Diagnosis-Using-ML

May 29, 2019

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")


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

```
C:\Users\shaza\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning:
```

```
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\shaza\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationW
  from numpy.core.umath_tests import inner1d
```

```
In [2]: data = pd.read_csv('training_variants')
        print('Number of data points : ', data.shape[0])
        print('Number of features : ', data.shape[1])
        print('Features : ', data.columns.values)
        data.head()
```

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

Out[2]:

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

```
In [3]: data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],s
        print('Number of data points : ', data_text.shape[0])
        print('Number of features : ', data_text.shape[1])
        print('Features : ', data_text.columns.values)
        data_text.head()
```

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

Out[3]:

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

```
In [4]: # loading stop words from nltk library
        import nltk
        nltk.download('stopwords')
        stop_words = set(stopwords.words('english'))


        def nlp_preprocessing(total_text, index, column):
            if type(total_text) is not int:
```

```
                string = ""
                # replace every special char with space
                total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
                # replace multiple spaces with single space
                total_text = re.sub('\s+',' ', total_text)
                # converting all the chars into lower-case.
                total_text = total_text.lower()

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

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

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\shaza\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 183.0298487 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

```
        0  cyclin dependent kinases cdks regulate variety...
        1  abstract background non small cell lung cancer...
        2  abstract background non small cell lung cancer...
        3  recent evidence demonstrated acquired uniparen...
        4  oncogenic mutations monomeric casitas b lineag...
```

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

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

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

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

In [10]:
```
result[result['ID']==1277]
```

Out[10]:

	ID	Gene	Variation	Class	TEXT
1277	1277	ARID5B	Truncating Mutations	1	ARID5B Truncating Mutations

In [11]:
```
# DATA SPLIT INTO TRAIN(64),CV(16),TEST(20)
```

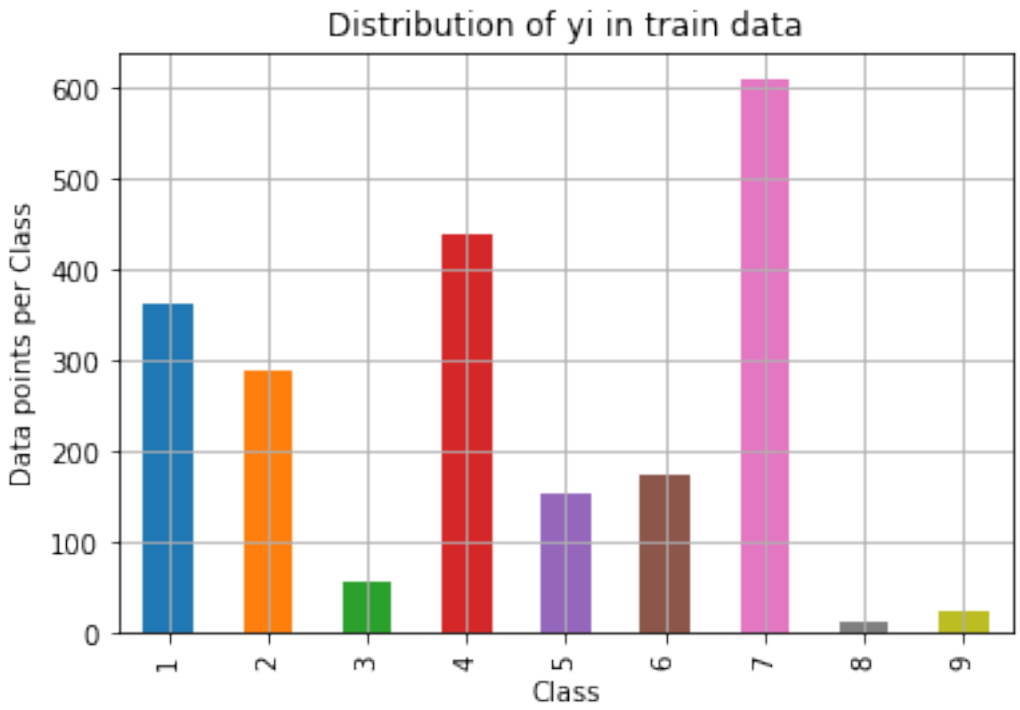
In [12]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

In [13]:
```
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])
```

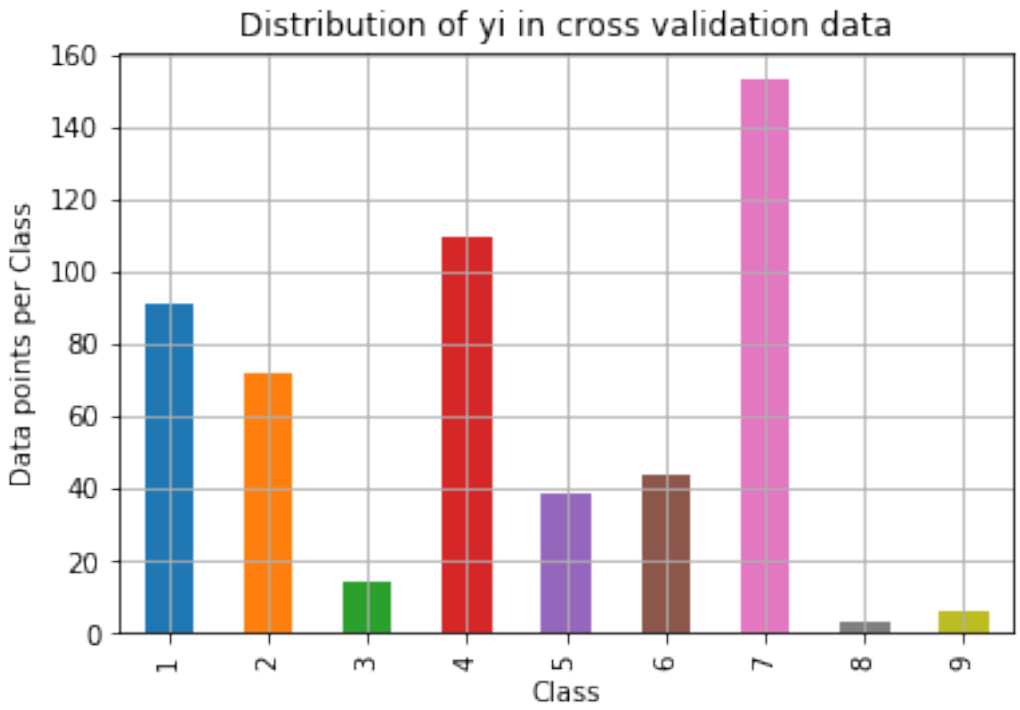
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

In [14]:
```
# SINCE WE WANT THE DISTRIBUTION OF  Y_I'S TO BE ROUGLY SAME IN ALL 3 SPLLITS , SO WE
```

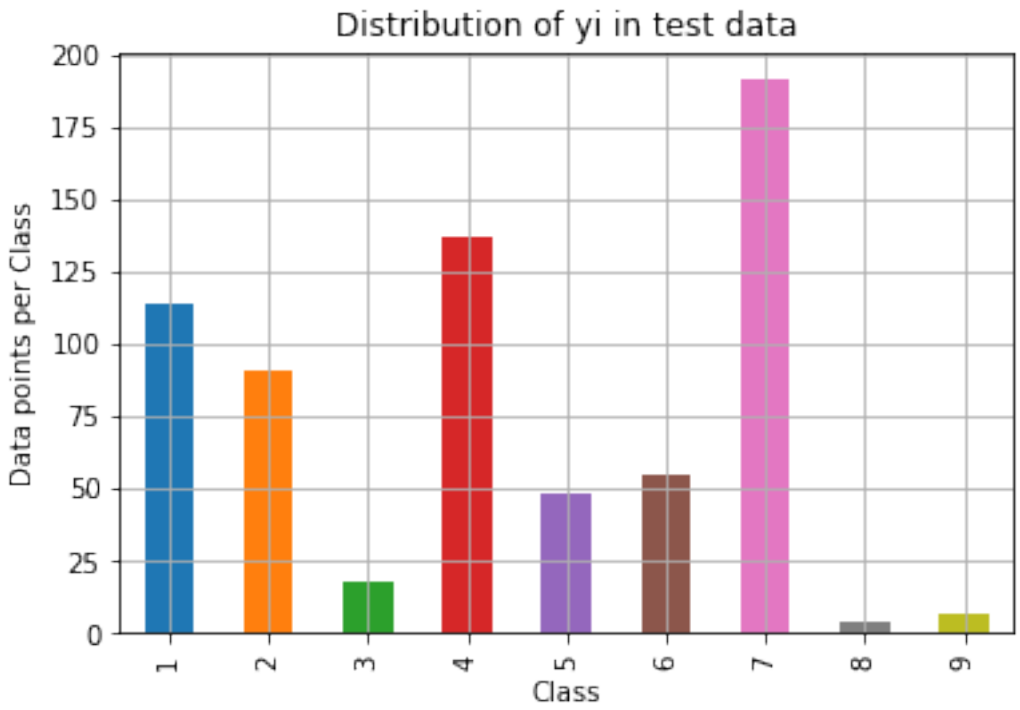
In [15]:
```
train_class_distribution = train_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()

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_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i

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()

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],

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()

sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i]
```




```
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 : 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 %)
------------------------------------------------------------------------
```



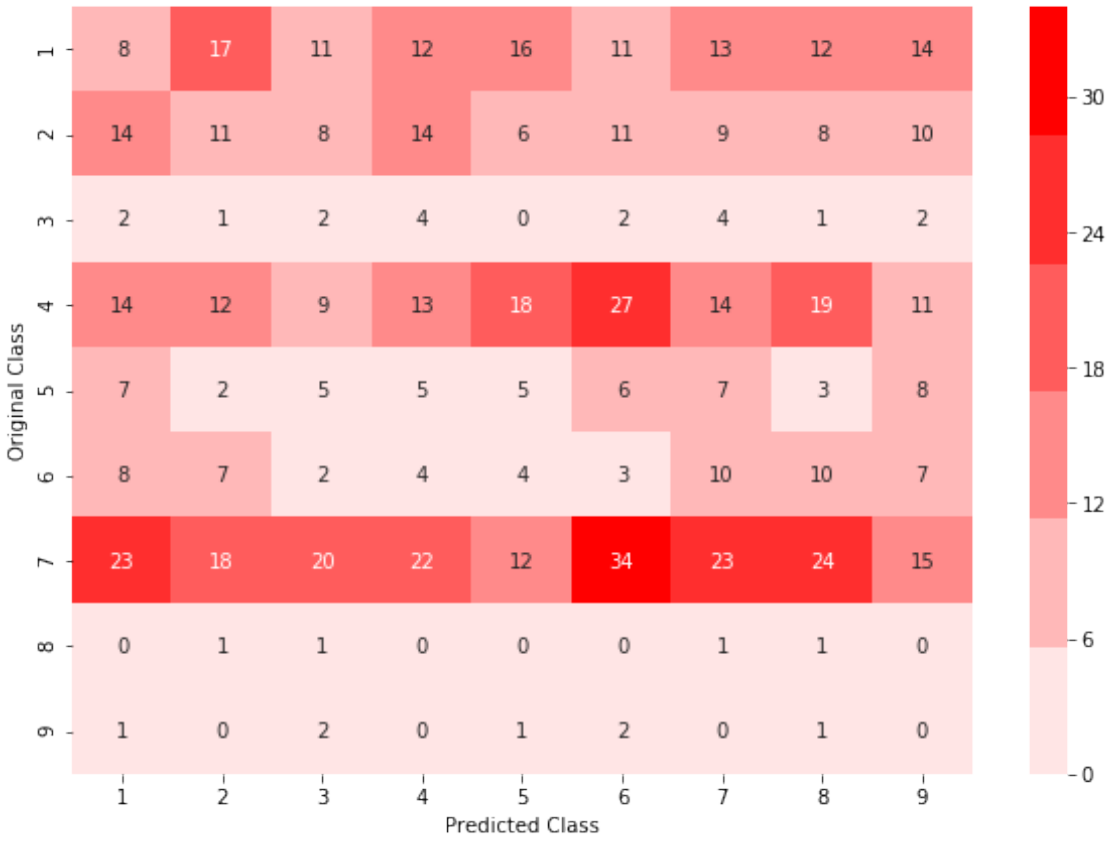

```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218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

```
In [16]: # From ABOVE HISTOGRAM-:
         # So the data is imbalanced and hence we are dealing with a MultiClass Classification
         # But the mostly similar distribution of Y_I's shows the Train and Test data have sim
         # Since for Machine Learning Model to be work very well, the distribution have to ver

In [17]: #MAKING A RANDOM MODEL

In [18]: # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
             # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

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

