:
1, 8.91202837172928: 1, 8.886814104923253: 1, 8.878918595992378: 1, 8.8
78878702605853: 1, 8.866346816665112: 1, 8.852824874082156: 1, 8.849277
565571976: 1, 8.836680600609997: 1, 8.828464759863108: 1, 8.82460445801
6248: 1, 8.818639848645148: 1, 8.816512638554475: 1, 8.808748210812913:
1, 8.80785960161107: 1, 8.794049396831024: 1, 8.76860307056277: 1, 8.73
570736347887: 1, 8.72348226114123: 1, 8.708546172153945: 1, 8.689149115
053159: 1, 8.679135816874336: 1, 8.67634887759013: 1, 8.67613124275223
8: 1, 8.672169837527742: 1, 8.671451653746379: 1, 8.620913608768818: 1,
8.608552441198215: 1, 8.602482301956679: 1, 8.587292459565445: 1, 8.576
290948172788: 1, 8.56635851223616: 1, 8.547863676303264: 1, 8.523134439
351656: 1, 8.520586772921067: 1, 8.517438599005827: 1, 8.51312497699232
7: 1, 8.512234422484115: 1, 8.501628110247058: 1, 8.497763937154: 1, 8.
49627593936061: 1, 8.493924596498571: 1, 8.476849498946217: 1, 8.472821
462246081: 1, 8.47132144145961: 1, 8.466321192275673: 1, 8.453715142215
56: 1, 8.44632507601793: 1, 8.437571788031322: 1, 8.42289651263085: 1,
8.4144720787316: 1, 8.395685427409491: 1, 8.349131448720254: 1, 8.34478
393772347: 1, 8.34403470285388: 1, 8.328369059107223: 1, 8.313521810101
```

```
71: 1, 8.288930114764094: 1, 8.260335882521606: 1, 8.25596301415283: 1,
8.253766520416034: 1, 8.25284280863542: 1, 8.23696736448709: 1, 8.23329
0249698241: 1, 8.218005607676174: 1, 8.214009610250864: 1, 8.1895665405
86762: 1, 8.162750233253597: 1, 8.16009608013911: 1, 8.154529170713907:
1, 8.141128923045862: 1, 8.124125033657771: 1, 8.122468427165828: 1, 8.
115508654733024: 1, 8.097782953843357: 1, 8.09312433248103: 1, 8.088341
762555451: 1, 8.084061523119626: 1, 8.05086905570654: 1, 8.049045535815
578: 1, 8.048478914124148: 1, 8.010450740752725: 1, 8.003812507248174:
1, 7.991711935328912: 1, 7.9897647874115725: 1, 7.962723765832628: 1,
7.961494990415918: 1, 7.959583503819116: 1, 7.937001918654346: 1, 7.930
13238367277: 1, 7.928438805357929: 1, 7.887201688708882: 1, 7.884567253
656023: 1, 7.882772394096681: 1, 7.859535764894392: 1, 7.85478842473602
5: 1, 7.853047954117967: 1, 7.834535986532519: 1, 7.797506057664205: 1,
7.790919688842753: 1, 7.75429563405663: 1, 7.734942069087361: 1, 7.7266
46477230326: 1, 7.726197308311359: 1, 7.705828398190272: 1, 7.693969920
070159: 1, 7.693216598868995: 1, 7.691964350424489: 1, 7.69133607174845
5: 1, 7.687511745307956: 1, 7.686987471580909: 1, 7.671957782061237: 1,
7.6701157596705585: 1, 7.667625458833367: 1, 7.658329513549792: 1, 7.64
1058995251268: 1, 7.639054364257074: 1, 7.637872067350544: 1, 7.6357326
33099721: 1, 7.633425370881991: 1, 7.574117968224907: 1, 7.573349387387
3865: 1, 7.568066817903649: 1, 7.551587104519816: 1, 7.542245739408234
5: 1, 7.534646374128706: 1, 7.530359356281681: 1, 7.520680563764968: 1,
7.497368490887803: 1, 7.496618313136909: 1, 7.470988894141228: 1, 7.451
487743825685: 1, 7.44081186412795: 1, 7.43498946665917: 1, 7.3645593914
47456: 1, 7.3080384276336545: 1, 7.293232452902024: 1, 7.27522562533674
9: 1, 7.274741800400445: 1, 7.261343727610959: 1, 7.234568191178083: 1,
7.233814686638065: 1, 7.21897166762031: 1, 7.154852668263432: 1, 7.1393
96047986362: 1, 7.106601825521047: 1, 7.084806483257146: 1, 7.075946978
526483: 1, 7.064339112516131: 1, 7.062294921950814: 1, 7.03236964095698
9: 1, 7.015784867457402: 1, 7.010486990360133: 1, 6.972129691755944: 1,
6.9649642133327525: 1, 6.912897216860773: 1, 6.895791859656203: 1, 6.88
856019694868: 1, 6.88291146548277: 1, 6.877383076955847: 1, 6.870933377
130807: 1, 6.859099878398563: 1, 6.792640193820731: 1, 6.77818469246655
7: 1, 6.723495287011001: 1, 6.692546187522791: 1, 6.65327737342695: 1,
6.632695213268368: 1, 6.260706975818979: 1})
```

In [56]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
```

```
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[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 vali
dation log loss is:",log_loss(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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2148101634148702
For values of alpha =  0.0001 The log loss is: 1.2407204504334766
For values of alpha =  0.001 The log loss is: 1.5356784047973855
For values of alpha =  0.01 The log loss is: 1.9429105492744883
For values of alpha =  0.1 The log loss is: 2.040761108212507
For values of alpha =  1 The log loss is: 2.0251740163882554
```



```
For values of best alpha =  1e-05 The train log loss is: 0.772792634132
9535
For values of best alpha =  1e-05 The cross validation log loss is: 1.2
148101634148702
For values of best alpha =  1e-05 The test log loss is: 1.1088266523773
178
```

In [57]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer()
    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_coun
ts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [58]:
```
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 appe
ared in train data")
```

```
1.351 % of word of test data appeared in train data
1.45 % of word of Cross Validation appeared in train data
```

In [59]:
```
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y,
clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

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

In [60]:
```
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 [61]:
```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])
    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())
    word_present = 0
    for i,v in enumerate(indices):
        if(v<fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no=True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point
 [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data p
oint [{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point
 [{}]".format(word,yes_no))

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

In [138]:
```
def get_impfeature_names_tfidf(indices, text, gene, var, no_features):
    gene_tfidf_vec = TfidfVectorizer(max_features = 1000)
    var_tfidf_vec = TfidfVectorizer(max_features = 1000)
    text_tfidf_vec = TfidfVectorizer(min_df=3)

    gene_vec = gene_tfidf_vec.fit(train_df['Gene'])
    var_vec  = var_tfidf_vec.fit(train_df['Variation'])
    text_vec = text_tfidf_vec.fit(train_df['TEXT'])
    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_tfidf_vec.get_feature_names())
    word_present = 0
    for i,v in enumerate(indices):
        if(v<fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no=True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point
 [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data p
oint [{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point
 [{}]".format(word,yes_no))

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

In [62]:
```
from scipy.sparse import hstack

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

```
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_fea
ture_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,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, trai
n_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_t
ext_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [63]:
```
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 = ", t
est_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, 22
27)
(number of data points * number of features) in test data =  (665, 222
7)
(number of data points * number of features) in cross validation data =
(532, 2227)
```

In [64]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_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, 2
7)
(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)
```

4.1. Base Line Model

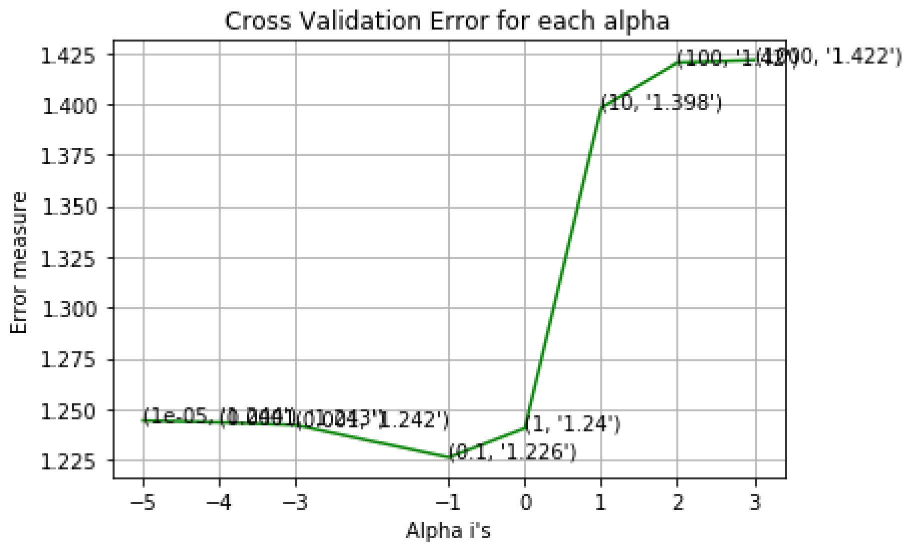
4.1.1. Naive Bayes

In [65]:
```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
```

```
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

```
for alpha = 1e-05
Log Loss : 1.2442766226624395
for alpha = 0.0001
Log Loss : 1.2433632257353313
for alpha = 0.001
Log Loss : 1.2420855832330644
```

```
for alpha = 0.1
Log Loss : 1.2261369626270107
for alpha = 1
Log Loss : 1.2404899532565405
for alpha = 10
Log Loss : 1.398186441499341
for alpha = 100
Log Loss : 1.420394827639618
for alpha = 1000
Log Loss : 1.4215970992551819
```

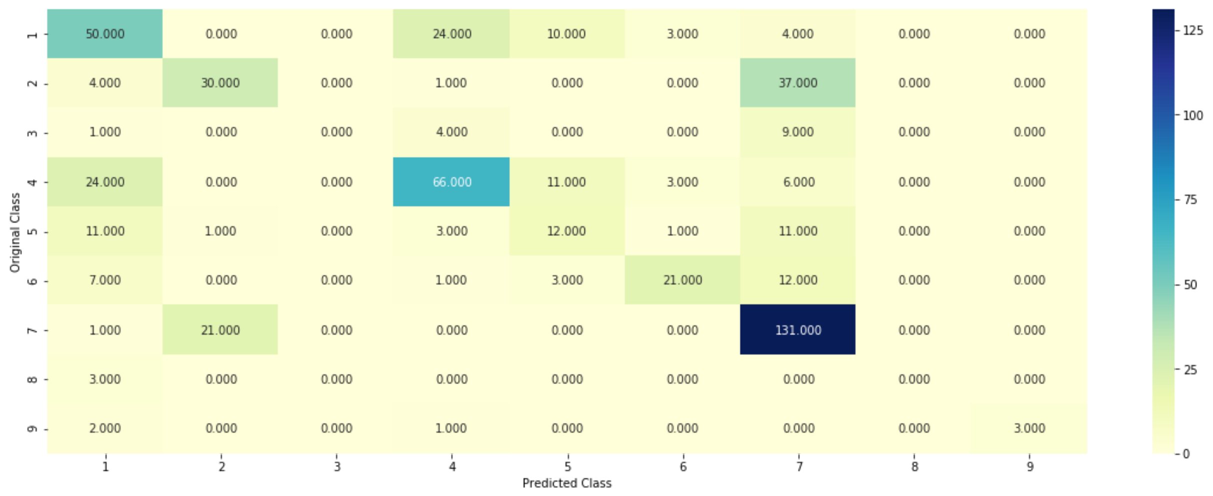


```
For values of best alpha =  0.1 The train log loss is: 0.79848532305220
04
For values of best alpha =  0.1 The cross validation log loss is: 1.226
1369626270107
For values of best alpha =  0.1 The test log loss is: 1.199899624066156
8
```