```
    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]

    cmap=sns.light_palette("red")
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(10,7))
    sns.heatmap(C, annot=True, cmap=cmap, xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(10,7))
    sns.heatmap(B, annot=True, cmap=cmap , fmt=".3f", xticklabels=labels, yticklabels=
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(10,7))
    sns.heatmap(A, annot=True, cmap=cmap , fmt=".3f", xticklabels=labels, yticklabels=
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

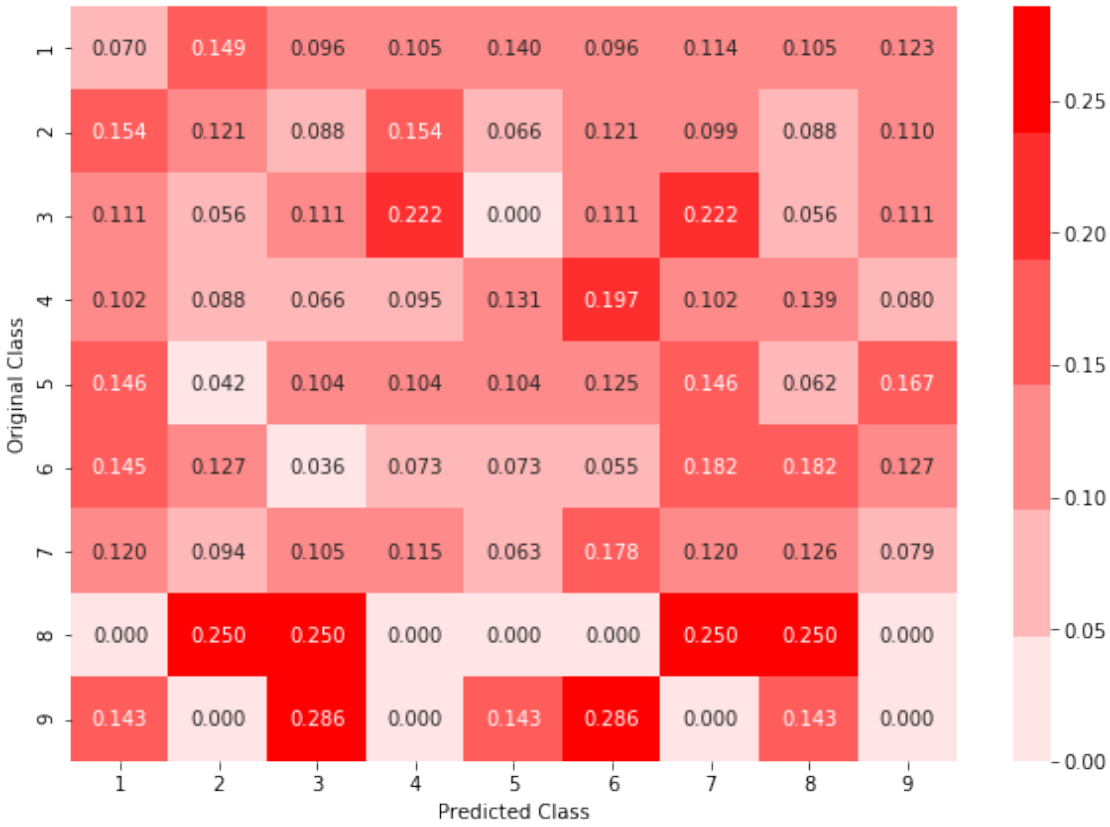
In [19]:
```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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_predicte

test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps

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 2.4106767305800423
Log loss on Test Data using Random Model 2.5713944165110174
-------------------- Confusion matrix --------------------
```



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

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



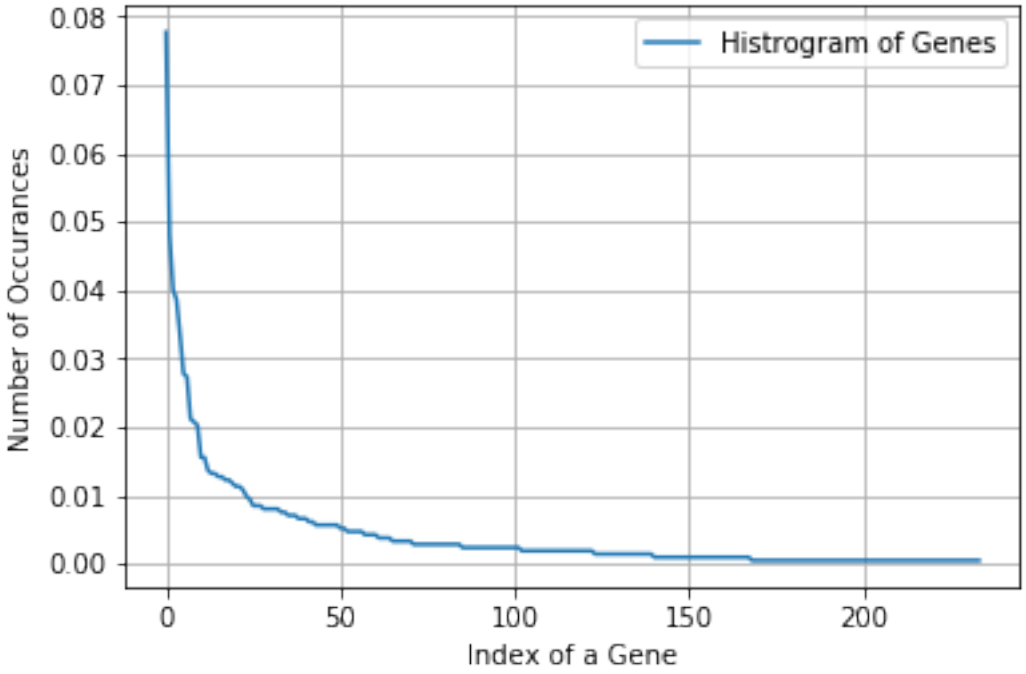
In [20]:
```
# code for response coding with laplace smoothing

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each ge
    gv_dict = dict()

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
```

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

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

In [21]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes in train data :', unique_genes.shape[0])
print(unique_genes.head(10))
```

```
Number of Unique Genes in train data : 234
BRCA1     165
TP53      102
BRCA2      85
EGFR       82
PTEN       71
KIT        59
BRAF       58
ERBB2      45
ALK        44
PDGFRA     43
Name: Gene, dtype: int64
```

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



In [23]:
```
#From HistoGram-:
    # skewed distribution
    # we found that there are lots of genes which occurs very few almost less than 10
```

In [24]:
```
# methods to encode a categorical random variable, Since Gene is a categorical random
    #1. Response Coding feature(here we use Probabilities) / or also called as Mean V
    #2. One-hot Encoding feature
```

In [25]:
```
#response-coding of the Gene feature
alpha = 1                    # alpha is used for laplace/Additive smoothing
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
print("The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

```
The shape of gene feature: (2124, 9)
```

In [26]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [27]:
```
train_df['Gene'].head()
```

```
Out[27]: 2309     JAK1
         1160     FAT1
         2658    BRCA1
         1857     MTOR
         3129     KRAS
         Name: Gene, dtype: object

In [28]: gene_vectorizer.get_feature_names()

Out[28]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1b',
          'arid2',
          'arid5b',
          'asxl2',
          'atm',
          'atr',
          'atrx',
          'aurka',
          'aurkb',
          'axl',
          'b2m',
          'bap1',
          'bcl10',
          'bcl2',
          'bcl2l11',
          'bcor',
          'braf',
          'brca1',
          'brca2',
          'brd4',
          'brip1',
          'btk',
          'card11',
          'carm1',
          'casp8',
          'cbl',
          'ccnd1',
          'ccnd3',
          'ccne1',
```

```
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgf4',
'fgfr1',
'fgfr2',
```

```
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'fubp1',
'gata3',
'gna11',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'men1',
```

```
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rac1',
'rad21',
'rad50',
```

```
'rad51b',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stat3',
'stk11',
'tcf3',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vhl',
'whsc1',
'whsc1l1',
```

```
          'xpo1',
          'xrcc2',
          'yap1']
```

In [29]:
```
print("The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

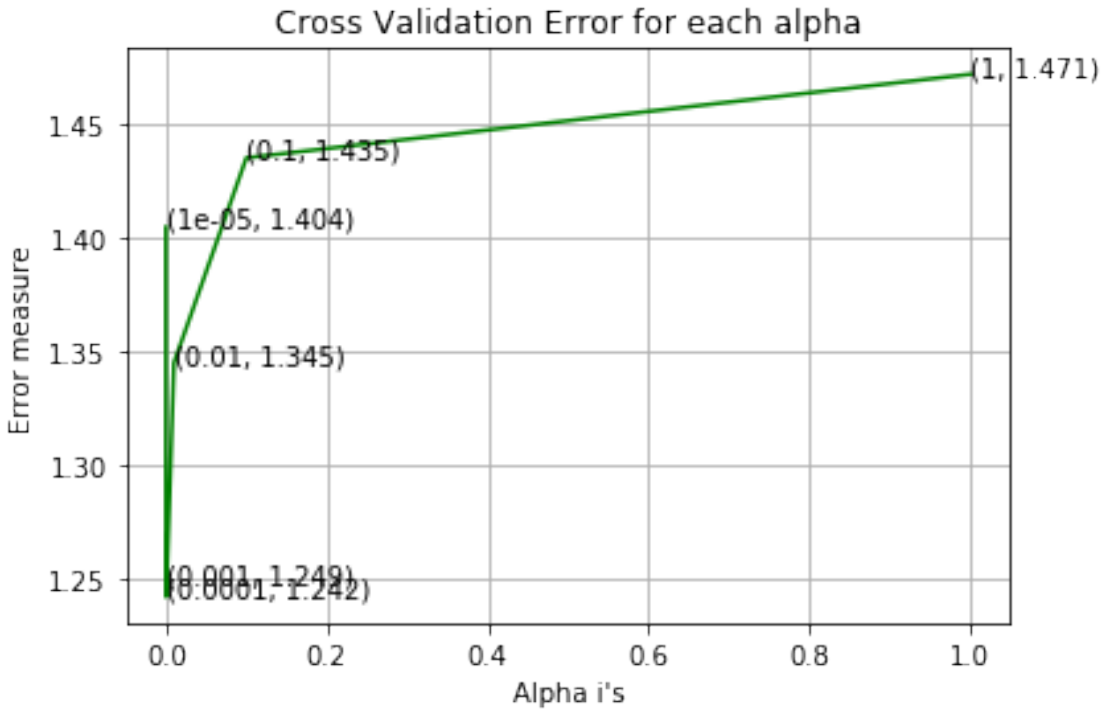
```
The shape of gene feature: (2124, 234)
```

In [30]:
```
# NOW PREDICTING Y_I ONLY WITH USING GENE FEATURE
```

In [31]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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


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

        predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
        predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.4043732683186334
For values of alpha =  0.0001 The log loss is: 1.2423202738418542
```

```
For values of alpha =  0.001 The log loss is: 1.2485713581246216
For values of alpha =  0.01 The log loss is: 1.3448753337649717
For values of alpha =  0.1 The log loss is: 1.4347402093260375
For values of alpha =  1 The log loss is: 1.4713922882612718
```



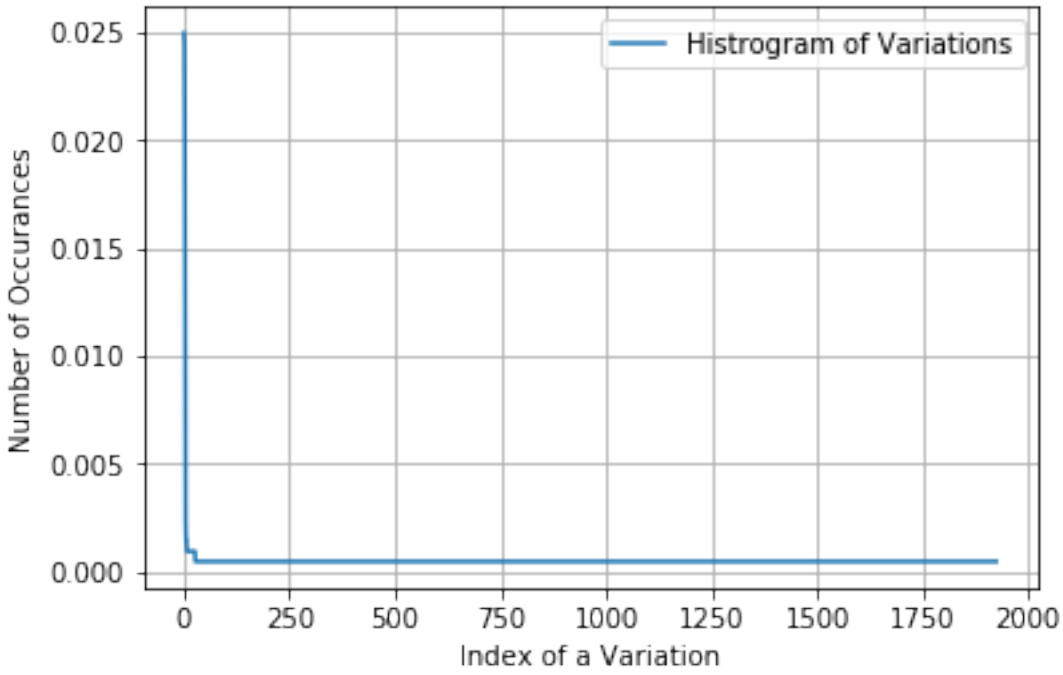

```
For values of best alpha =  0.0001 The train log loss is: 1.0182304092728538
For values of best alpha =  0.0001 The cross validation log loss is: 1.2423202738418542
For values of best alpha =  0.0001 The test log loss is: 1.194659084662193
```

```
In [32]: # "VARIATION" FEATURE ANALYSIS

In [33]: unique_variations = train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         print(unique_variations.head(10))
```

```
Number of Unique Variations : 1925
Deletion                 53
Truncating_Mutations     52
Amplification            45
Fusions                  25
Overexpression            5
G12V                      3
```

```
Q61H                    3
G12D                    2
TMPRSS2-ETV1_Fusion     2
P34R                    2
Name: Variation, dtype: int64
```