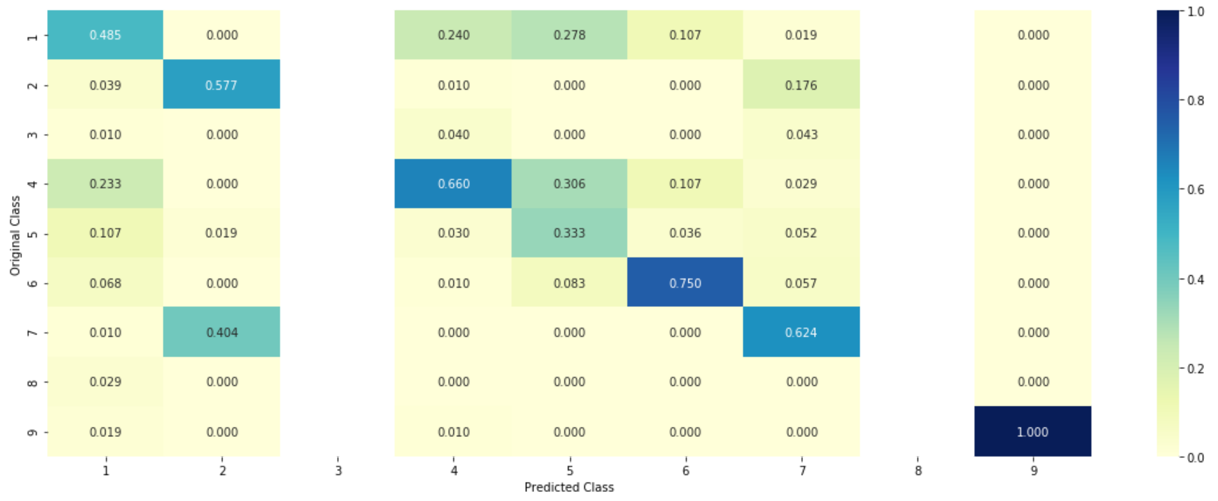
In [66]:
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

```
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-pro
bability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))
```

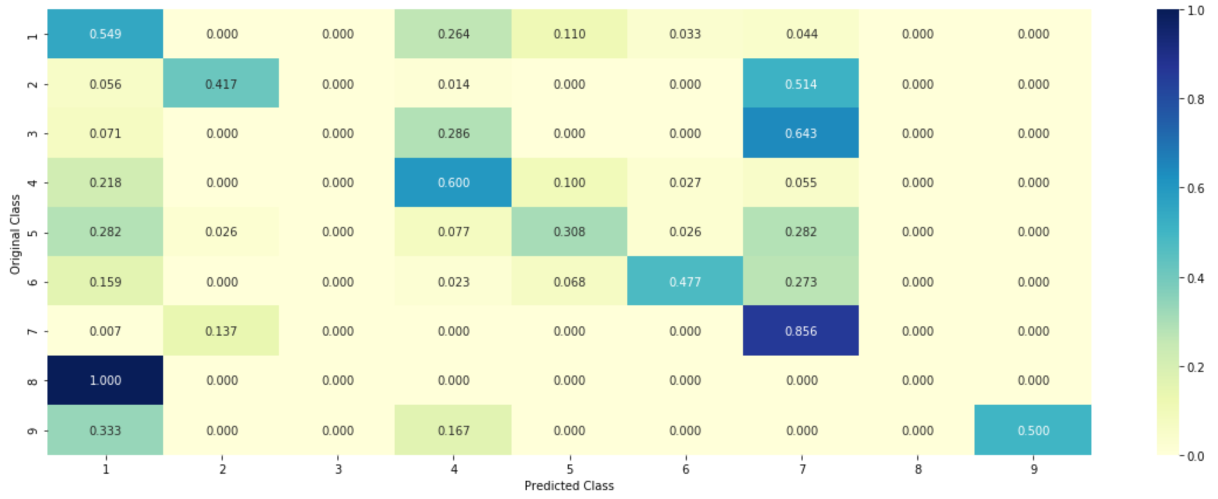
```
Log Loss : 1.2261369626270107
Number of missclassified point : 0.41165413533383459
-------------------- Confusion matrix --------------------
```



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



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



In [70]:
```
test_point_index=3
no_feature =500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0485 0.0451 0.0168 0.0465 0.0335 0.7
556 0.0461 0.0043 0.0037]]
Actual Class : 6
--------------------------------------------------
128 Text feature [0001] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

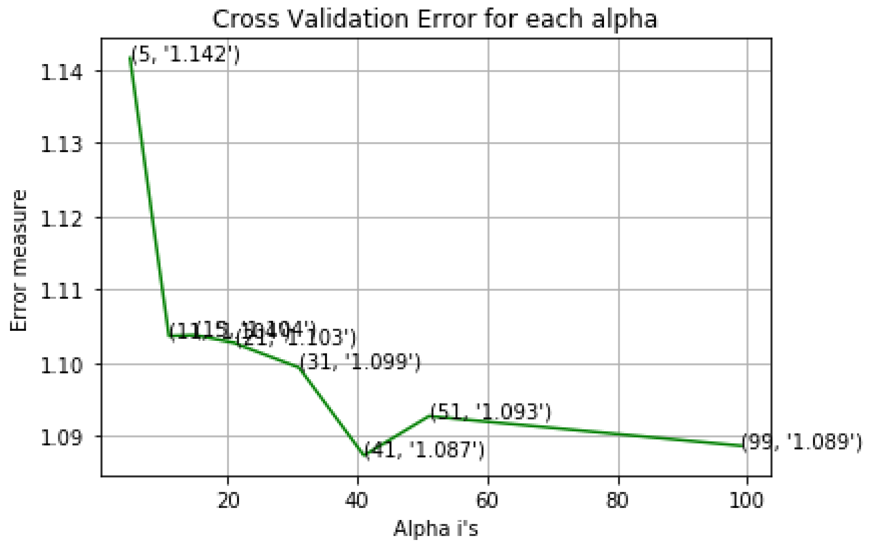
In [71]:

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

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

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


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

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

```
for alpha = 5
Log Loss : 1.1415779392265757
for alpha = 11
Log Loss : 1.1036520093495181
for alpha = 15
Log Loss : 1.1037809410733435
for alpha = 21
Log Loss : 1.1027570696679192
for alpha = 31
Log Loss : 1.0994111613505804
for alpha = 41
Log Loss : 1.0873904842294886
```

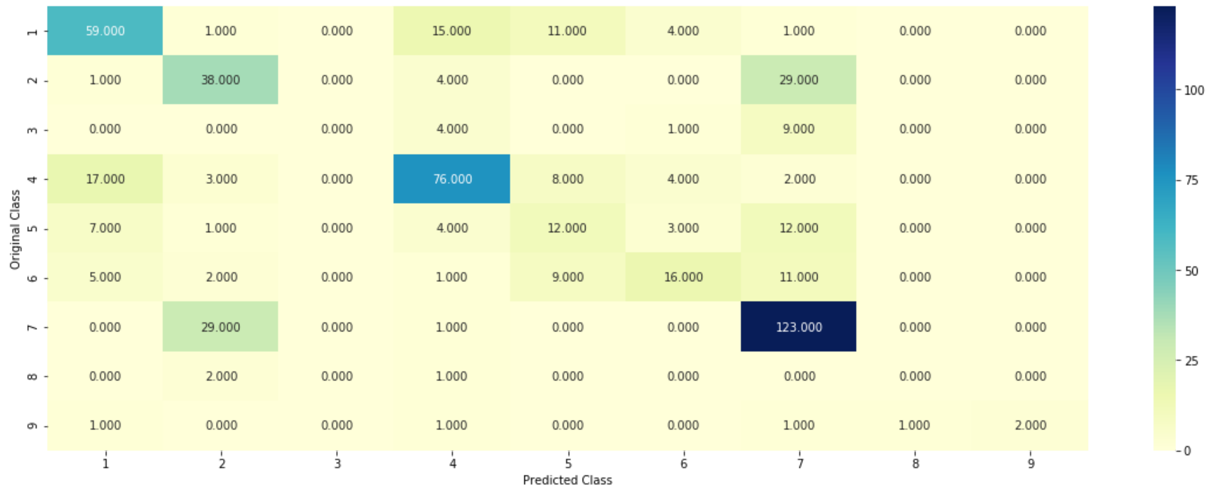
```
for alpha = 51
Log Loss : 1.0927459764087593
for alpha = 99
Log Loss : 1.088691262044198
```



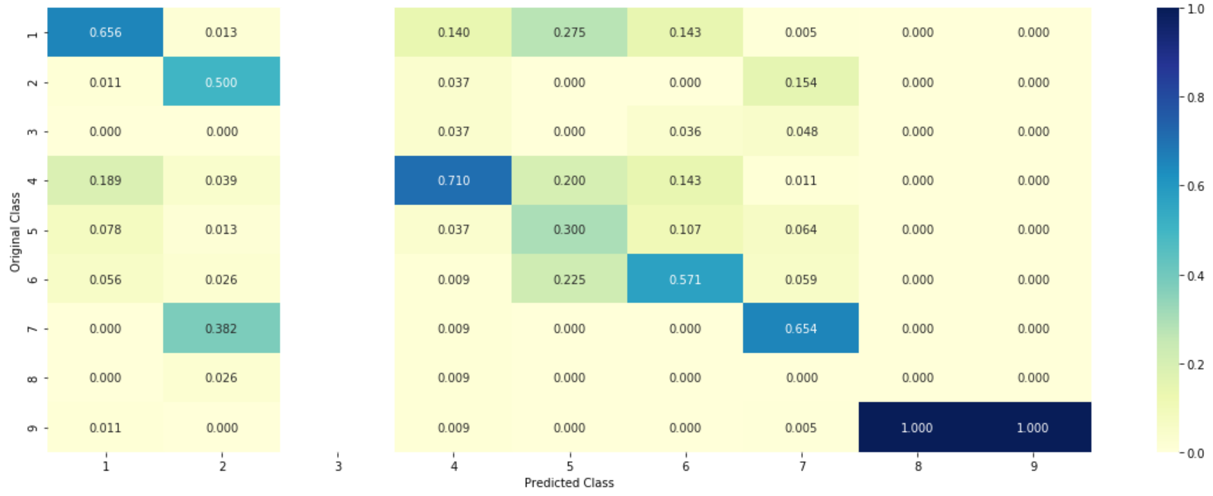
```
For values of best alpha =  41 The train log loss is: 0.823394408360955
9
For values of best alpha =  41 The cross validation log loss is: 1.0873
904842294886
For values of best alpha =  41 The test log loss is: 1.1381835319005038
```

In [72]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x
_responseCoding, cv_y, clf)
```

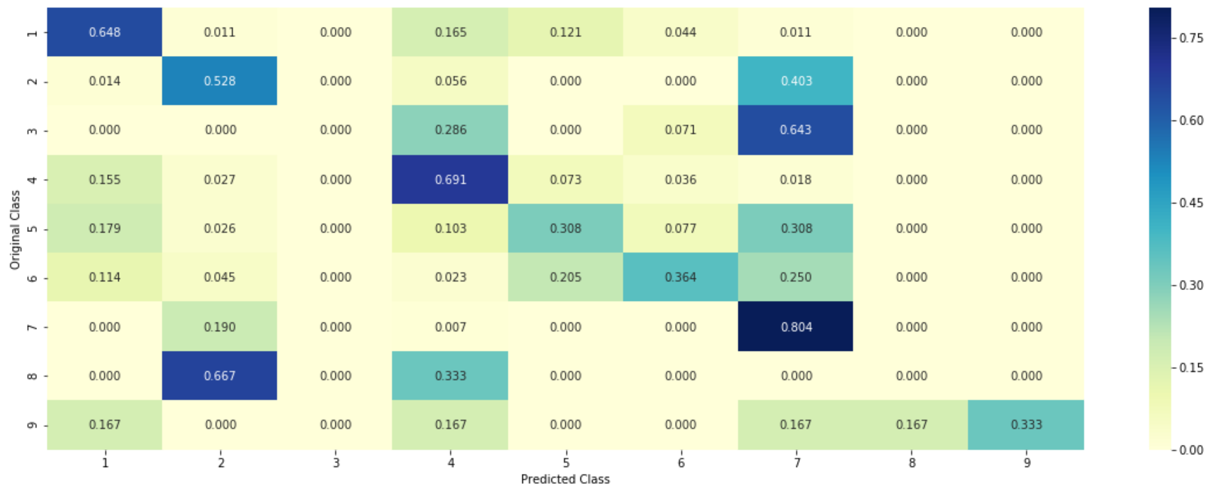
```
Log loss : 1.0873904842294886
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```



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



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



In [88]:
```
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 = 150
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].resh
ape(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 : 9
Actual Class : 4
The  41  nearest neighbours of the test points belongs to classes [4 4
4 4 1 1 4 1 1 4 1 1 1 1 4 4 4 4 1 1 1 1 1 4 4 4 4 4 4 1 6 1 4 4 4 4
 4 1 1 1]
Fequency of nearest points : Counter({4: 21, 1: 19, 6: 1})
```

In [89]:
```
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 = 129

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].resh
ape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

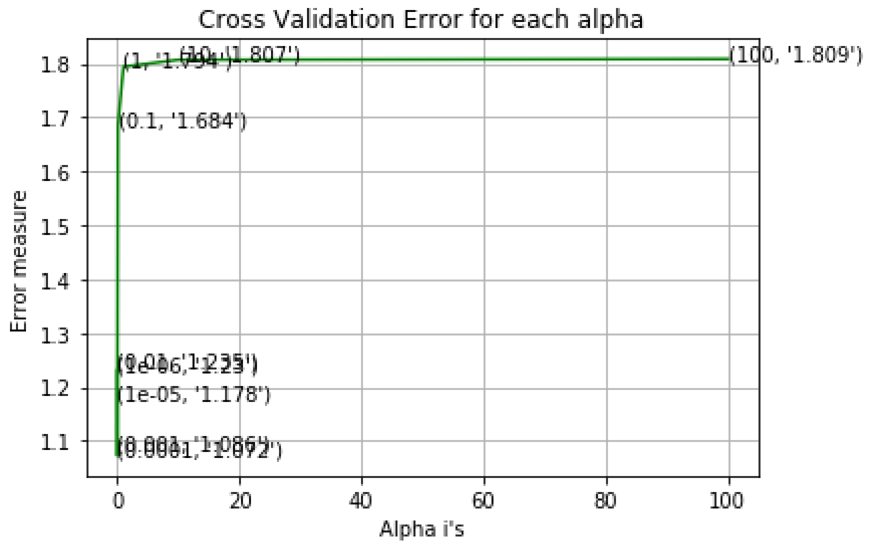
```
Predicted Class : 7
Actual Class : 7
the k value for knn is 41 and the nearest neighbours of the test points
belongs to classes [7 3 3 3 3 3 3 3 7 7 7 7 7 7 7 7 7 7 2 2 7 7 2 7 7 7
3 7 7 2 3 7 7 7 2 5 7
 7 7 7 7]
Fequency of nearest points : Counter({7: 26, 3: 9, 2: 5, 5: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

In [145]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='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 vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
for alpha = 1e-06
Log Loss : 1.2303642842012958
for alpha = 1e-05
Log Loss : 1.1777387536210098
for alpha = 0.0001
Log Loss : 1.071783239542718
for alpha = 0.001
Log Loss : 1.085704362544589
for alpha = 0.01
Log Loss : 1.235085491714876
for alpha = 0.1
Log Loss : 1.6839924148775527
for alpha = 1
Log Loss : 1.794468716097687
for alpha = 10
Log Loss : 1.8073878773647039
for alpha = 100
Log Loss : 1.8089331810021625
```

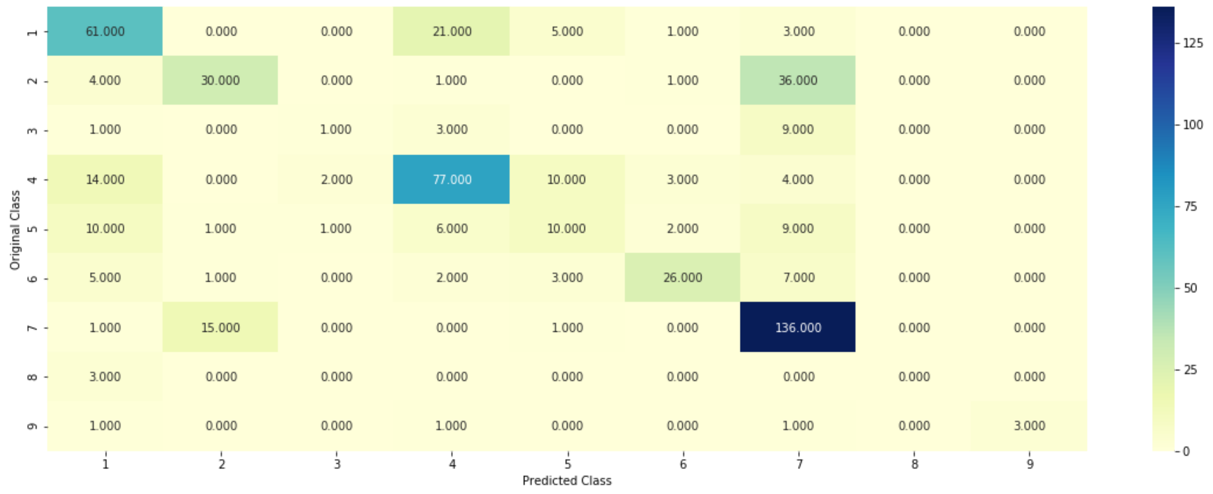


```
For values of best alpha =  0.0001 The train log loss is: 0.59472803293
74434
For values of best alpha =  0.0001 The cross validation log loss is: 1.
071783239542718
```

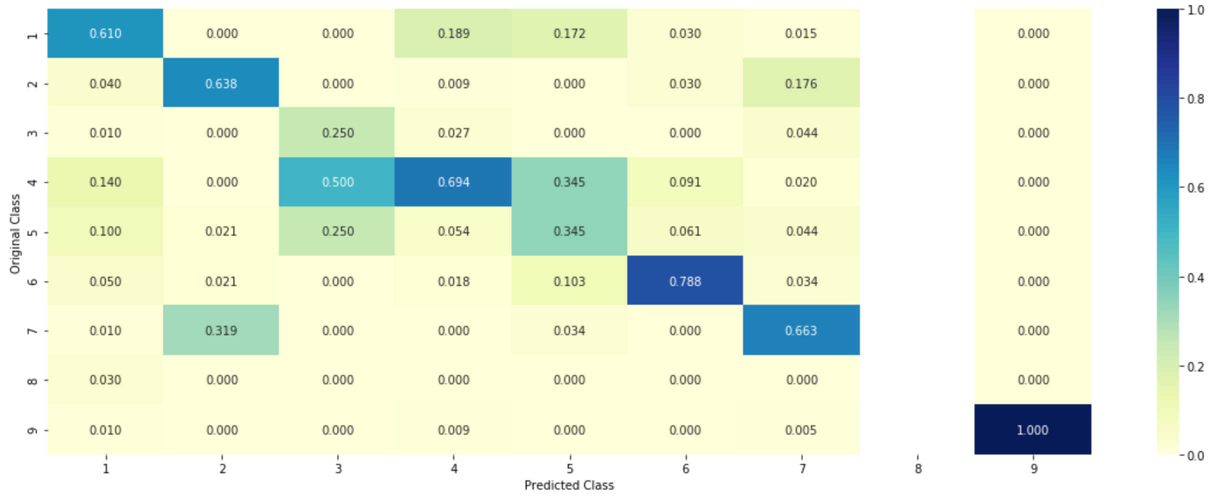
```
For values of best alpha =  0.0001 The test log loss is: 0.990118304992
4845
```

In [146]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

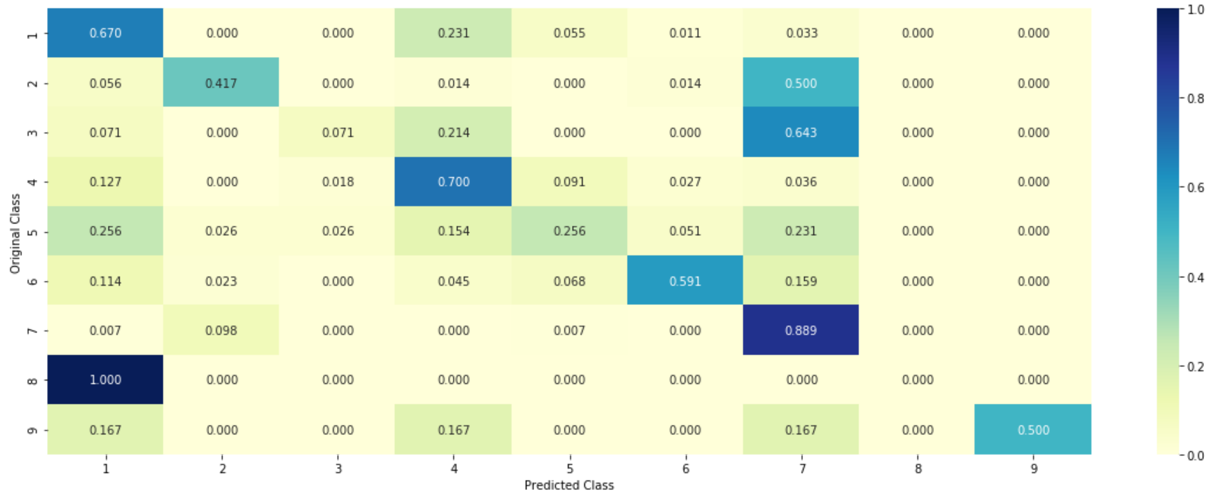
```
Log loss : 1.071783239542718
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



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



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



In [92]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
ry point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

In [147]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 500
no_feature = 1000
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tfidf(indices[0], test_df['TEXT'].iloc[test_point_
index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc
[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[5.573e-01 3.100e-03 2.630e-02 9.270e-0
2 3.162e-01 1.500e-03 1.200e-03
  1.600e-03 1.000e-04]]
```

```
 1.000e-05 1.000e-04]]
Actual Class : 1
--------------------------------------------------
285 Text feature [12] present in test data point [True]
512 Text feature [09] present in test data point [True]
559 Text feature [0886] present in test data point [True]
660 Text feature [0008] present in test data point [True]
682 Text feature [105] present in test data point [True]
Out of the top  1000  features  5 are present in query point
```

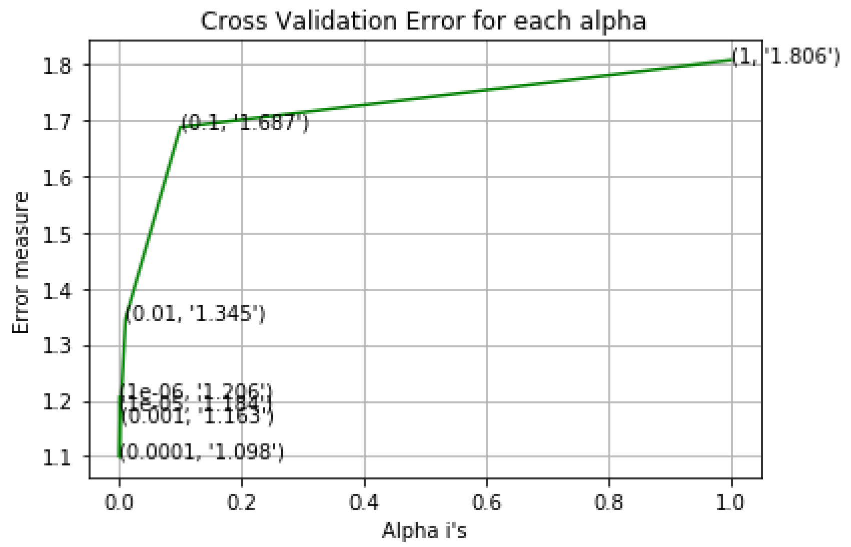
LR WITHOUT CLASS BALANCING

In [142]:

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

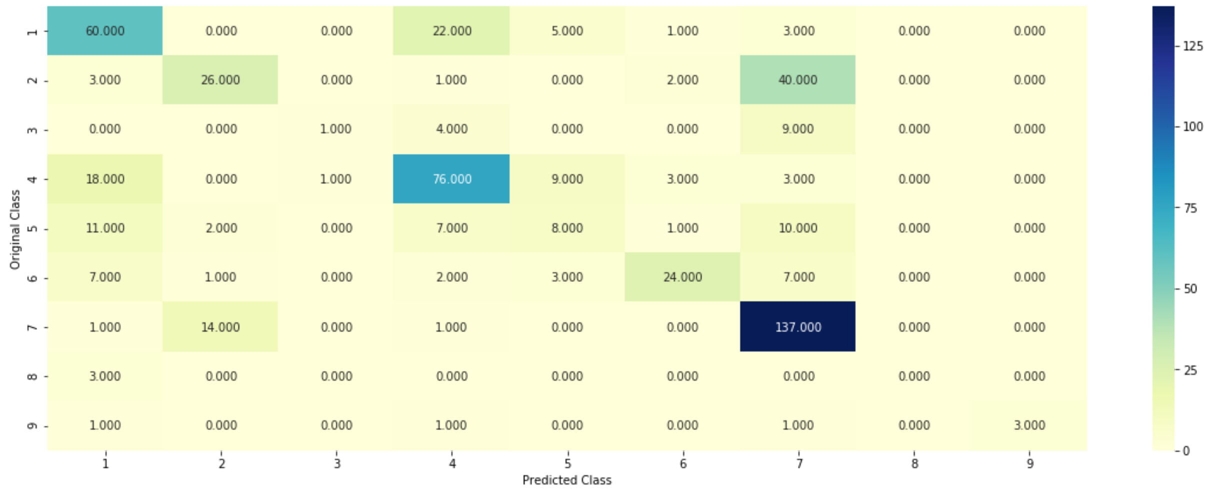
```
for alpha = 1e-06
Log Loss : 1.2056043300866421
for alpha = 1e-05
Log Loss : 1.1844814098459369
for alpha = 0.0001
Log Loss : 1.0982240658305977
for alpha = 0.001
Log Loss : 1.1633042306643904
for alpha = 0.01
Log Loss : 1.3452681280145753
for alpha = 0.1
Log Loss : 1.6870075013833372
for alpha = 1
Log Loss : 1.8062937757896782
```