In [34]:
```
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 [35]:
```
#response-coding of the Gene feature
alpha = 1                # alpha is used for laplace smoothing
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_
```

In [36]:
```
print("The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

```
The shape of Variation feature: (2124, 9)
```

In [37]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [38]:
```
print("The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

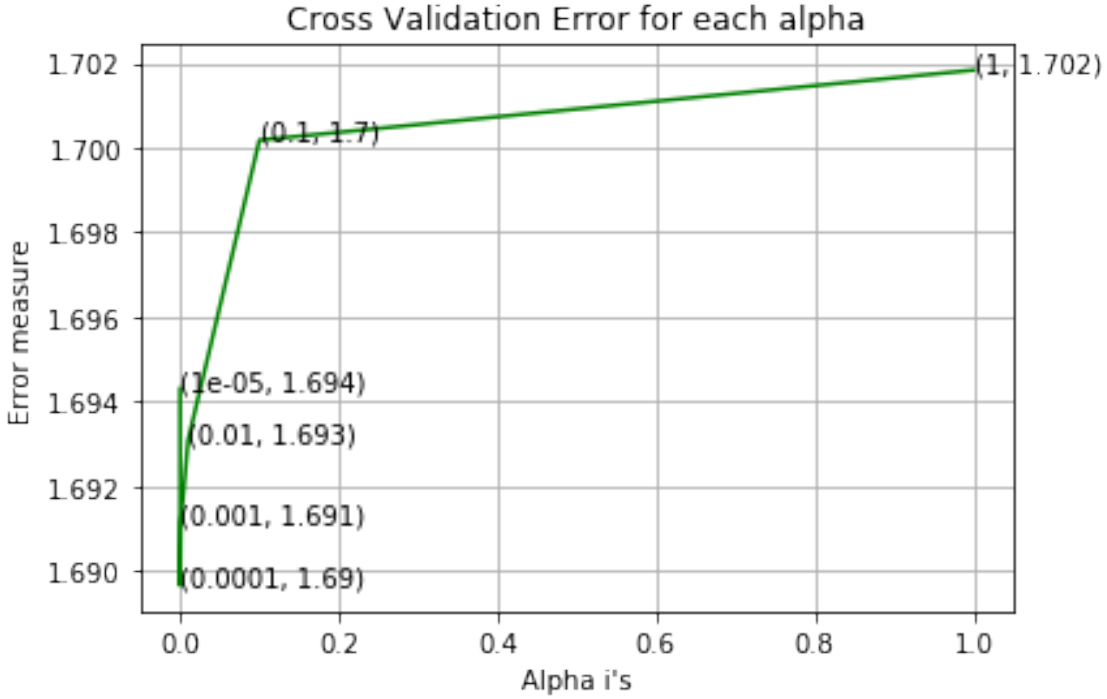
```
The shape of Variation feature: (2124, 1959)
```

In [39]:
```
# NOW PREDICTING Y_I'S USING ONLY VARIATION FEATURE
```

In [40]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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

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


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

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

```
        predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.6942998887635736
For values of alpha =  0.0001 The log loss is: 1.6896393979299233
For values of alpha =  0.001 The log loss is: 1.6911415115617012
For values of alpha =  0.01 The log loss is: 1.6930707974746042
For values of alpha =  0.1 The log loss is: 1.700180688801278
For values of alpha =  1 The log loss is: 1.701837288637838
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7296921640301866
For values of best alpha =  0.0001 The cross validation log loss is: 1.6896393979299233
For values of best alpha =  0.0001 The test log loss is: 1.7112674655469617
```

```
In [41]: # Univariate Analysis on Text Feature
```

```
In [42]: # cls_text is a data frame
         # for every row in data fram consider the 'TEXT'
         # split the words by space
         # make a dict with those words
         # increment its count whenever we see that word

         def extract_dictionary_paddle(cls_text):
```

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

In [43]:
```
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 )/(total_dict.get(w
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TE
            row_index += 1
    return text_feature_responseCoding
```

In [44]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*nu
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 53184
```

In [45]:
```
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 [46]: #response coding of text features
        train_text_feature_responseCoding  = get_text_responsecoding(train_df)
        test_text_feature_responseCoding  = get_text_responsecoding(test_df)
        cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

```
In [47]: # we convert each row values such that they sum to 1
        train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_fe
        test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
        cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res
```

```
In [48]: # don't forget to normalize every feature
        train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```
In [49]: sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
        sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

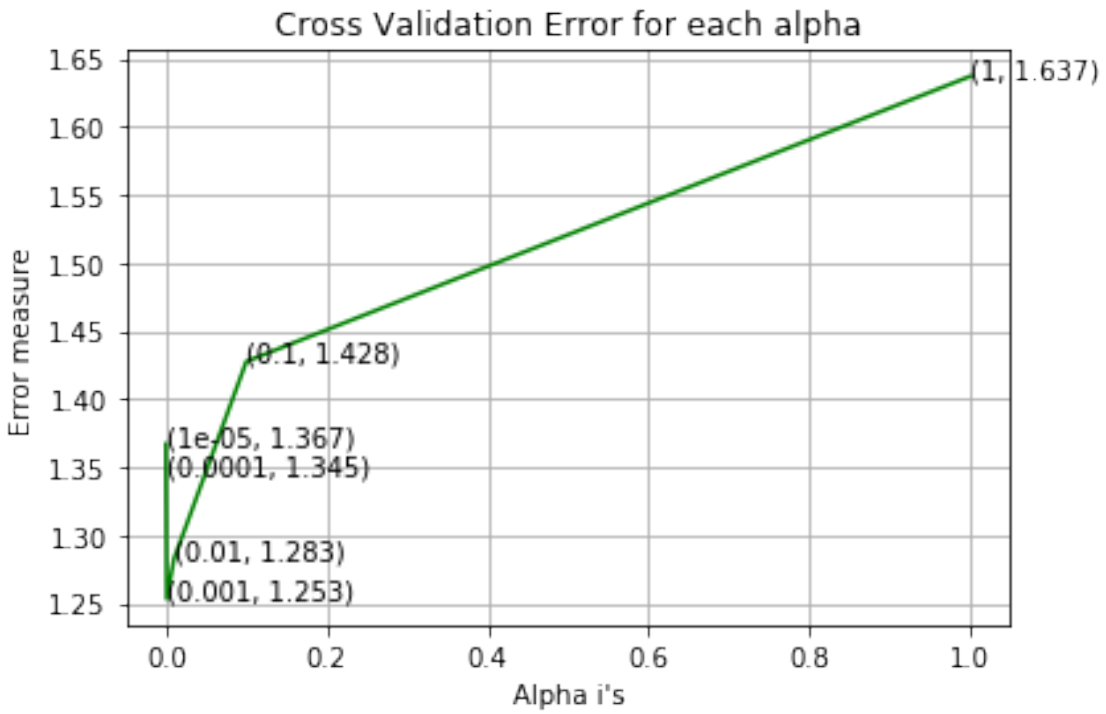
```
In [50]: # Number of words for a given frequency.
        print(Counter(sorted_text_occur))
```

```
Counter({3: 5308, 4: 3918, 6: 2898, 5: 2438, 10: 2119, 7: 1946, 8: 1944, 9: 1471, 12: 1419, 11
```

```
In [51]: # Train a Logistic regression+Calibration model using text features whicha re on-hot
        alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

    predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
    predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.3670882598600838
For values of alpha =  0.0001 The log loss is: 1.3447043598168051
For values of alpha =  0.001 The log loss is: 1.2533243729890935
For values of alpha =  0.01 The log loss is: 1.2825266893559684
For values of alpha =  0.1 The log loss is: 1.4276730577701786
For values of alpha =  1 The log loss is: 1.637137951015819
```




```
For values of best alpha =  0.001 The train log loss is: 0.7553225235425016
For values of best alpha =  0.001 The cross validation log loss is: 1.2533243729890935
For values of best alpha =  0.001 The test log loss is: 1.1965908934309086
```

```
In [52]: # MACHINE LEARNING MODELS
```

```
In [53]: #Data preparation for ML models.
         #Misc. functionns for ML models

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

             # for calculating log_loss we willl provide the array of probabilities belongs to
             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
             plot_confusion_matrix(test_y, pred_y)
```

```
In [54]: 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 [55]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

In [56]:
```
# STACKING THE THREE TYPES OF FEATURES
```

In [57]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
```

```
        # a = [[1, 2],
        #      [3, 4]]
        # b = [[4, 5],
        #      [6, 7]]
        # hstack(a, b) = [[1, 2, 4, 5],
        #                 [ 3, 4, 6, 7]]

        train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_
        test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
        cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_c

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

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

        cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).t
        cv_y = np.array(list(cv_df['Class']))


        train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_var
        test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variat
        cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fe

        train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_
        test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_res
        cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseC
```

In [58]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotC
print("(number of data points * number of features) in test data = ", test_x_onehotCod
print("(number of data points * number of features) in cross validation data =", cv_x_
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55377)
(number of data points * number of features) in test data =  (665, 55377)
(number of data points * number of features) in cross validation data = (532, 55377)
```

In [59]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respons
print("(number of data points * number of features) in test data = ", test_x_responseC
print("(number of data points * number of features) in cross validation data =", cv_x_
```

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

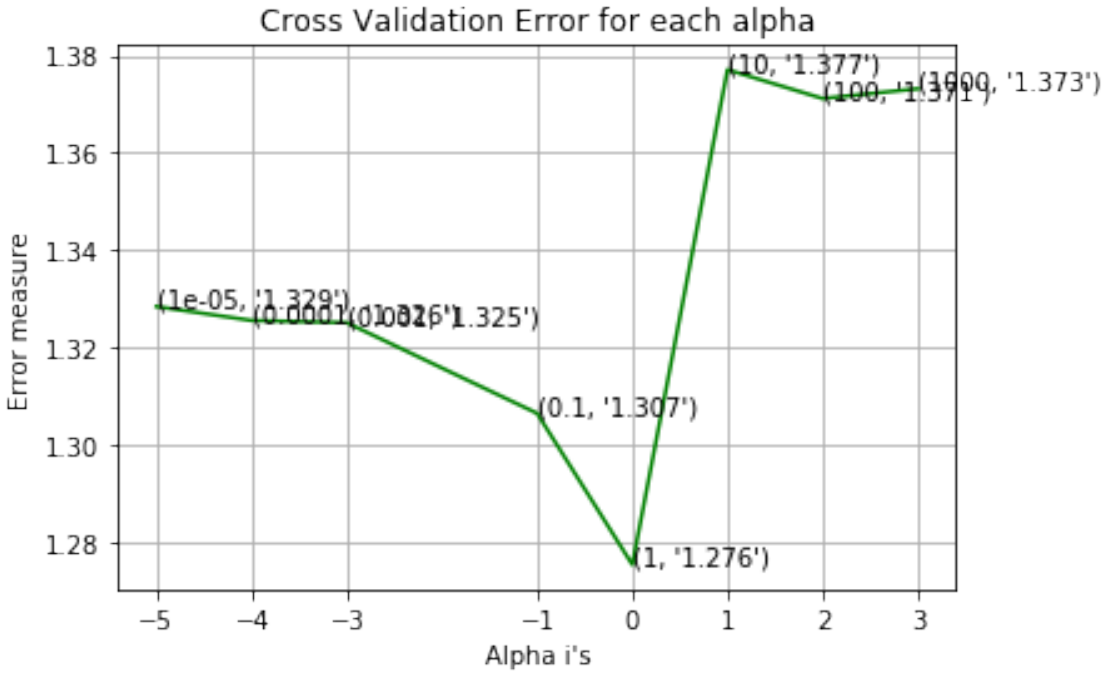
```
(number of data points * number of features) in cross validation data = (532, 27)
```

```
In [60]: # BASE LINE MODEL
         # NAIVE BAYES
         # WITH HYPER PARAMETER TUNING
```

```
In [61]: 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=
             # to avoid rounding error while multiplying probabilites we use log-probability e
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 1.3285127547197002
```

```
for alpha = 0.0001
Log Loss : 1.3256487242858312
for alpha = 0.001
Log Loss : 1.3251957988635086
for alpha = 0.1
Log Loss : 1.306613935171177
for alpha = 1
Log Loss : 1.2755928877368068
for alpha = 10
Log Loss : 1.3770571063840178
for alpha = 100
Log Loss : 1.371152138662326
for alpha = 1000
Log Loss : 1.373156290374542
```