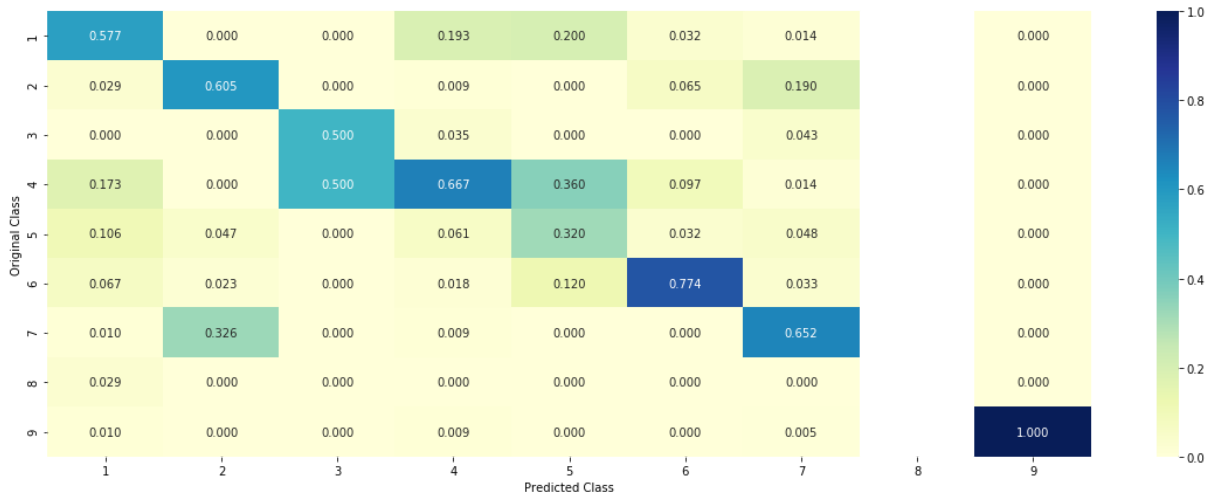
```
For values of best alpha =  0.0001 The train log loss is: 0.58279794708
2412
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0982240658305977
For values of best alpha =  0.0001 The test log loss is: 1.006412144758
9
```

In [143]:
```
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_o
nehotCoding, cv_y, clf)
```

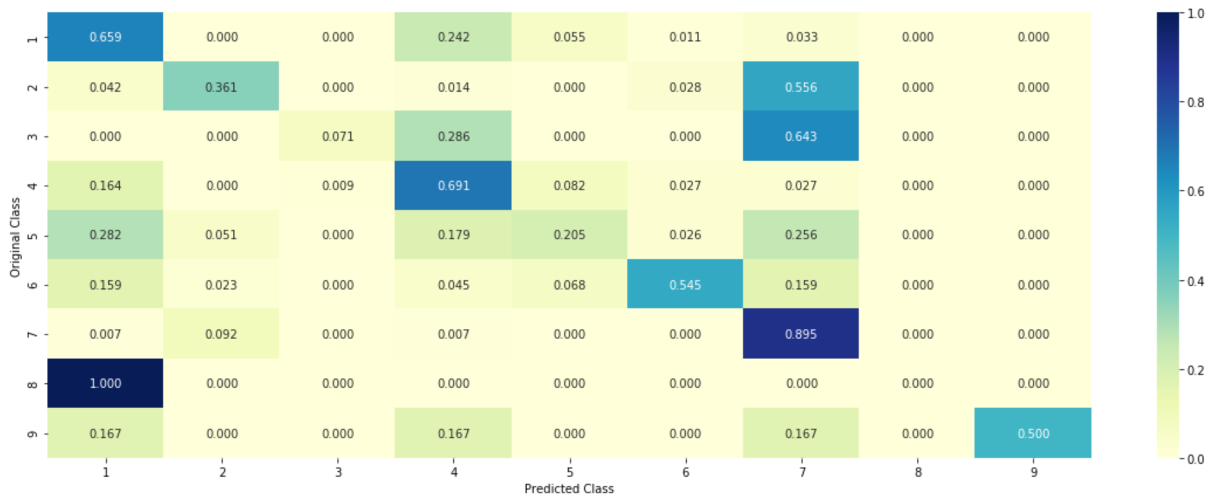
```
Log loss : 1.0982240658305977
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



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



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



In [144]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 200
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tfidf(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: [[5.780e-02 6.600e-03 3.100e-03 8.918e-0
1 1.320e-02 3.800e-03 2.090e-02
  2.400e-03 4.000e-04]]
```

```
Actual Class : 4
--------------------------------------------------
136 Text feature [13] present in test data point [True]
219 Text feature [122] present in test data point [True]
270 Text feature [064] present in test data point [True]
299 Text feature [101] present in test data point [True]
335 Text feature [007] present in test data point [True]
351 Text feature [105] present in test data point [True]
407 Text feature [002] present in test data point [True]
448 Text feature [0075] present in test data point [True]
482 Text feature [001] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

Using CountVectorizer with bigram and trigram

In [108]:
```
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding_bigram = gene_vectorizer.fit_transform(
train_df['Gene'])
test_gene_feature_onehotCoding_bigram = gene_vectorizer.transform(test_
df['Gene'])
cv_gene_feature_onehotCoding_bigram = gene_vectorizer.transform(cv_df[
'Gene'])
```

In [109]:
```
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding_bigram = variation_vectorizer.fit_
transform(train_df['Variation'])
test_variation_feature_onehotCoding_bigram = variation_vectorizer.trans
form(test_df['Variation'])
cv_variation_feature_onehotCoding_bigram = variation_vectorizer.transfo
rm(cv_df['Variation'])
```

In [110]:
```
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding_bigram = text_vectorizer.fit_transform(
train_df['TEXT'])
# getting all the feature names (words)
#train_text_features= text_vectorizer.get_feature_names()
```

```
# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and
 returns (1*number of features) vector
train_text_feature_onehotCoding_bigram = normalize(train_text_feature_o
nehotCoding_bigram, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding_bigram = text_vectorizer.transform(test_
df['TEXT'])
# don't forget to normalize every feature
test_text_feature_onehotCoding_bigram = normalize(test_text_feature_one
hotCoding_bigram, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding_bigram = text_vectorizer.transform(cv_df[
'TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding_bigram = normalize(cv_text_feature_onehotC
oding_bigram, axis=0)
```

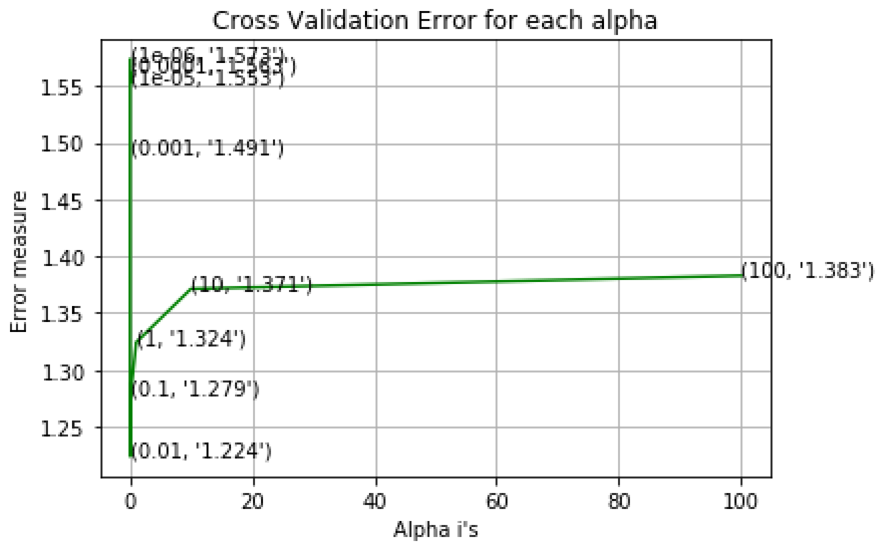
In [111]:

```
train_gene_var_onehotCoding_bigram = hstack((train_gene_feature_onehotC
oding_bigram,train_variation_feature_onehotCoding_bigram))
test_gene_var_onehotCoding_bigram = hstack((test_gene_feature_onehotCod
ing_bigram,test_variation_feature_onehotCoding_bigram))
cv_gene_var_onehotCoding_bigram = hstack((cv_gene_feature_onehotCoding_
bigram,cv_variation_feature_onehotCoding_bigram))

train_x_onehotCoding_bigram = hstack((train_gene_var_onehotCoding_bigra
m, train_text_feature_onehotCoding_bigram)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding_bigram = hstack((test_gene_var_onehotCoding_bigram,
 test_text_feature_onehotCoding_bigram)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding_bigram = hstack((cv_gene_var_onehotCoding_bigram, cv_
text_feature_onehotCoding_bigram)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

LOGISTIC REGRESSION WITH CLASS BALANCING AND USING BIGRAM

In [112]:
```
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_bigram, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_bigram, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_bigram)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

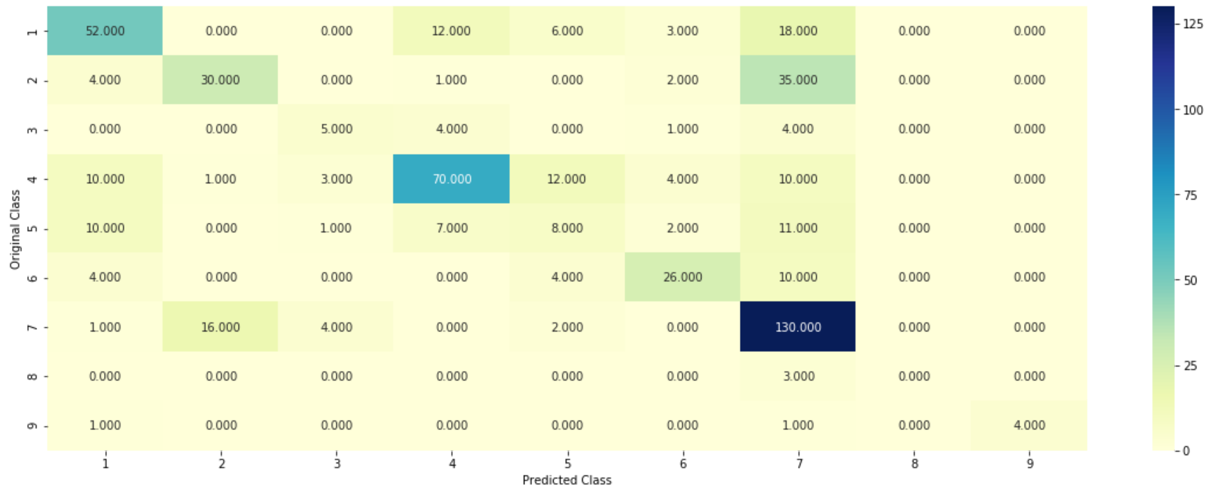
```
for alpha = 1e-06
Log Loss : 1.5730414450985997
for alpha = 1e-05
Log Loss : 1.552955134257466
for alpha = 0.0001
Log Loss : 1.5627526359761572
for alpha = 0.001
Log Loss : 1.4911550657109236
for alpha = 0.01
Log Loss : 1.2236850836121171
for alpha = 0.1
Log Loss : 1.2789896201021673
for alpha = 1
Log Loss : 1.3239395998659487
for alpha = 10
Log Loss : 1.3712552514164116
for alpha = 100
Log Loss : 1.3829123916913362
```



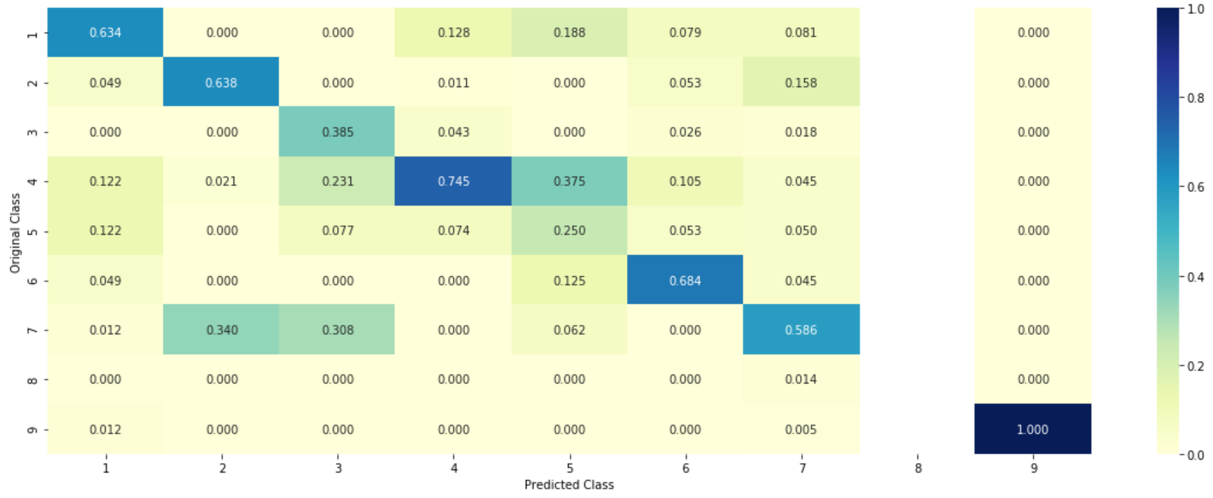
```
For values of best alpha =  0.01 The train log loss is: 0.8601777610201
468
For values of best alpha =  0.01 The cross validation log loss is: 1.22
36850836121171
For values of best alpha =  0.01 The test log loss is: 1.21539741218648
92
```

In [113]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_bigram, train_y,
 cv_x_onehotCoding_bigram, cv_y, clf)
```