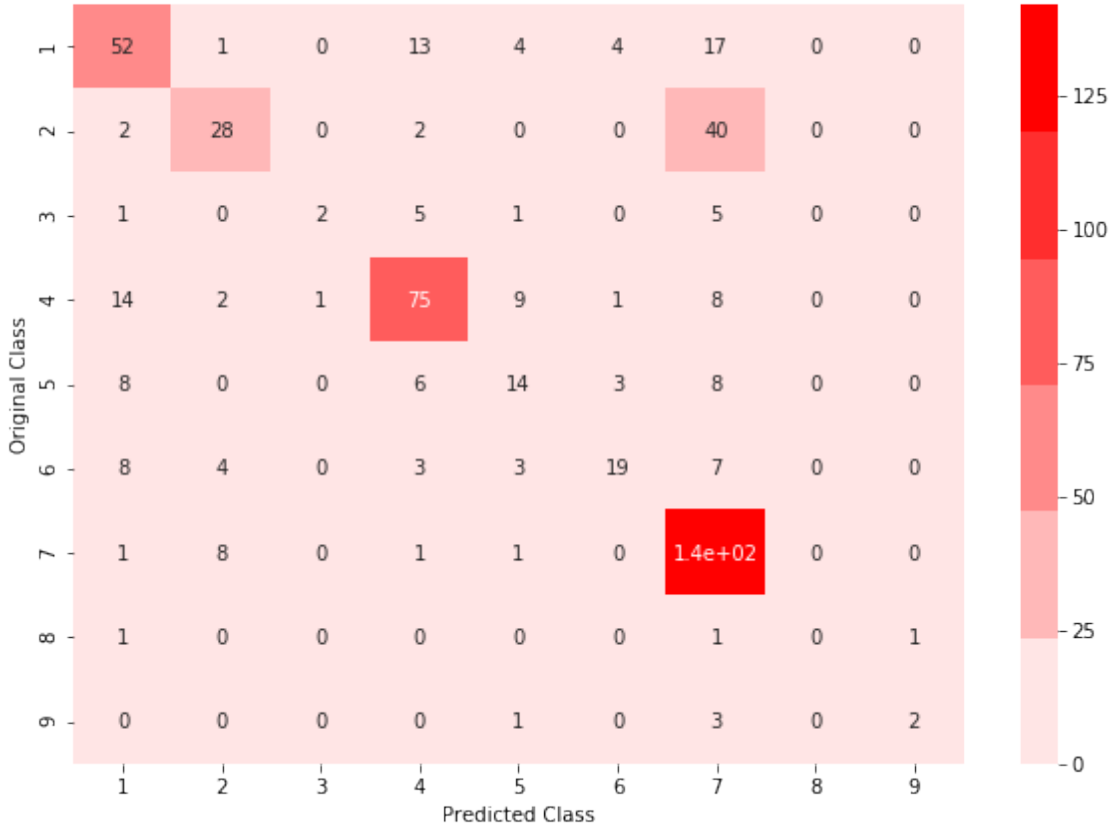



```
For values of best alpha =  1 The train log loss is: 0.9186142348363319
For values of best alpha =  1 The cross validation log loss is: 1.2755928877368068
For values of best alpha =  1 The test log loss is: 1.264659752826925
```

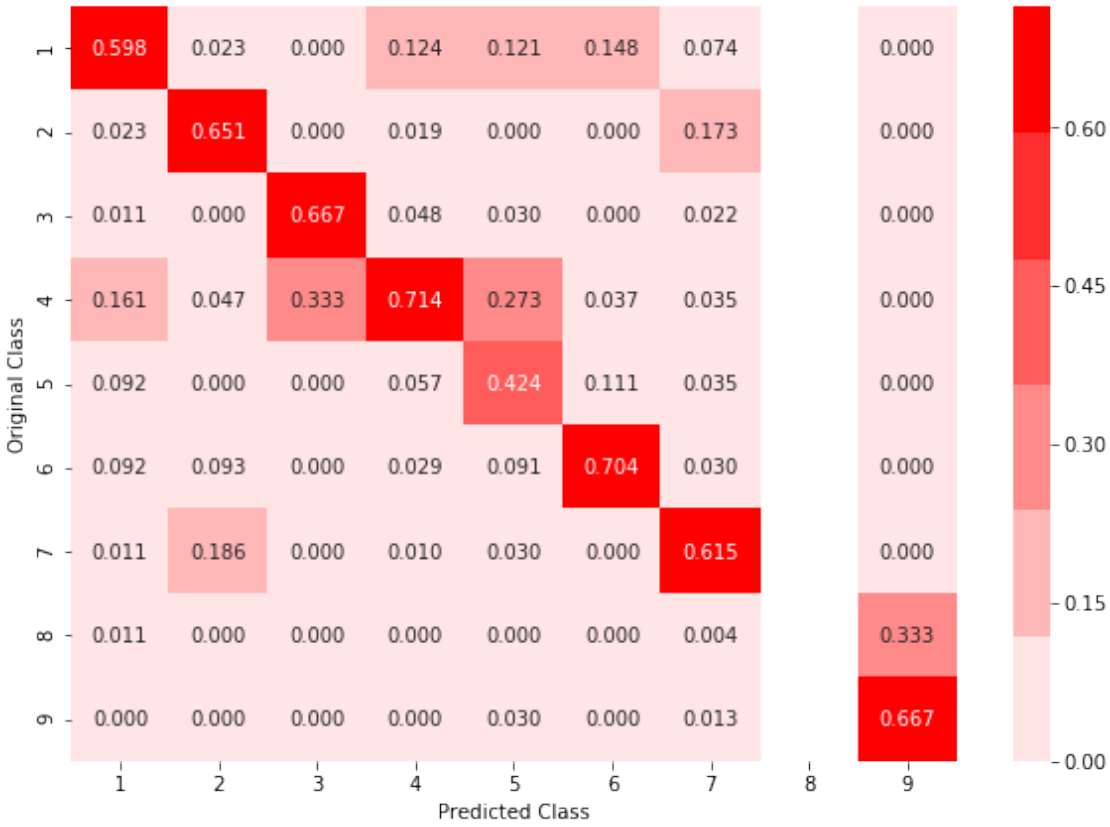
```
In [62]: # TESTING THE MODEL WITH BEST HYPER PARAMTERS

In [63]: clf = MultinomialNB(alpha=alpha[best_alpha])
         clf.fit(train_x_onehotCoding, train_y)
```

```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estim
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_oneh
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

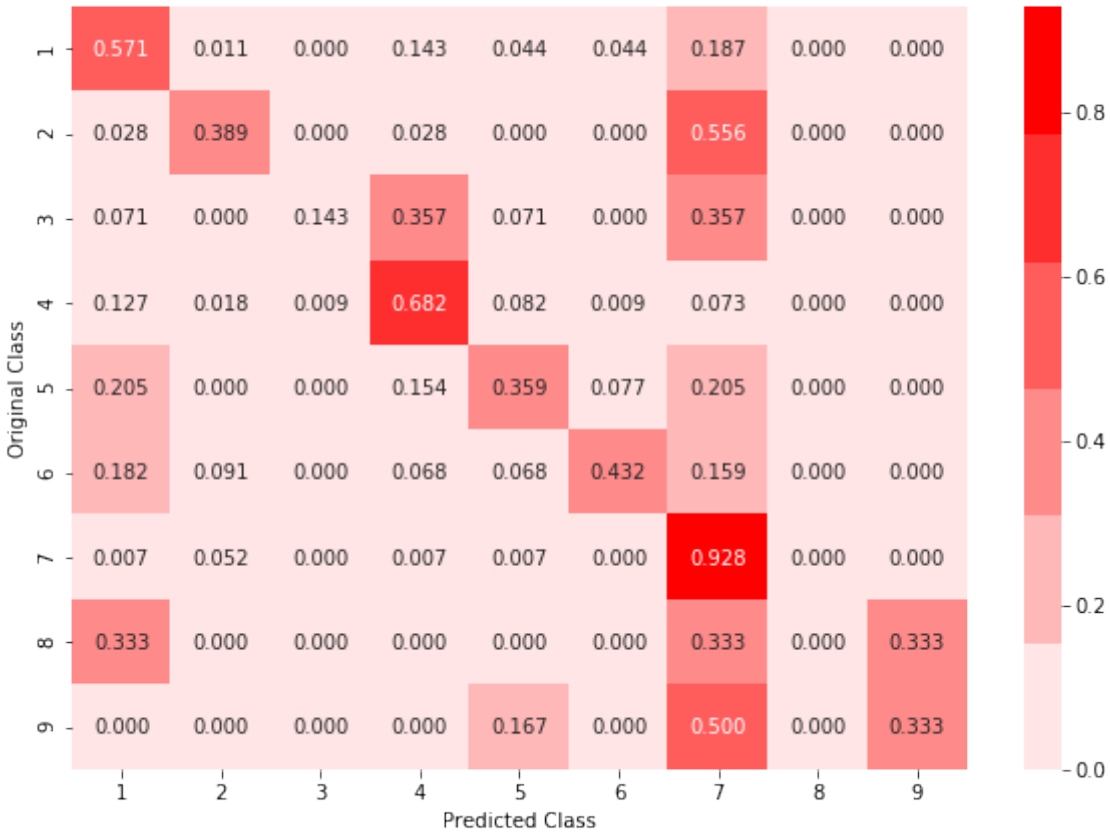
```
Log Loss : 1.2755928877368068
Number of missclassified point : 0.37218045112781956
-------------------- Confusion matrix --------------------
```




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



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



In [64]: *# FEATURE IMPORTANCE, CORRECTLY CLASSIFIED POINT*

In [65]:
```
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_onehotC
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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0817 0.0889 0.023  0.0911 0.0472 0.0344 0.6243 0.0046 0.004
Actual Class : 7
--------------------------------------------------
16 Text feature [presence] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [activating] present in test data point [True]
19 Text feature [well] present in test data point [True]
20 Text feature [cells] present in test data point [True]
```

```
22 Text feature [downstream] present in test data point [True]
23 Text feature [shown] present in test data point [True]
25 Text feature [previously] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
27 Text feature [recently] present in test data point [True]
28 Text feature [cell] present in test data point [True]
29 Text feature [expressing] present in test data point [True]
30 Text feature [addition] present in test data point [True]
32 Text feature [also] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [showed] present in test data point [True]
35 Text feature [independent] present in test data point [True]
36 Text feature [compared] present in test data point [True]
37 Text feature [10] present in test data point [True]
38 Text feature [inhibitor] present in test data point [True]
39 Text feature [mutations] present in test data point [True]
40 Text feature [higher] present in test data point [True]
41 Text feature [described] present in test data point [True]
42 Text feature [obtained] present in test data point [True]
43 Text feature [similar] present in test data point [True]
44 Text feature [potential] present in test data point [True]
45 Text feature [observed] present in test data point [True]
46 Text feature [suggest] present in test data point [True]
47 Text feature [activation] present in test data point [True]
48 Text feature [growth] present in test data point [True]
50 Text feature [respectively] present in test data point [True]
51 Text feature [total] present in test data point [True]
53 Text feature [interestingly] present in test data point [True]
54 Text feature [12] present in test data point [True]
55 Text feature [using] present in test data point [True]
59 Text feature [report] present in test data point [True]
60 Text feature [inhibition] present in test data point [True]
62 Text feature [due] present in test data point [True]
63 Text feature [3b] present in test data point [True]
64 Text feature [identified] present in test data point [True]
65 Text feature [confirmed] present in test data point [True]
68 Text feature [although] present in test data point [True]
69 Text feature [reported] present in test data point [True]
70 Text feature [fig] present in test data point [True]
71 Text feature [1a] present in test data point [True]
72 Text feature [followed] present in test data point [True]
73 Text feature [proliferation] present in test data point [True]
74 Text feature [mutation] present in test data point [True]
76 Text feature [whereas] present in test data point [True]
77 Text feature [two] present in test data point [True]
78 Text feature [increased] present in test data point [True]
81 Text feature [phosphorylation] present in test data point [True]
82 Text feature [without] present in test data point [True]
```