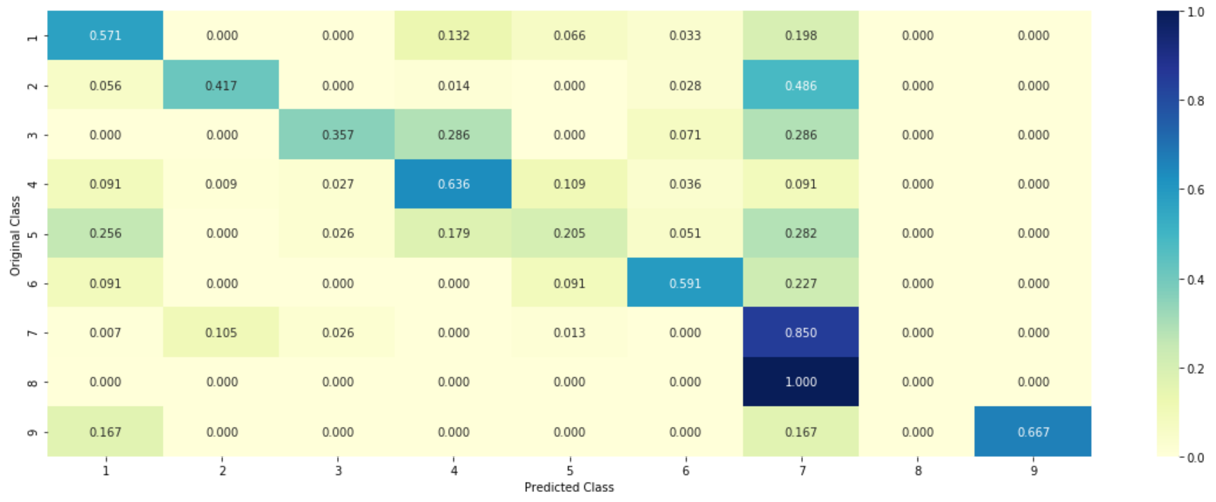
```
Log loss : 1.2236850836121171
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.571	0.000	0.000	0.132	0.066	0.033	0.198	0.000	0.000
2	0.056	0.417	0.000	0.014	0.000	0.028	0.486	0.000	0.000
3	0.000	0.000	0.357	0.286	0.000	0.071	0.286	0.000	0.000
4	0.091	0.009	0.027	0.636	0.109	0.036	0.091	0.000	0.000
5	0.256	0.000	0.026	0.179	0.205	0.051	0.282	0.000	0.000
6	0.091	0.000	0.000	0.000	0.091	0.591	0.227	0.000	0.000
7	0.007	0.105	0.026	0.000	0.013	0.000	0.850	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.167	0.000	0.000	0.000	0.000	0.000	0.167	0.000	0.667

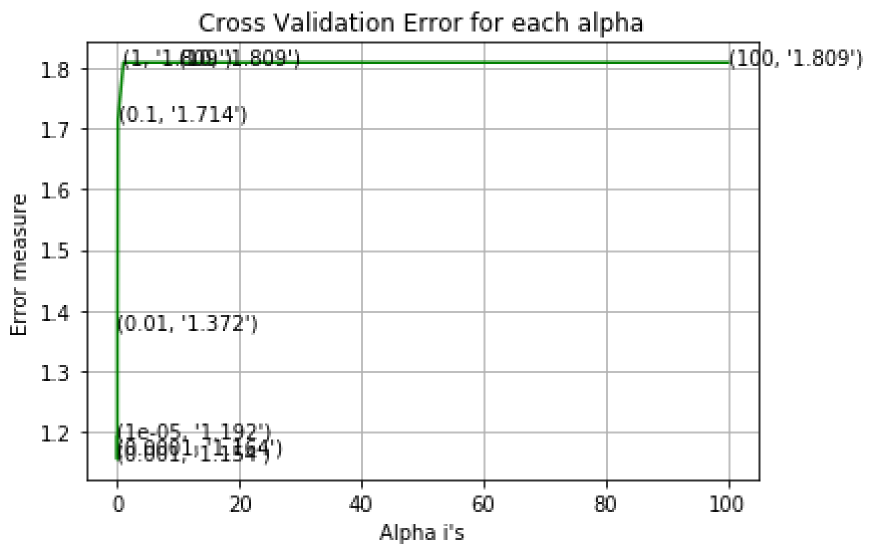
Linear SVM

In [127]:
```
alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='bal
anced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2'
, loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='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)

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

```
for C = 1e-05
Log Loss : 1.1924750398240764
for C = 0.0001
Log Loss : 1.1644093502351454
for C = 0.001
Log Loss : 1.1541324741416927
for C = 0.01
Log Loss : 1.3720471947767994
for C = 0.1
```

```
Log Loss : 1.7141841503386108
for C = 1
Log Loss : 1.8093712686489547
for C = 10
Log Loss : 1.8093711373933496
for C = 100
Log Loss : 1.8093759039024806
```

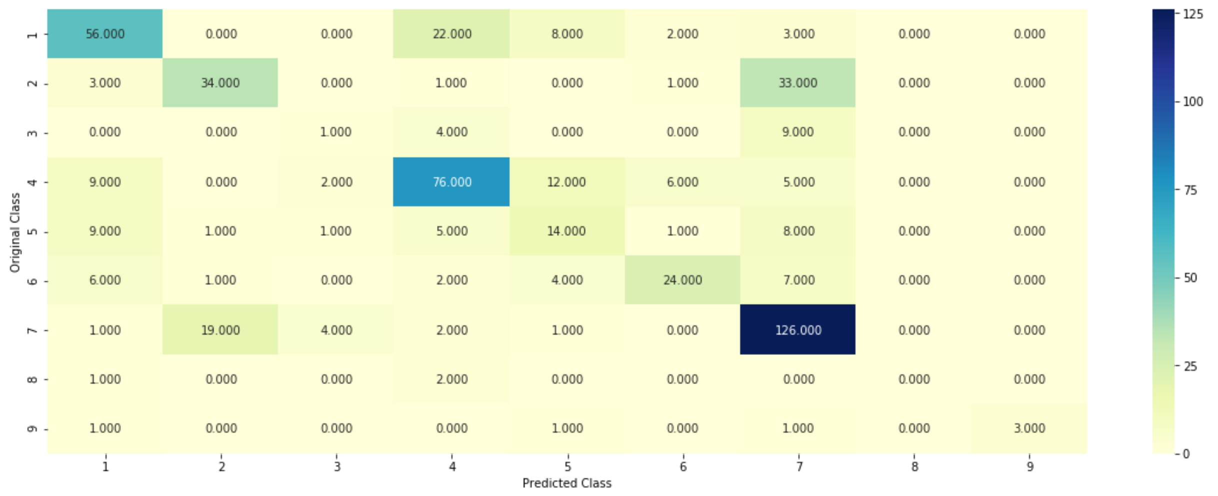


```
For values of best alpha =  0.001 The train log loss is: 0.800920004153
4785
For values of best alpha =  0.001 The cross validation log loss is: 1.1
541324741416927
For values of best alpha =  0.001 The test log loss is: 1.0700818180951
652
```

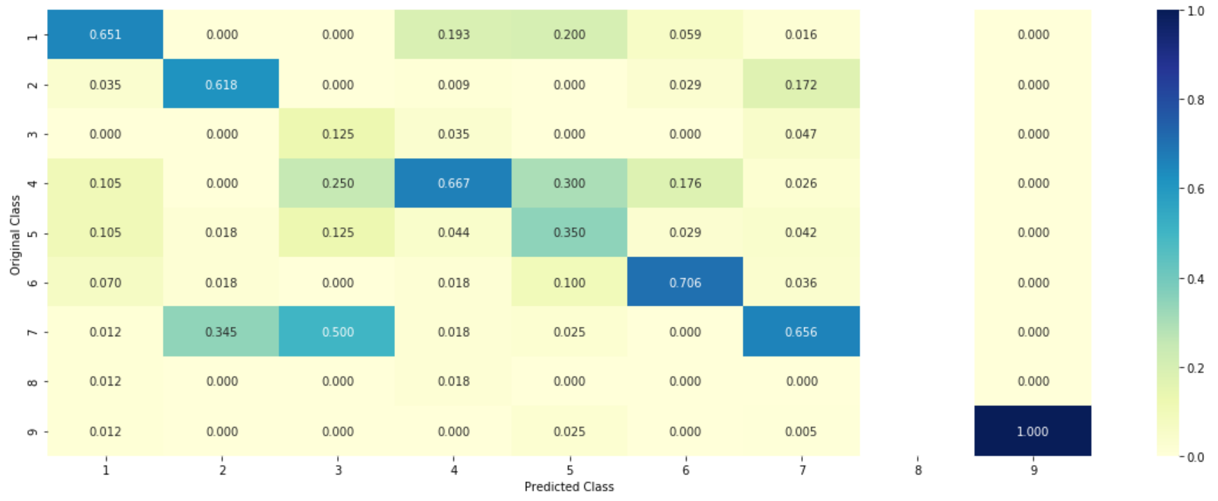
In [128]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

```
Log loss : 1.1541324741416927

Number of mis-classified points : 0.37218045112781956
```

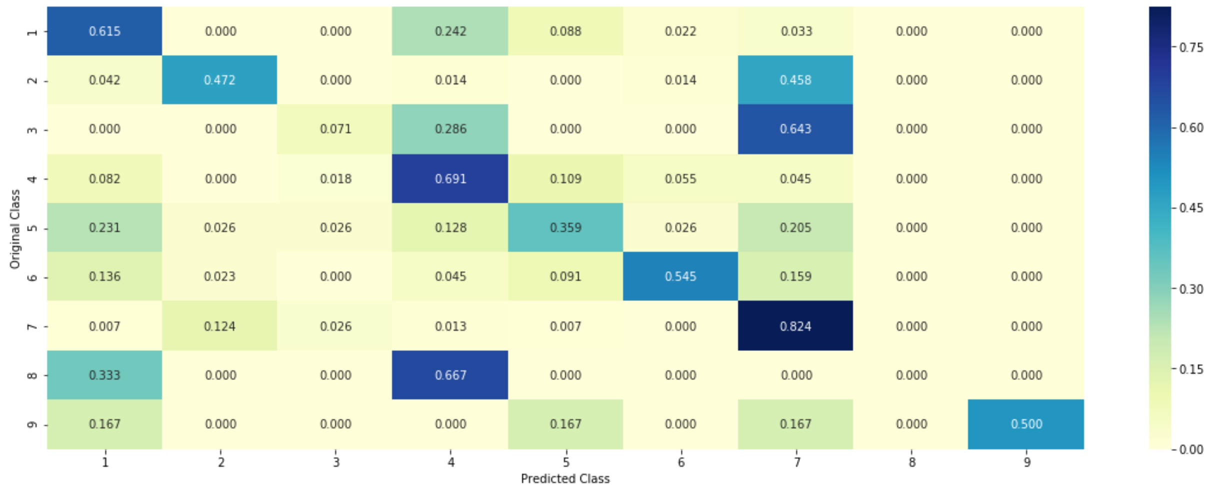
```
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.615	0.000	0.000	0.242	0.088	0.022	0.033	0.000	0.000
2	0.042	0.472	0.000	0.014	0.000	0.014	0.458	0.000	0.000
3	0.000	0.000	0.071	0.286	0.000	0.000	0.643	0.000	0.000
4	0.082	0.000	0.018	0.691	0.109	0.055	0.045	0.000	0.000
5	0.231	0.026	0.026	0.128	0.359	0.026	0.205	0.000	0.000
6	0.136	0.023	0.000	0.045	0.091	0.545	0.159	0.000	0.000
7	0.007	0.124	0.026	0.013	0.007	0.000	0.824	0.000	0.000
8	0.333	0.000	0.000	0.667	0.000	0.000	0.000	0.000	0.000
9	0.167	0.000	0.000	0.000	0.167	0.000	0.167	0.000	0.500

In [139]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 50
#test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tfidf(indices[0], test_df['TEXT'].iloc[test_point_
index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc
[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[8.595e-01 2.910e-02 2.300e-03 2.600e-0
3 3.900e-03 5.300e-03 9.550e-02
  1.400e-03 5.000e-04]]
Actual Class : 2

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