```
84 Text feature [15] present in test data point [True]
85 Text feature [consistent] present in test data point [True]
86 Text feature [leading] present in test data point [True]
87 Text feature [demonstrated] present in test data point [True]
88 Text feature [3a] present in test data point [True]
91 Text feature [performed] present in test data point [True]
96 Text feature [either] present in test data point [True]
97 Text feature [mechanisms] present in test data point [True]
98 Text feature [different] present in test data point [True]
99 Text feature [recent] present in test data point [True]
Out of the top  100  features  63 are present in query point
```

```
In [66]: # FEATURE IMPORTANCE, INCORRECTLY CLASSIFIED POINT
```

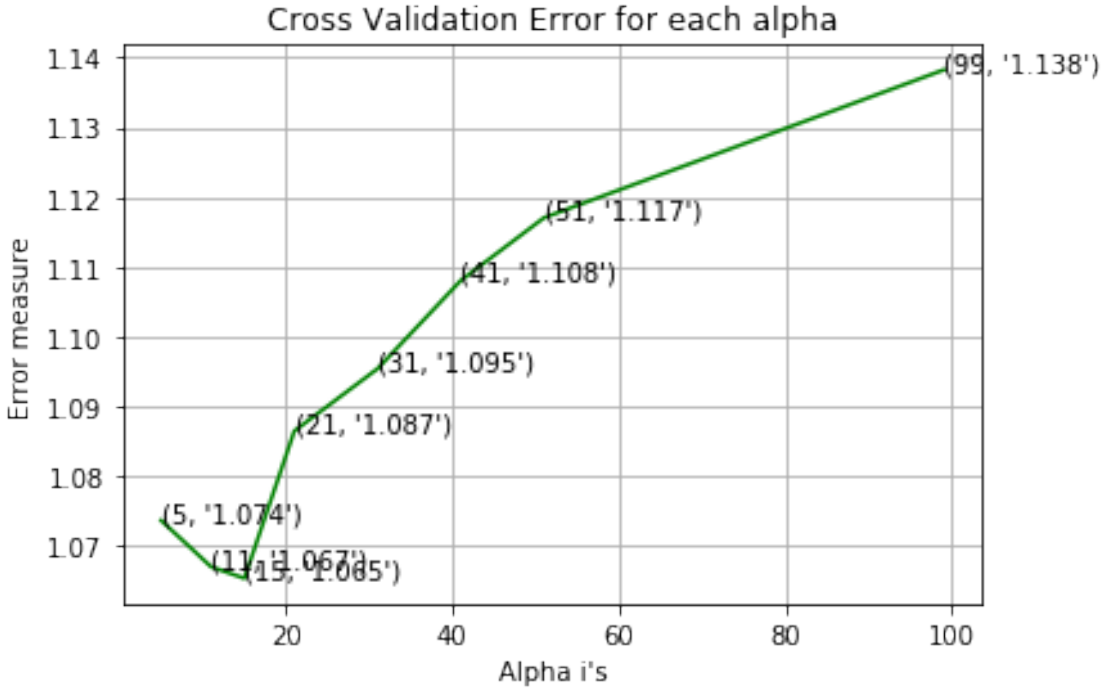
```
In [67]: test_point_index = 100
         no_feature = 100
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
         print("Actual 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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0844 0.0919 0.0236 0.1062 0.0482 0.0353 0.6006 0.0047 0.005
Actual Class : 1
--------------------------------------------------
16 Text feature [presence] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [activating] present in test data point [True]
19 Text feature [well] present in test data point [True]
20 Text feature [cells] present in test data point [True]
22 Text feature [downstream] present in test data point [True]
23 Text feature [shown] present in test data point [True]
24 Text feature [found] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
27 Text feature [recently] present in test data point [True]
28 Text feature [cell] present in test data point [True]
29 Text feature [expressing] present in test data point [True]
30 Text feature [addition] present in test data point [True]
31 Text feature [may] present in test data point [True]
32 Text feature [also] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [showed] present in test data point [True]
35 Text feature [independent] present in test data point [True]
37 Text feature [10] present in test data point [True]
```

```
38 Text feature [inhibitor] present in test data point [True]
39 Text feature [mutations] present in test data point [True]
41 Text feature [described] present in test data point [True]
42 Text feature [obtained] present in test data point [True]
43 Text feature [similar] present in test data point [True]
44 Text feature [potential] present in test data point [True]
45 Text feature [observed] present in test data point [True]
46 Text feature [suggest] present in test data point [True]
47 Text feature [activation] present in test data point [True]
48 Text feature [growth] present in test data point [True]
49 Text feature [factor] present in test data point [True]
50 Text feature [respectively] present in test data point [True]
51 Text feature [total] present in test data point [True]
52 Text feature [studies] present in test data point [True]
54 Text feature [12] present in test data point [True]
55 Text feature [using] present in test data point [True]
59 Text feature [report] present in test data point [True]
61 Text feature [new] present in test data point [True]
62 Text feature [due] present in test data point [True]
63 Text feature [3b] present in test data point [True]
64 Text feature [identified] present in test data point [True]
65 Text feature [confirmed] present in test data point [True]
68 Text feature [although] present in test data point [True]
69 Text feature [reported] present in test data point [True]
70 Text feature [fig] present in test data point [True]
73 Text feature [proliferation] present in test data point [True]
74 Text feature [mutation] present in test data point [True]
75 Text feature [various] present in test data point [True]
76 Text feature [whereas] present in test data point [True]
77 Text feature [two] present in test data point [True]
78 Text feature [increased] present in test data point [True]
79 Text feature [suggests] present in test data point [True]
80 Text feature [including] present in test data point [True]
82 Text feature [without] present in test data point [True]
84 Text feature [15] present in test data point [True]
85 Text feature [consistent] present in test data point [True]
86 Text feature [leading] present in test data point [True]
88 Text feature [3a] present in test data point [True]
89 Text feature [enhanced] present in test data point [True]
90 Text feature [three] present in test data point [True]
91 Text feature [performed] present in test data point [True]
94 Text feature [occur] present in test data point [True]
95 Text feature [thus] present in test data point [True]
96 Text feature [either] present in test data point [True]
97 Text feature [mechanisms] present in test data point [True]
98 Text feature [different] present in test data point [True]
99 Text feature [recent] present in test data point [True]
Out of the top  100  features  66 are present in query point
```

```
In [68]: # K Nearest Neighbour Classification Algo-:

In [69]: 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=
             # to avoid rounding error while multiplying probabilites we use log-probability e
             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_
         predict_y = sig_clf.predict_proba(cv_x_responseCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
         predict_y = sig_clf.predict_proba(test_x_responseCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo

for alpha = 5
Log Loss : 1.0737136647792824
for alpha = 11
Log Loss : 1.067076582287097
for alpha = 15
Log Loss : 1.0654189940485501
for alpha = 21
```

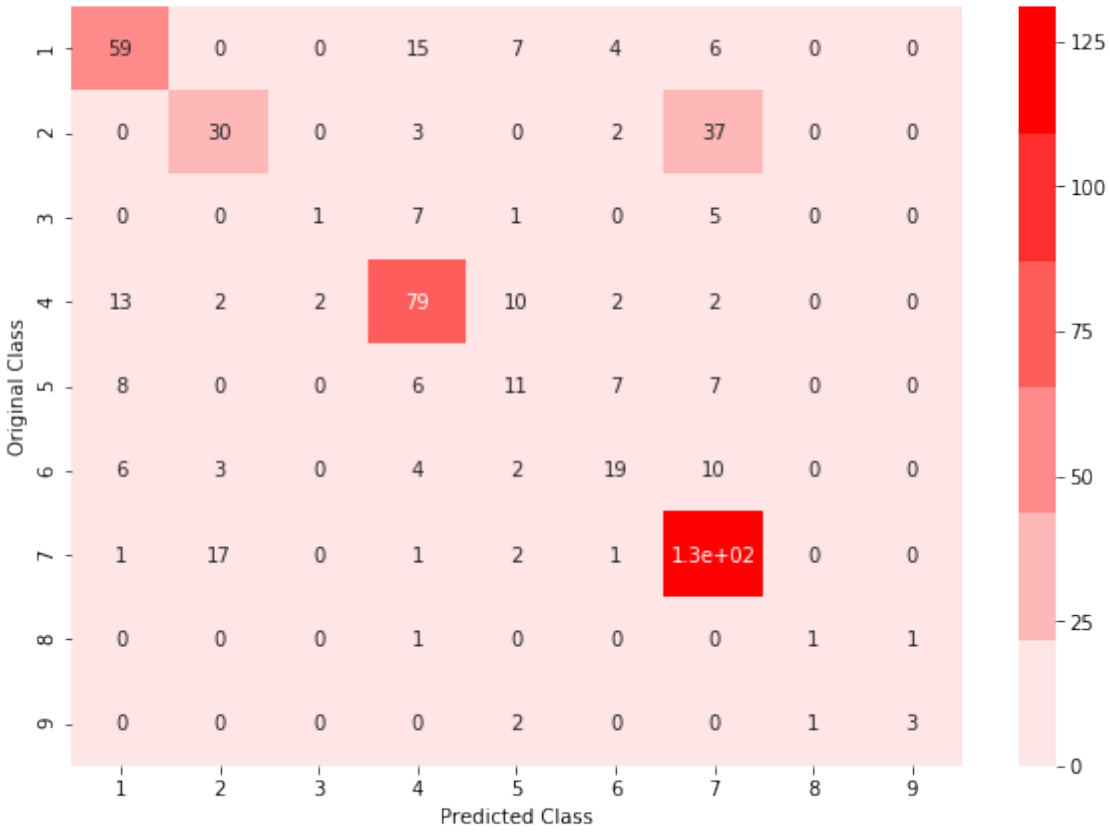
```
Log Loss : 1.0865148572390728
for alpha = 31
Log Loss : 1.0954418145301876
for alpha = 41
Log Loss : 1.1080711189014103
for alpha = 51
Log Loss : 1.1170659972633223
for alpha = 99
Log Loss : 1.1382386395945334
```



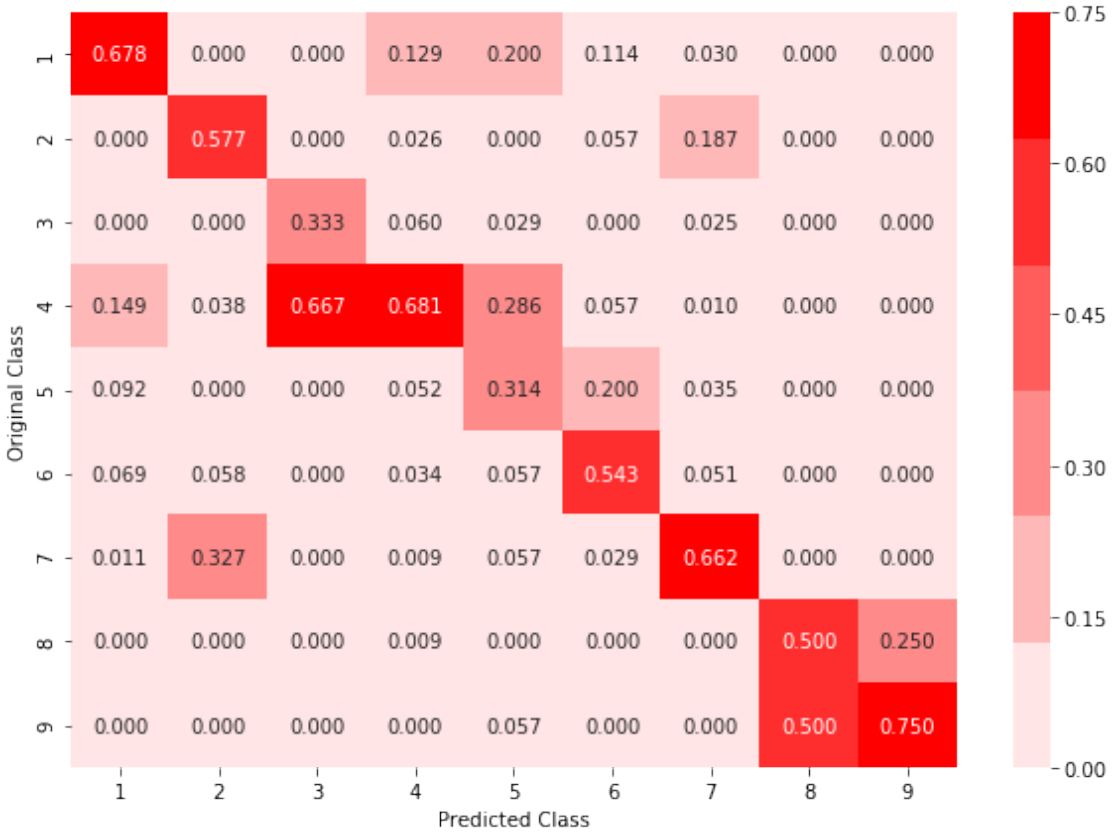

```
For values of best alpha =  15 The train log loss is: 0.6877666331292938
For values of best alpha =  15 The cross validation log loss is: 1.0654189940485501
For values of best alpha =  15 The test log loss is: 1.0204252261917468
```

```
In [70]: # Testing the model with best hyper paramters

In [71]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding
```

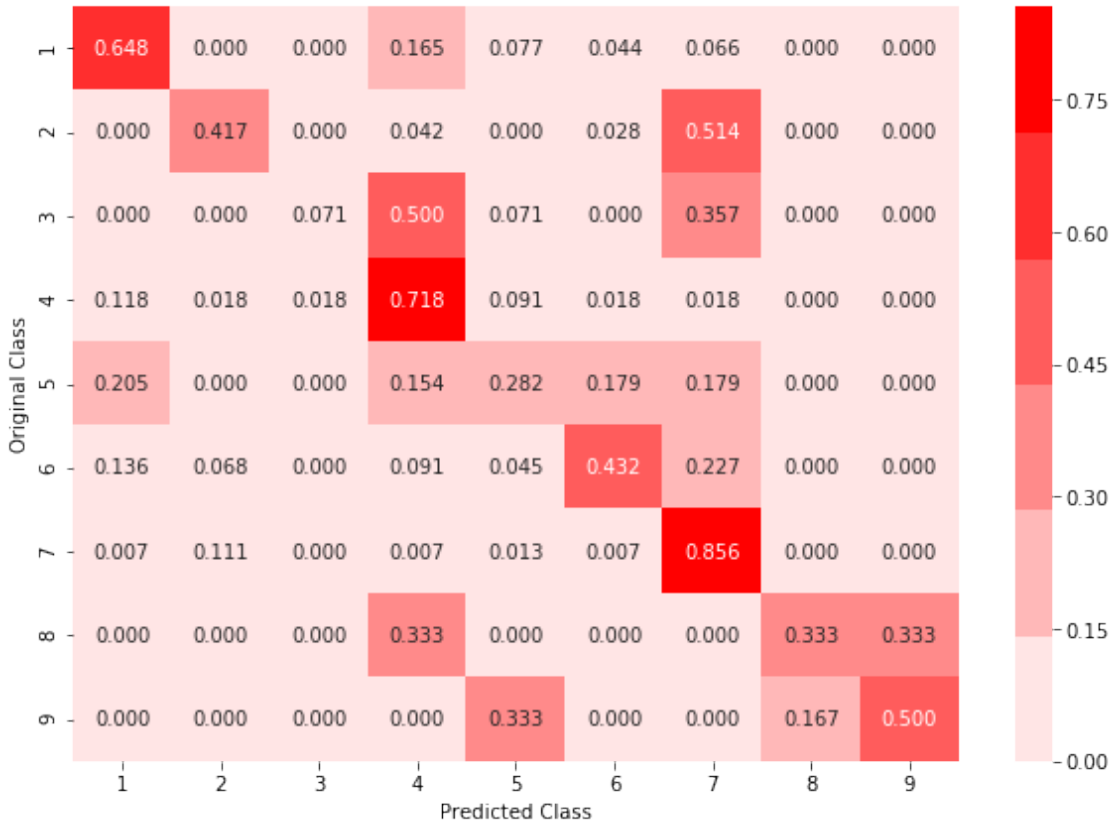
```
Log loss : 1.0654189940485501
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```



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



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



In [72]:
```
# Sample Query point-1
```

In [73]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 4
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [7 7 7 7 7 7 7 7 7 7 7 7 7 7
Fequency of nearest points : Counter({7: 15})
```

In [74]:
```
# Sample Query Point-2
```

```
In [75]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 100

         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
         print("Predicted Class :", predicted_cls[0])
         print("Actual Class :", test_y[test_point_index])
         neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alp
         print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the te
         print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

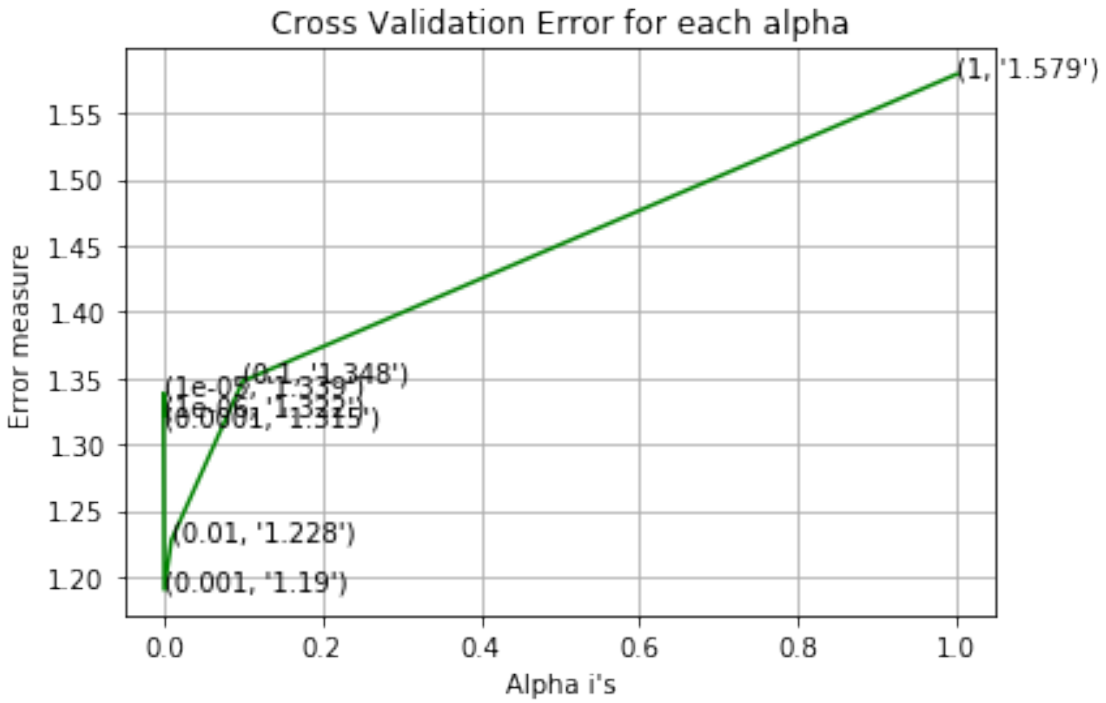
```
Predicted Class : 7
Actual Class : 1
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [4 1
Fequency of nearest points : Counter({7: 6, 2: 3, 6: 2, 1: 2, 4: 1, 5: 1})
```

```
In [76]: # LR Without Class balancing
```

```
In [77]: 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=1
             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_l
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(test_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 1e-06
Log Loss : 1.3220719048137128
for alpha = 1e-05
Log Loss : 1.338664324964034
for alpha = 0.0001
Log Loss : 1.3154892039432249
for alpha = 0.001
Log Loss : 1.1904380156631562
for alpha = 0.01
Log Loss : 1.2277984838568077
for alpha = 0.1
Log Loss : 1.3477302778739328
for alpha = 1
Log Loss : 1.5792659346665625
```

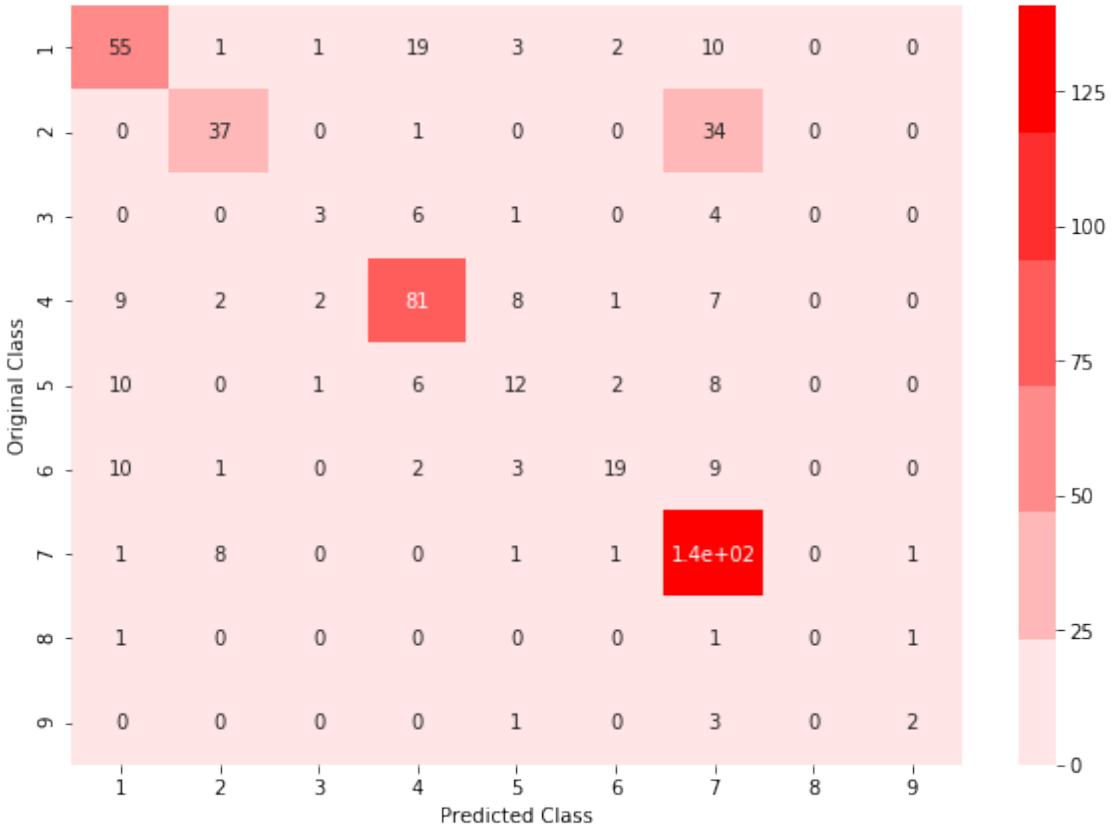


```
For values of best alpha =  0.001 The train log loss is: 0.6148359876675966
For values of best alpha =  0.001 The cross validation log loss is: 1.1904380156631562
For values of best alpha =  0.001 The test log loss is: 1.1449960781486952
```

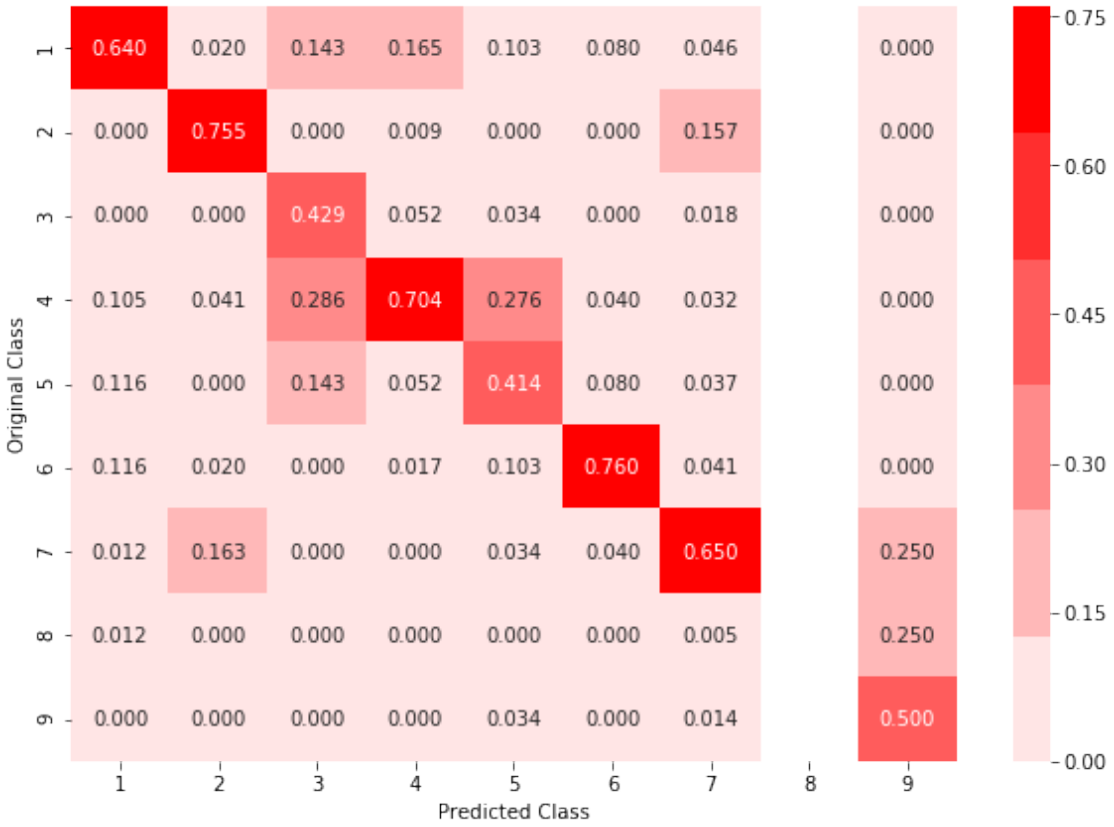
```
In [78]: # Testing model with best hyper parameters
```

```
In [79]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
         predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv
```