```
310 Text feature [097] present in test data point [True]
318 Text feature [02] present in test data point [True]
332 Text feature [12] present in test data point [True]
367 Text feature [038] present in test data point [True]
488 Text feature [1014] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

In [140]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tfidf(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: [[0.0562 0.0528 0.0117 0.7645 0.0363 0.0
165 0.0565 0.0034 0.0021]]
Actual Class : 4
--------------------------------------------------
161 Text feature [07] present in test data point [True]
287 Text feature [13] present in test data point [True]
428 Text feature [02] present in test data point [True]
472 Text feature [04] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

In [119]:

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(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.cl
asses_, 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_, ep
s=1e-15))
```

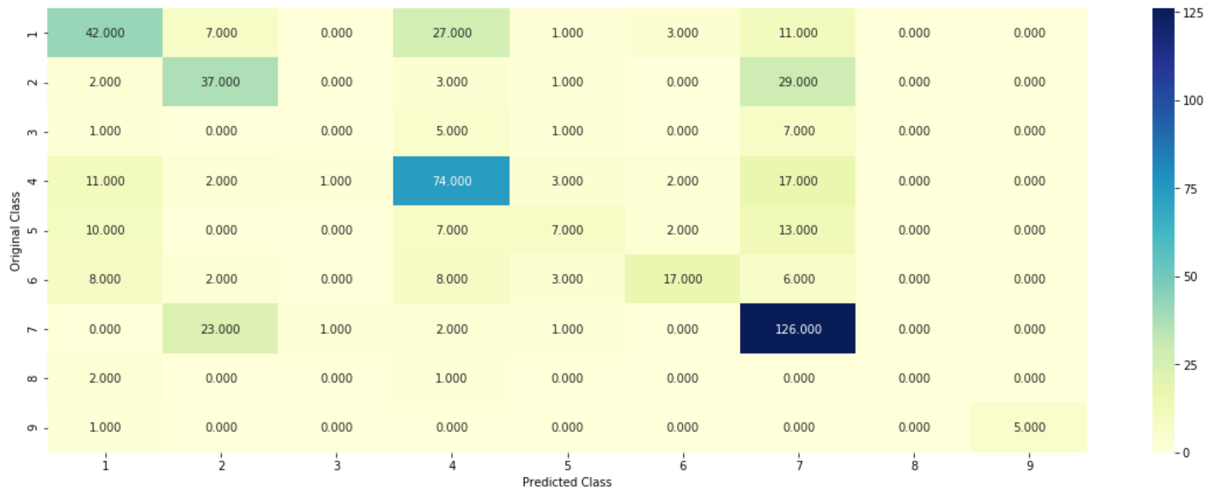
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2397055221067135
for n_estimators = 100 and max depth =  10
Log Loss : 1.2599002601414668
for n_estimators = 200 and max depth =  5
Log Loss : 1.2333371609472659
for n_estimators = 200 and max depth =  10
Log Loss : 1.253699119061069
for n_estimators = 500 and max depth =  5
Log Loss : 1.2296755045750676
for n_estimators = 500 and max depth =  10
Log Loss : 1.2531199340574077
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2252658400197152
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2469103481560504
for n_estimators = 2000 and max depth =  5
Log Loss : 1.223526323570853
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2449741220182315
For values of best estimator =  2000 The train log loss is: 0.859726333
8115166
For values of best estimator =  2000 The cross validation log loss is:
1.223526323570853
For values of best estimator =  2000 The test log loss is: 1.1921106986
524912
```

In [163]:

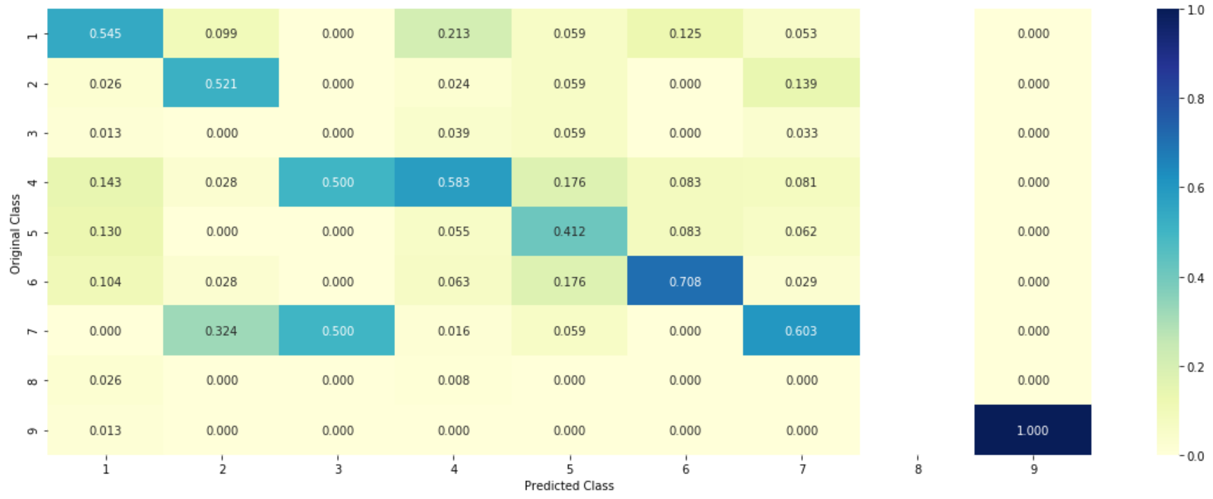
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth = max_depth[int(best_alpha%2)], random_state=4
2, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

```
Log loss : 1.2105319540134507
Number of mis-classified points : 0.42105263157894735
```

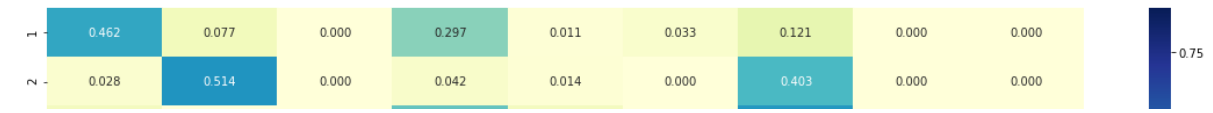
```
-------------------- Confusion matrix --------------------
```

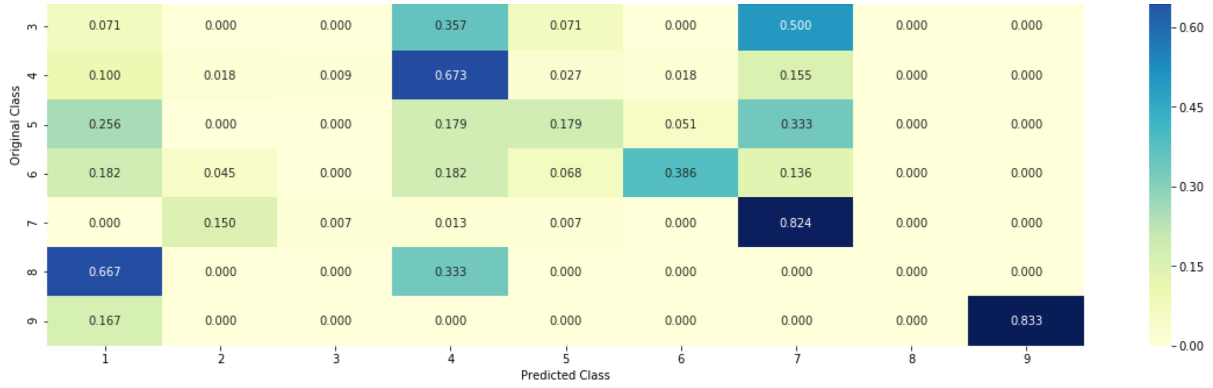


```
-------------------- Precision matrix (Columm Sum=1) ------------------
--
```



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





In [164]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(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_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0576 0.1812 0.0184 0.0387 0.0443 0.0
336 0.616  0.0063 0.0039]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]

1 Text feature [inhibitors] present in test data point [True]
```

```
1 Text feature [inhibitors] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [inhibitor] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [activated] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
13 Text feature [function] present in test data point [True]
15 Text feature [oncogenic] present in test data point [True]
16 Text feature [growth] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
18 Text feature [erk] present in test data point [True]
19 Text feature [therapy] present in test data point [True]
20 Text feature [receptor] present in test data point [True]
21 Text feature [kinases] present in test data point [True]
23 Text feature [therapeutic] present in test data point [True]
24 Text feature [akt] present in test data point [True]
25 Text feature [loss] present in test data point [True]
26 Text feature [downstream] present in test data point [True]
27 Text feature [constitutively] present in test data point [True]
28 Text feature [inhibition] present in test data point [True]
29 Text feature [treated] present in test data point [True]
30 Text feature [resistance] present in test data point [True]
31 Text feature [yeast] present in test data point [True]
32 Text feature [activate] present in test data point [True]
35 Text feature [months] present in test data point [True]
36 Text feature [proliferation] present in test data point [True]
37 Text feature [efficacy] present in test data point [True]
38 Text feature [survival] present in test data point [True]
39 Text feature [patients] present in test data point [True]
40 Text feature [stability] present in test data point [True]
44 Text feature [cells] present in test data point [True]
45 Text feature [transforming] present in test data point [True]
46 Text feature [functional] present in test data point [True]
47 Text feature [dose] present in test data point [True]
48 Text feature [protein] present in test data point [True]
49 Text feature [inhibited] present in test data point [True]

50 Text feature [extracellular] present in test data point [True]
```

```
51 Text feature [phospho] present in test data point [True]
55 Text feature [cell] present in test data point [True]
56 Text feature [sensitivity] present in test data point [True]
57 Text feature [ligand] present in test data point [True]
61 Text feature [clinical] present in test data point [True]
62 Text feature [egfr] present in test data point [True]
64 Text feature [oncogene] present in test data point [True]
65 Text feature [ic50] present in test data point [True]
66 Text feature [phosphatase] present in test data point [True]
67 Text feature [serum] present in test data point [True]
68 Text feature [phosphorylated] present in test data point [True]
70 Text feature [pathogenic] present in test data point [True]
71 Text feature [resistant] present in test data point [True]
72 Text feature [expressing] present in test data point [True]
74 Text feature [ras] present in test data point [True]
75 Text feature [lines] present in test data point [True]
79 Text feature [predicted] present in test data point [True]
81 Text feature [atp] present in test data point [True]
83 Text feature [active] present in test data point [True]
86 Text feature [potential] present in test data point [True]
88 Text feature [independent] present in test data point [True]
95 Text feature [p53] present in test data point [True]
96 Text feature [expected] present in test data point [True]
99 Text feature [amplification] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

In [165]:

```
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_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.218  0.1292 0.0259 0.1951 0.077  0.0
622 0.2715 0.0089 0.0123]]
Actuall Class : 7
--------------------------------------------------
2 Text feature [activating] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
8 Text feature [suppressor] present in test data point [True]
9 Text feature [activated] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [missense] present in test data point [True]
12 Text feature [nonsense] present in test data point [True]
13 Text feature [function] present in test data point [True]
15 Text feature [oncogenic] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
18 Text feature [erk] present in test data point [True]
25 Text feature [loss] present in test data point [True]
29 Text feature [treated] present in test data point [True]
30 Text feature [resistance] present in test data point [True]
32 Text feature [activate] present in test data point [True]
36 Text feature [proliferation] present in test data point [True]
39 Text feature [patients] present in test data point [True]
44 Text feature [cells] present in test data point [True]
46 Text feature [functional] present in test data point [True]
48 Text feature [protein] present in test data point [True]
55 Text feature [cell] present in test data point [True]
56 Text feature [sensitivity] present in test data point [True]
60 Text feature [unstable] present in test data point [True]
61 Text feature [clinical] present in test data point [True]
62 Text feature [egfr] present in test data point [True]
64 Text feature [oncogene] present in test data point [True]
68 Text feature [phosphorylated] present in test data point [True]
71 Text feature [resistant] present in test data point [True]
74 Text feature [ras] present in test data point [True]
75 Text feature [lines] present in test data point [True]
79 Text feature [predicted] present in test data point [True]
86 Text feature [potential] present in test data point [True]
```

```
89 Text feature [harboring] present in test data point [True]
96 Text feature [expected] present in test data point [True]
Out of the top  100  features  36 are present in query point
```