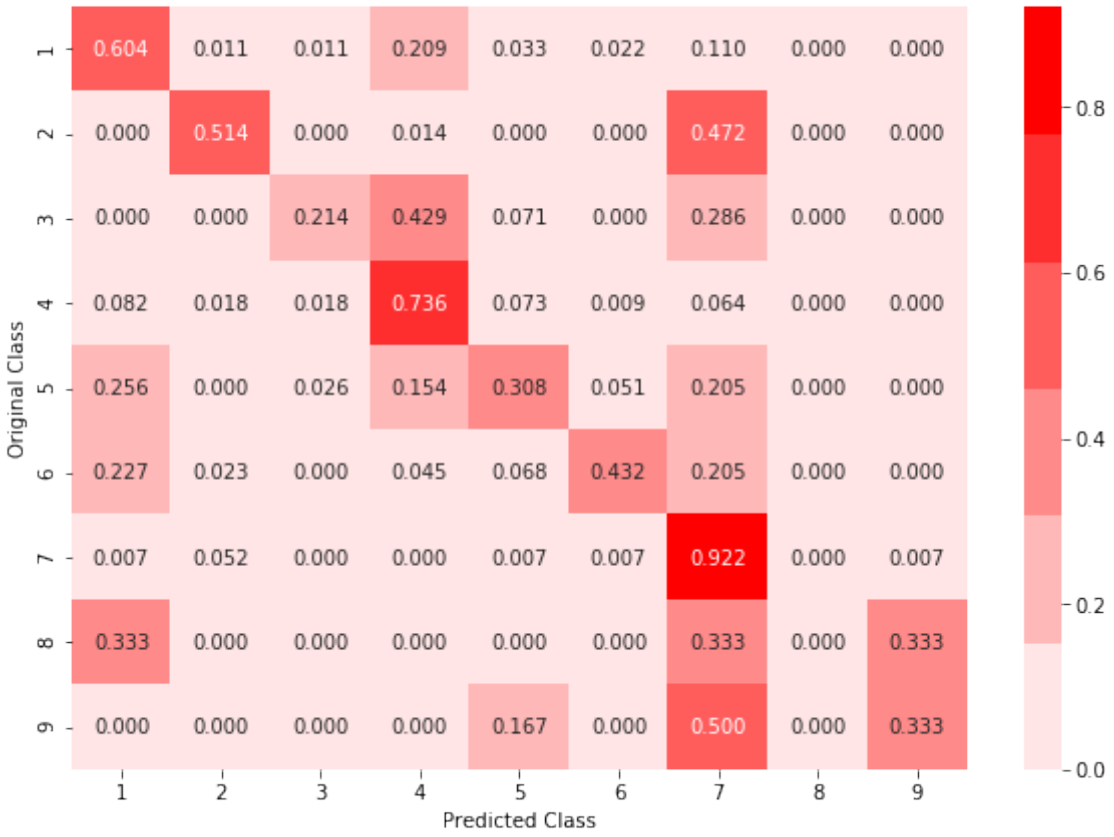
```
Log loss : 1.1904380156631562
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```




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



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



In [80]:
```
# Feature Importance, Correctly Classified point
```

In [81]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0817 0.0692 0.0098 0.1165 0.0378 0.0214 0.655  0.0042 0.0045
Actual Class : 7
--------------------------------------------------
110 Text feature [constitutive] present in test data point [True]
117 Text feature [constitutively] present in test data point [True]
145 Text feature [stat] present in test data point [True]
```

```
212 Text feature [jak] present in test data point [True]
256 Text feature [technology] present in test data point [True]
263 Text feature [interleukin] present in test data point [True]
264 Text feature [expressing] present in test data point [True]
265 Text feature [stat5] present in test data point [True]
320 Text feature [activating] present in test data point [True]
340 Text feature [ligand] present in test data point [True]
373 Text feature [activated] present in test data point [True]
462 Text feature [proliferation] present in test data point [True]
475 Text feature [serum] present in test data point [True]
Out of the top  500  features  13 are present in query point
```

```
In [82]: # Feature Importance, Inorrectly Classified point
```

```
In [83]: 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_onehotC
         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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1615 0.1214 0.0212 0.318  0.0535 0.0529 0.2528 0.0074 0.011
Actual Class : 1
--------------------------------------------------
311 Text feature [sending] present in test data point [True]
397 Text feature [ank] present in test data point [True]
403 Text feature [escherichia] present in test data point [True]
406 Text feature [nonsense] present in test data point [True]
476 Text feature [missense] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

```
In [84]: # Random Forest Classifier
         # Hyper paramter tuning (With One hot Encoding)
```

```
In [85]: 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, ra
                 clf.fit(train_x_onehotCoding, train_y)
                 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

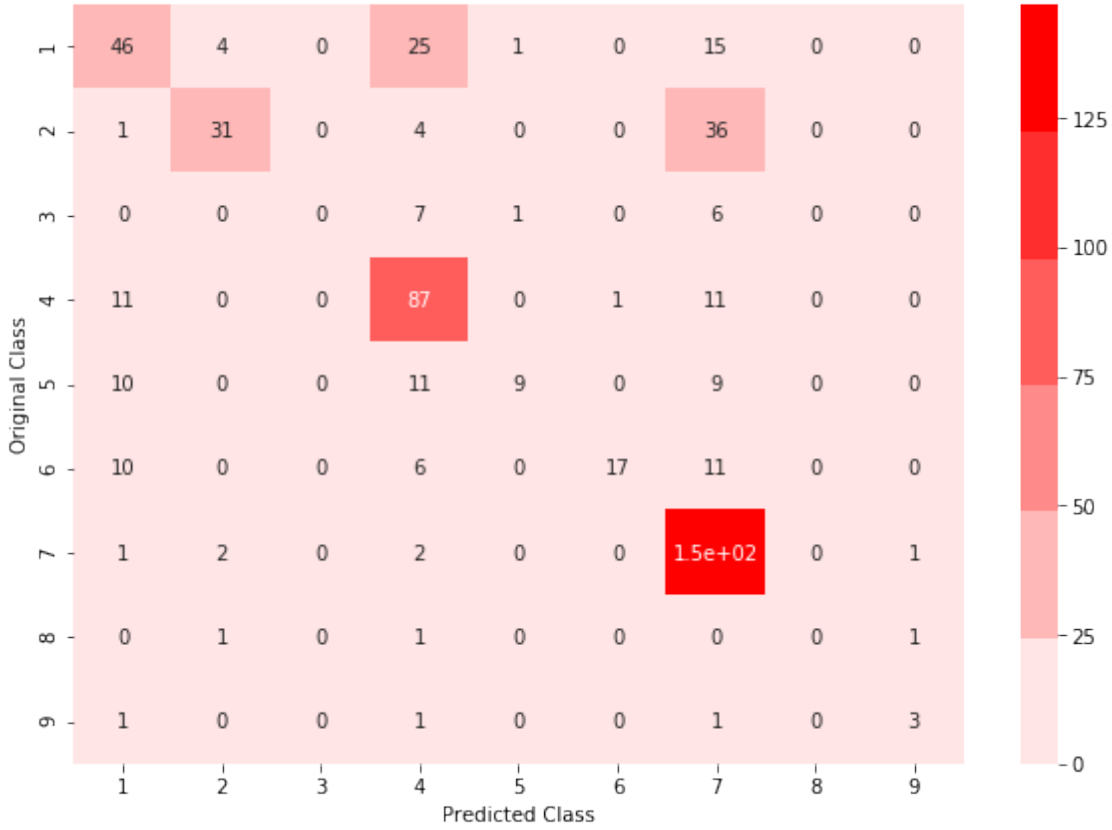
```
            sig_clf.fit(train_x_onehotCoding, train_y)
            sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
            cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, e
            print("Log Loss :",log_loss(cv_y, sig_clf_probs))


    best_alpha = np.argmin(cv_log_error_array)
    clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
    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
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validati
    predict_y = sig_clf.predict_proba(test_x_onehotCoding)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2391805007253798
for n_estimators = 100 and max depth =  10
Log Loss : 1.1776588467411708
for n_estimators = 200 and max depth =  5
Log Loss : 1.2261369510975186
for n_estimators = 200 and max depth =  10
Log Loss : 1.1702210062110516
for n_estimators = 500 and max depth =  5
Log Loss : 1.2105201065275508
for n_estimators = 500 and max depth =  10
Log Loss : 1.1616637439103308
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2077974303378027
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1569704867863988
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2063874742121623
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1550185361134009
For values of best estimator =  2000 The train log loss is: 0.7103790727983404
For values of best estimator =  2000 The cross validation log loss is: 1.1550185361134009
For values of best estimator =  2000 The test log loss is: 1.1440887143359608
```

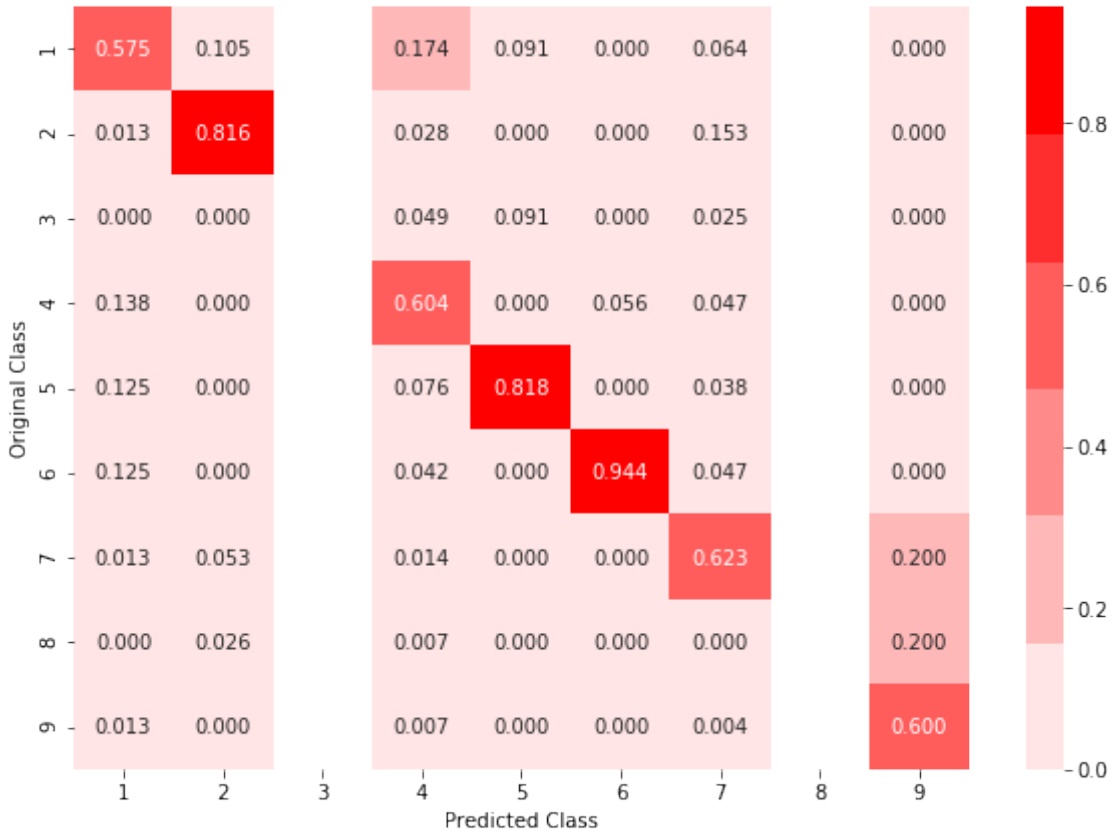
```
In [86]: # Testing model with best hyper parameters (One Hot Encoding)
```

```
In [87]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
         predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y
```