Hyperparameter tuning with responsecoding

In [166]:

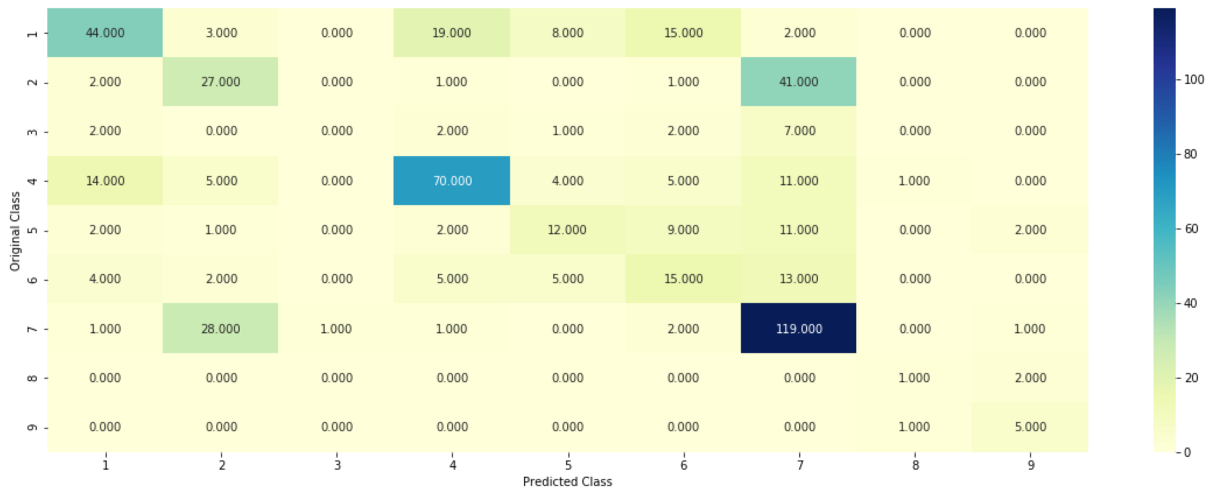
```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=
clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='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 tra
in 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 cro
ss validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t 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 : 2.0547777252913018
for n_estimators = 10 and max depth =  3
Log Loss : 1.953676653855976
for n_estimators = 10 and max depth =  5
Log Loss : 1.6698205234111465
for n_estimators = 10 and max depth =  10
Log Loss : 1.7112663974916105
for n_estimators = 50 and max depth =  2
Log Loss : 1.7215718569820495
for n_estimators = 50 and max depth =  3
Log Loss : 1.6160301304803026
for n_estimators = 50 and max depth =  5
Log Loss : 1.423140648060624
for n_estimators = 50 and max depth =  10
Log Loss : 1.8253986350439335
for n_estimators = 100 and max depth =  2
Log Loss : 1.670223636643592
for n_estimators = 100 and max depth =  3
Log Loss : 1.5859465652005753
for n_estimators = 100 and max depth =  5
```

```
Log Loss : 1.377211615052794
for n_estimators = 100 and max depth =  10
Log Loss : 1.8147254126494394
for n_estimators = 200 and max depth =  2
Log Loss : 1.681051319989161
for n_estimators = 200 and max depth =  3
Log Loss : 1.518154970450883
for n_estimators = 200 and max depth =  5
Log Loss : 1.3953982844263015
for n_estimators = 200 and max depth =  10
Log Loss : 1.7672061285434486
for n_estimators = 500 and max depth =  2
Log Loss : 1.7676655333024185
for n_estimators = 500 and max depth =  3
Log Loss : 1.5628881840035729
for n_estimators = 500 and max depth =  5
Log Loss : 1.4323275867567167
for n_estimators = 500 and max depth =  10
Log Loss : 1.8264945799489345
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6658299947757949
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5517527369982589
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4719151667749484
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8299654896823927
For values of best alpha =  100 The train log loss is: 0.06369862279907
79
For values of best alpha =  100 The cross validation log loss is: 1.377
211615052794
For values of best alpha =  100 The test log loss is: 1.272079213659028
2
```

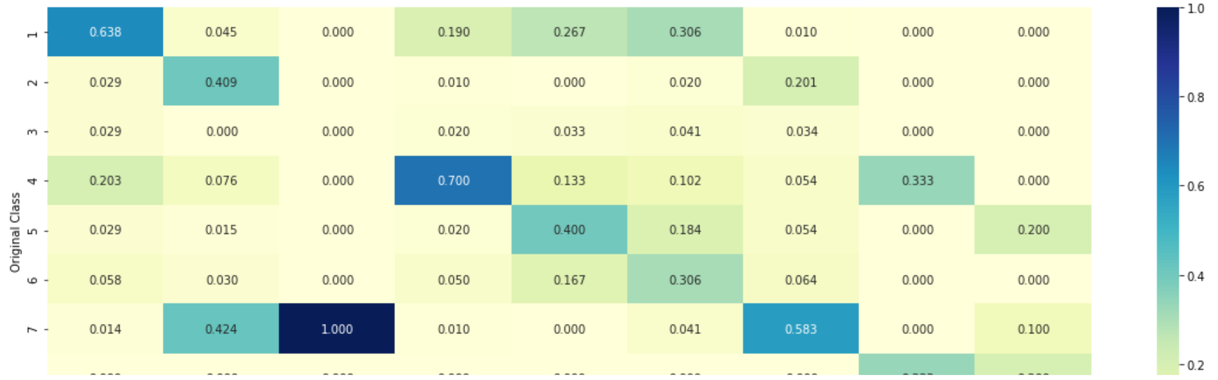
In [167]:
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='au
to',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_
responseCoding,cv_y, clf)
```

```
Log loss : 1.377211615052794
Number of mis-classified points : 0.44924812030075 19
-------------------- Confusion matrix --------------------
```

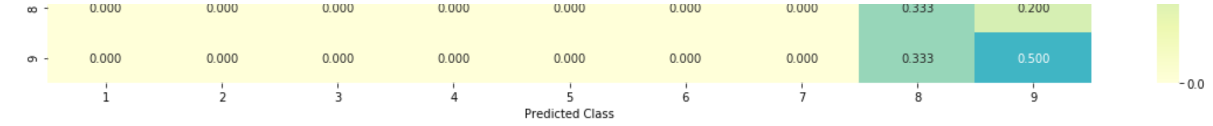


	1	2	3	4	5	6	7	8	9
1	44.000	3.000	0.000	19.000	8.000	15.000	2.000	0.000	0.000
2	2.000	27.000	0.000	1.000	0.000	1.000	41.000	0.000	0.000
3	2.000	0.000	0.000	2.000	1.000	2.000	7.000	0.000	0.000
4	14.000	5.000	0.000	70.000	4.000	5.000	11.000	1.000	0.000
5	2.000	1.000	0.000	2.000	12.000	9.000	11.000	0.000	2.000
6	4.000	2.000	0.000	5.000	5.000	15.000	13.000	0.000	0.000
7	1.000	28.000	1.000	1.000	0.000	2.000	119.000	0.000	1.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	5.000

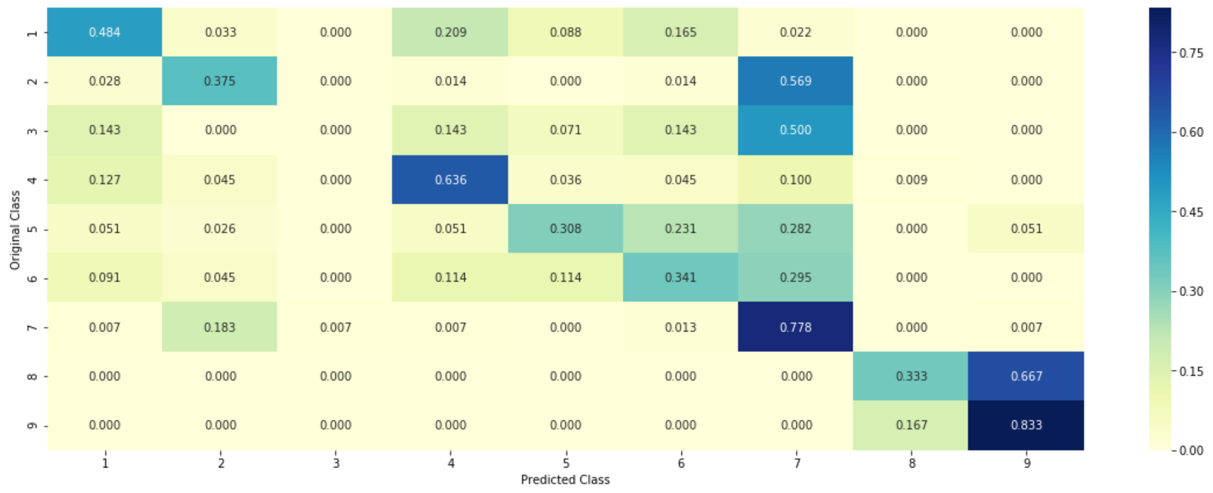
```
-------------------- Precision matrix (Columm Sum=1) -------------------
--
```



	1	2	3	4	5	6	7	8	9
1	0.638	0.045	0.000	0.190	0.267	0.306	0.010	0.000	0.000
2	0.029	0.409	0.000	0.010	0.000	0.020	0.201	0.000	0.000
3	0.029	0.000	0.000	0.020	0.033	0.041	0.034	0.000	0.000
4	0.203	0.076	0.000	0.700	0.133	0.102	0.054	0.333	0.000
5	0.029	0.015	0.000	0.020	0.400	0.184	0.054	0.000	0.200
6	0.058	0.030	0.000	0.050	0.167	0.306	0.064	0.000	0.000
7	0.014	0.424	1.000	0.010	0.000	0.041	0.583	0.000	0.100



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



In [168]:
```python
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='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)
```

```python
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 : 7
Predicted Class Probabilities: [[0.0171 0.2541 0.1928 0.0173 0.0158 0.0
492 0.3767 0.0546 0.0226]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene 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
Text is important feature
Text is important feature


Text is important feature
Text is important feature
```

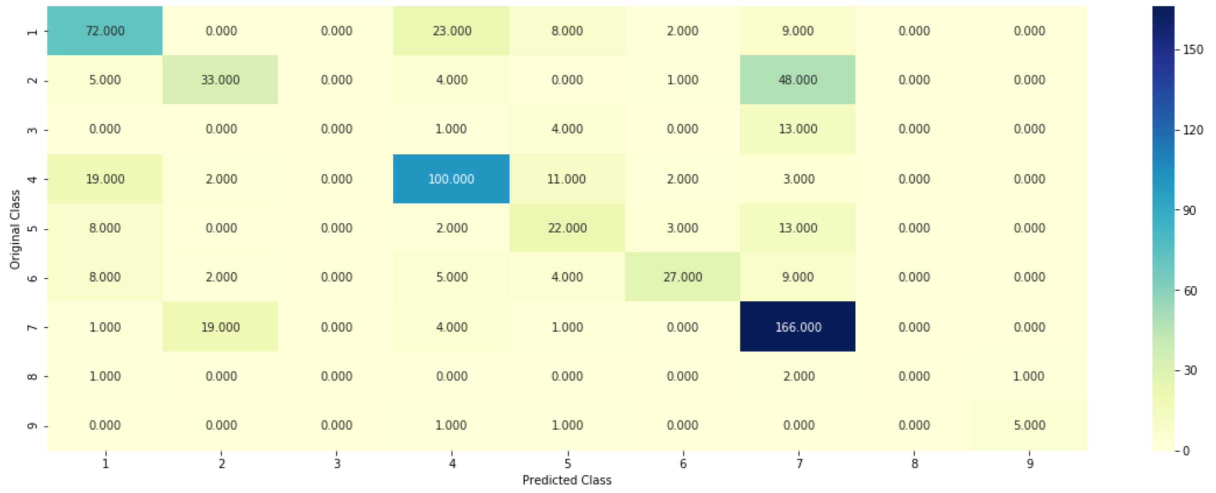
In [120]:
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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_cl
f1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig
_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predic
t_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3
], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.09
Support vector machines : Log Loss: 1.81
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.036
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.517
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.159
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.202
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.338
```

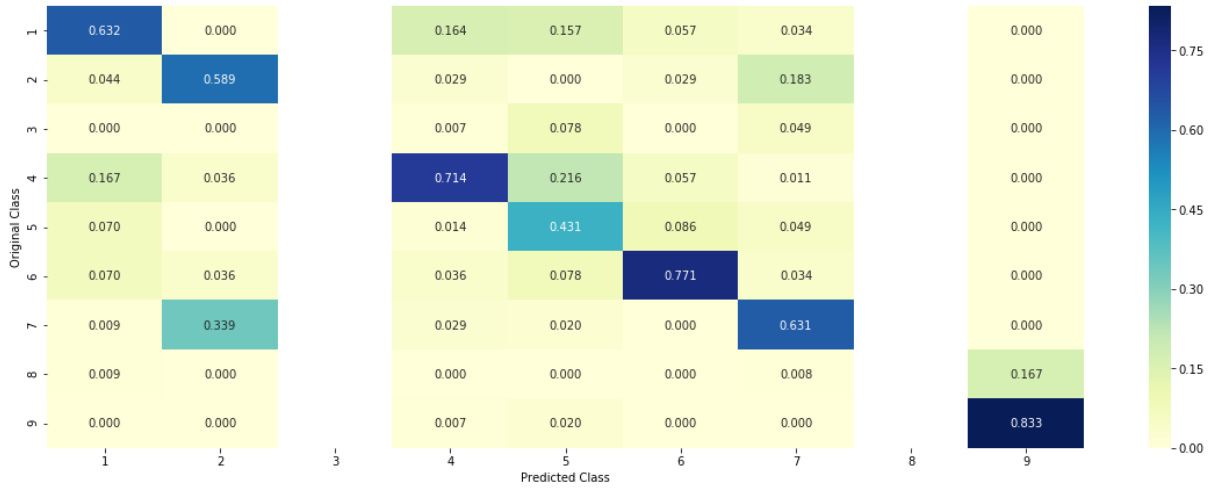
In [121]:

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

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

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

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

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

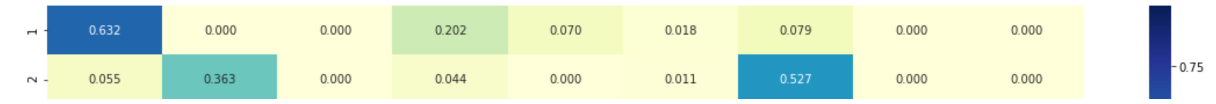
```
Log loss (train) on the stacking classifier : 0.8122457875085958
Log loss (CV) on the stacking classifier : 1.158607936412606
Log loss (test) on the stacking classifier : 1.1439378593731084
Number of missclassified point : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

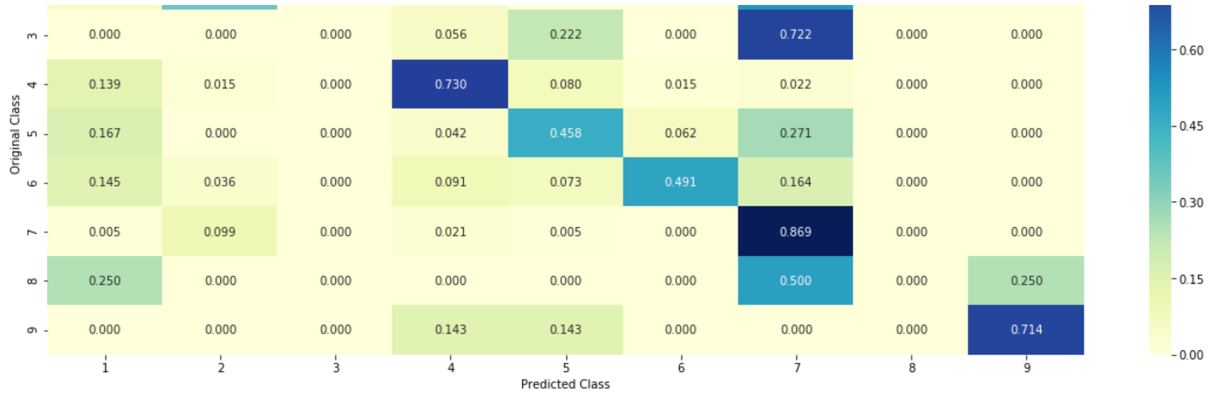


```
-------------------- Precision matrix (Columm Sum=1) ------------------
--
```



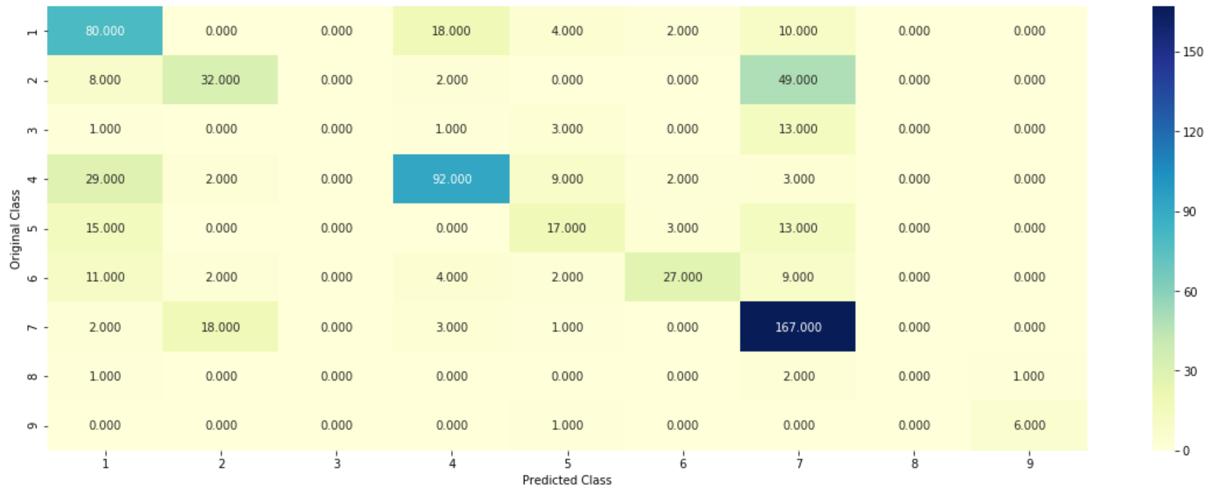
```
-------------------- Recall matrix (Row sum=1) --------------------
```



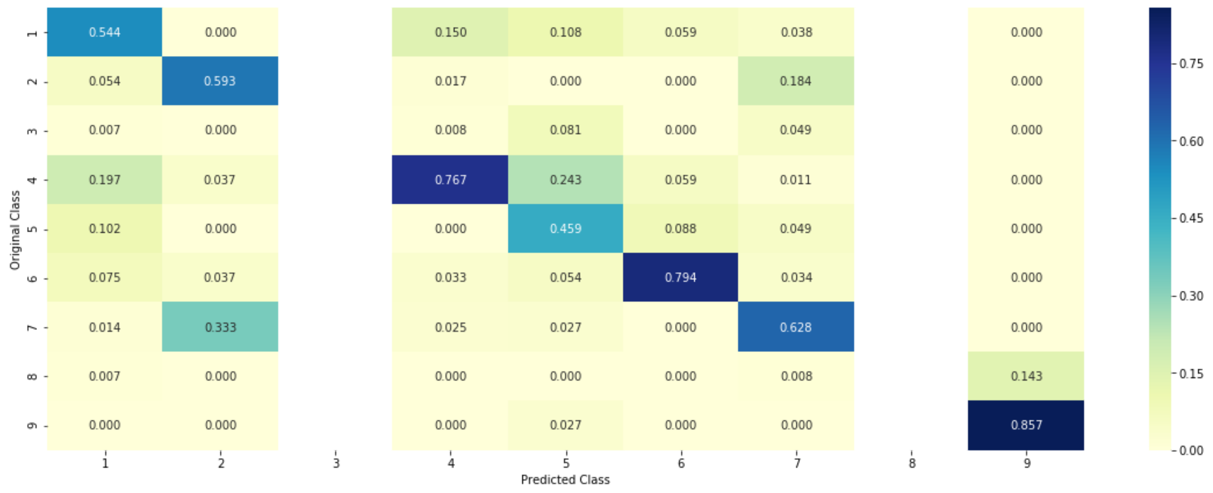


In [122]:
```python
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, v
clf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.pr
edict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vcl
f.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predic
t(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_oneh
otCoding))
```

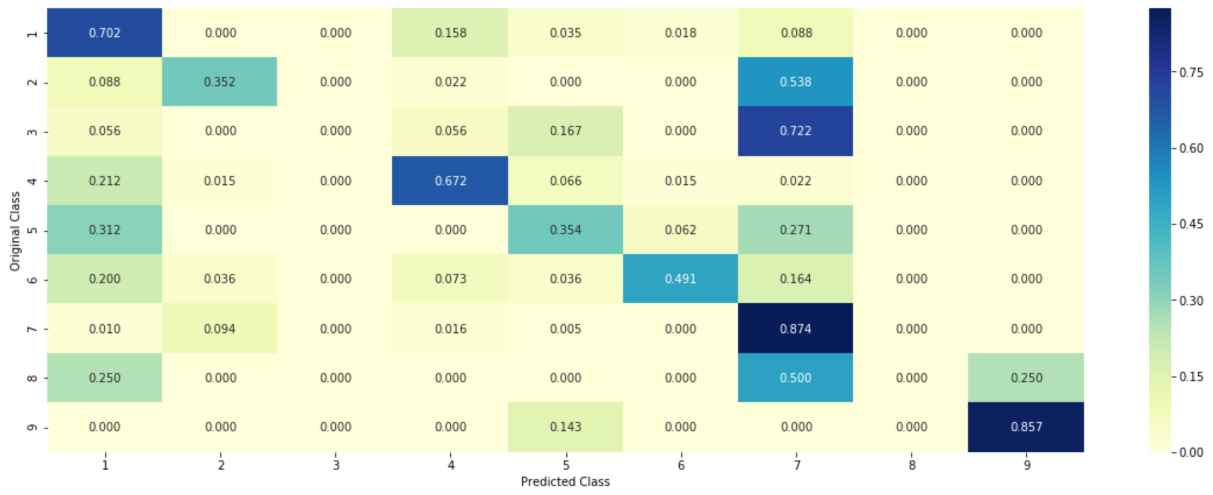
```
Log loss (train) on the VotingClassifier : 0.9509172243980866
Log loss (CV) on the VotingClassifier : 1.2177897593546751
Log loss (test) on the VotingClassifier : 1.175970767941151
Number of missclassified point : 0.3669172932330827
-------------------- Confusion matrix --------------------
```



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



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



Observations

BY using tfidf vectorizer we got a log loss of 0.99 in Logistic regression model with class balancing

we performed tfidf vectorizer on each model and found better logloss (i.e reduced logloss) for each model

we performed count vectorizer with (including bigram) on Logistic regression model and got a logloss of 1.21

the best model we got is Logistic regression with 0.99 test logloss using tfidf vectorization

In [2]:
```
!pip install prettytable
```

```
Requirement already satisfied: prettytable in c:\users\lenovo\anaconda3
\lib\site-packages (0.7.2)
```

```
distributed 1.21.8 requires msgpack, which is not installed.
You are using pip version 10.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade
pip' command.
```

In [8]:
```
from prettytable import PrettyTable
x = PrettyTable(['model' , 'Train logloss' , 'cv logloss' , 'Test loglo
ss' , 'missclassified points'])
x.add_row(['naive bayes',0.798,1.226,1.1999,0.41])
x.add_row(['knn',0.82,1.0873,1.138,0.38])
x.add_row(['logistic regression with class balancing',0.594,1.071,0.99,
0.35])
x.add_row(['logistic regression without class balancing',0.58,1.098,1.0
06,0.37])
x.add_row(['linear svm',0.800,1.154,1.076,0.37])
x.add_row(['random forest',0.859,1.223,1.192,0.42])
x.add_row(['max voting classifier',0.95,1.217,1.176,0.36])

print(x)
```

```
+----------------------------------------------+---------------+--------
----+-------------+-----------------------+
|                     model                    | Train logloss | cv logl
oss | Test logloss | missclassified points |
+----------------------------------------------+---------------+--------
----+-------------+-----------------------+
|                 naive bayes                  |     0.798     |   1.226
    |    1.1999    |          0.41         |
|                     knn                      |      0.82     |   1.087
3   |    1.138     |          0.38         |
|   logistic regression with class balancing   |     0.594     |   1.071
    |     0.99     |          0.35         |
| logistic regression without class balancing |      0.58     |   1.098
    |    1.006     |          0.37         |
|                  linear svm                  |      0.8      |   1.154
```

```
    |    1.076     |          0.37          |
|                random forest                |     0.859     |   1.223
    |    1.192     |          0.42          |
|            max voting classifier            |      0.95     |   1.217
    |    1.176     |          0.36          |
+---------------------------------------------+---------------+--------
----+--------------+------------------------+
```