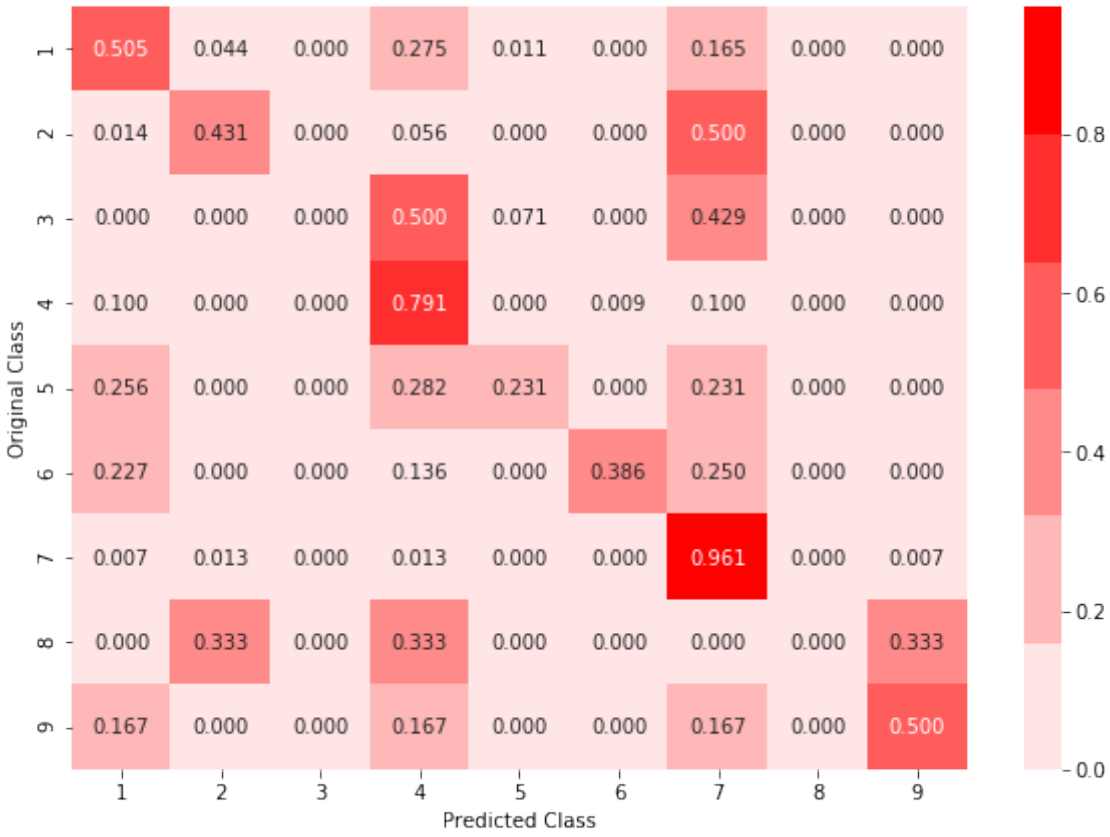
```
Log loss : 1.1550185361134 01
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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



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



In [88]:
```
# Feature Importance
# Correctly Classified point
```

In [89]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
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_onehotC
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1178 0.0897 0.0232 0.1469 0.053  0.0359 0.522  0.0057 0.0058
```

```
Actual Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [activation] present in test data point [True]
4 Text feature [tyrosine] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
8 Text feature [activated] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
10 Text feature [function] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
13 Text feature [growth] present in test data point [True]
15 Text feature [cells] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
19 Text feature [receptor] present in test data point [True]
24 Text feature [downstream] present in test data point [True]
25 Text feature [patients] present in test data point [True]
26 Text feature [kinases] present in test data point [True]
27 Text feature [constitutively] present in test data point [True]
28 Text feature [inhibitor] present in test data point [True]
30 Text feature [functional] present in test data point [True]
31 Text feature [defective] present in test data point [True]
34 Text feature [cell] present in test data point [True]
38 Text feature [f3] present in test data point [True]
40 Text feature [protein] present in test data point [True]
44 Text feature [variants] present in test data point [True]
45 Text feature [ba] present in test data point [True]
47 Text feature [inhibition] present in test data point [True]
48 Text feature [ligand] present in test data point [True]
55 Text feature [expressing] present in test data point [True]
56 Text feature [activate] present in test data point [True]
62 Text feature [proliferation] present in test data point [True]
63 Text feature [serum] present in test data point [True]
66 Text feature [proteins] present in test data point [True]
67 Text feature [expression] present in test data point [True]
71 Text feature [inhibited] present in test data point [True]
91 Text feature [assays] present in test data point [True]
94 Text feature [abolish] present in test data point [True]
96 Text feature [stimulation] present in test data point [True]
98 Text feature [lines] present in test data point [True]
99 Text feature [response] present in test data point [True]
Out of the top  100  features  39 are present in query point
```

```
In [90]: # Inorrectly Classified point

In [91]: 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_onehotC
        print("Actuall Class :", test_y[test_point_index])
        indices = np.argsort(-clf.feature_importances_)
        print("-"*50)
        get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.195  0.1358 0.0262 0.2192 0.065  0.0469 0.297  0.007  0.008
Actuall Class : 1
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [activation] present in test data point [True]
6 Text feature [missense] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
8 Text feature [activated] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
10 Text feature [function] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
13 Text feature [growth] present in test data point [True]
14 Text feature [loss] present in test data point [True]
15 Text feature [cells] present in test data point [True]
16 Text feature [nonsense] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
19 Text feature [receptor] present in test data point [True]
22 Text feature [3t3] present in test data point [True]
23 Text feature [akt] present in test data point [True]
24 Text feature [downstream] present in test data point [True]
25 Text feature [patients] present in test data point [True]
26 Text feature [kinases] present in test data point [True]
27 Text feature [constitutively] present in test data point [True]
28 Text feature [inhibitor] present in test data point [True]
30 Text feature [functional] present in test data point [True]
32 Text feature [frameshift] present in test data point [True]
34 Text feature [cell] present in test data point [True]
40 Text feature [protein] present in test data point [True]
44 Text feature [variants] present in test data point [True]
48 Text feature [ligand] present in test data point [True]
51 Text feature [extracellular] present in test data point [True]
55 Text feature [expressing] present in test data point [True]
56 Text feature [activate] present in test data point [True]
62 Text feature [proliferation] present in test data point [True]
65 Text feature [egfr] present in test data point [True]
66 Text feature [proteins] present in test data point [True]
67 Text feature [expression] present in test data point [True]
74 Text feature [clinical] present in test data point [True]
```

```
84 Text feature [mammalian] present in test data point [True]
85 Text feature [inactivation] present in test data point [True]
91 Text feature [assays] present in test data point [True]
95 Text feature [p53] present in test data point [True]
98 Text feature [lines] present in test data point [True]
Out of the top  100  features  41 are present in query point
```

```
In [92]: # Hyper paramter tuning (With Response Coding)
```

```
In [93]: 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, ra
                 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_, e
                 print("Log Loss :",log_loss(cv_y, sig_clf_probs))


         best_alpha = np.argmin(cv_log_error_array)
         clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         predict_y = sig_clf.predict_proba(train_x_responseCoding)
         print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is
         predict_y = sig_clf.predict_proba(cv_x_responseCoding)
         print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation l
         predict_y = sig_clf.predict_proba(test_x_responseCoding)
         print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:"
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1949948450159757
for n_estimators = 10 and max depth =  3
Log Loss : 1.7048442106496058
for n_estimators = 10 and max depth =  5
Log Loss : 1.373630189681922
for n_estimators = 10 and max depth =  10
Log Loss : 1.929227176169621
for n_estimators = 50 and max depth =  2
Log Loss : 1.6605991250165646
```

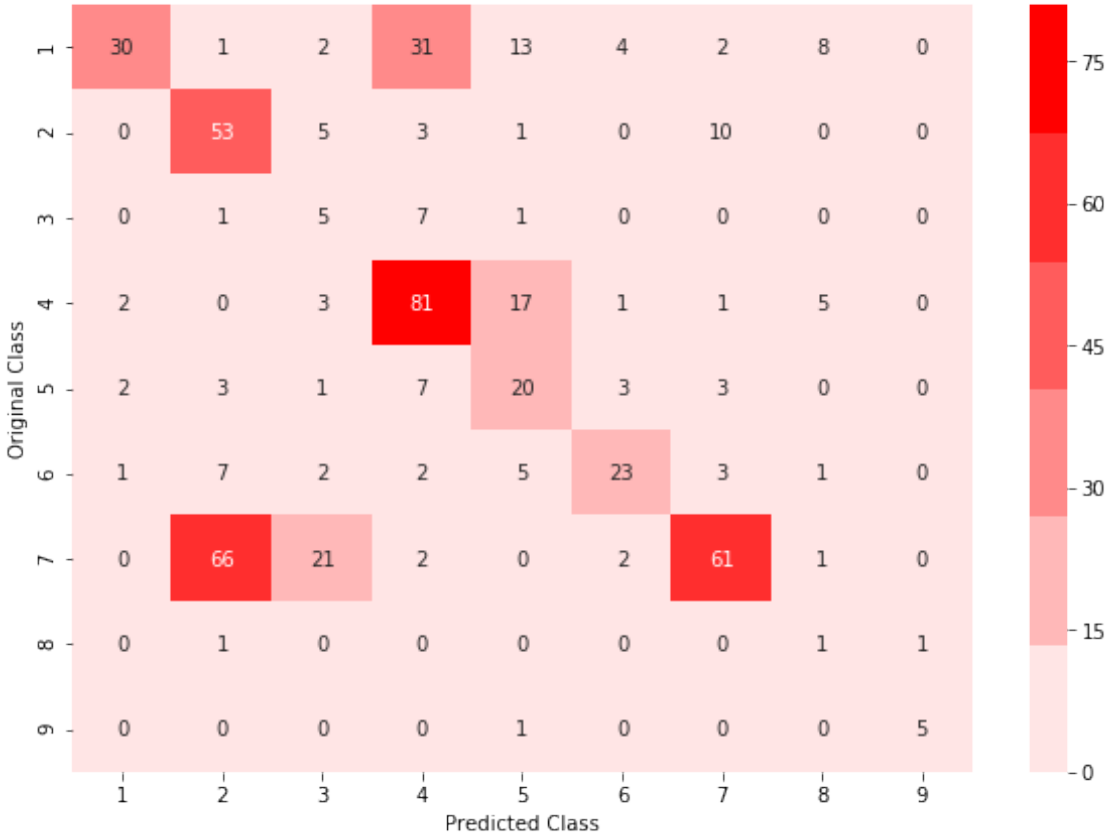
```
for n_estimators = 50 and max depth =  3
Log Loss : 1.4128120521967635
for n_estimators = 50 and max depth =  5
Log Loss : 1.3360438421539043
for n_estimators = 50 and max depth =  10
Log Loss : 1.7452483758209292
for n_estimators = 100 and max depth =  2
Log Loss : 1.556040418777501
for n_estimators = 100 and max depth =  3
Log Loss : 1.4814482074415307
for n_estimators = 100 and max depth =  5
Log Loss : 1.2890227521058697
for n_estimators = 100 and max depth =  10
Log Loss : 1.6420594771953798
for n_estimators = 200 and max depth =  2
Log Loss : 1.5998586562151416
for n_estimators = 200 and max depth =  3
Log Loss : 1.5051243771007619
for n_estimators = 200 and max depth =  5
Log Loss : 1.308840006146743
for n_estimators = 200 and max depth =  10
Log Loss : 1.6129744679651963
for n_estimators = 500 and max depth =  2
Log Loss : 1.6342902614503085
for n_estimators = 500 and max depth =  3
Log Loss : 1.5563373706973112
for n_estimators = 500 and max depth =  5
Log Loss : 1.3383220262562576
for n_estimators = 500 and max depth =  10
Log Loss : 1.6566820256937194
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6190173633039036
for n_estimators = 1000 and max depth =  3
Log Loss : 1.536624305916502
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3279822072770122
for n_estimators = 1000 and max depth =  10
Log Loss : 1.645911561112513
For values of best alpha =  100 The train log loss is: 0.050653522919969905
For values of best alpha =  100 The cross validation log loss is: 1.2890227521058697
For values of best alpha =  100 The test log loss is: 1.2918759133612314
```

In [94]:
```
# Testing model with best hyper parameters (Response Coding)
```

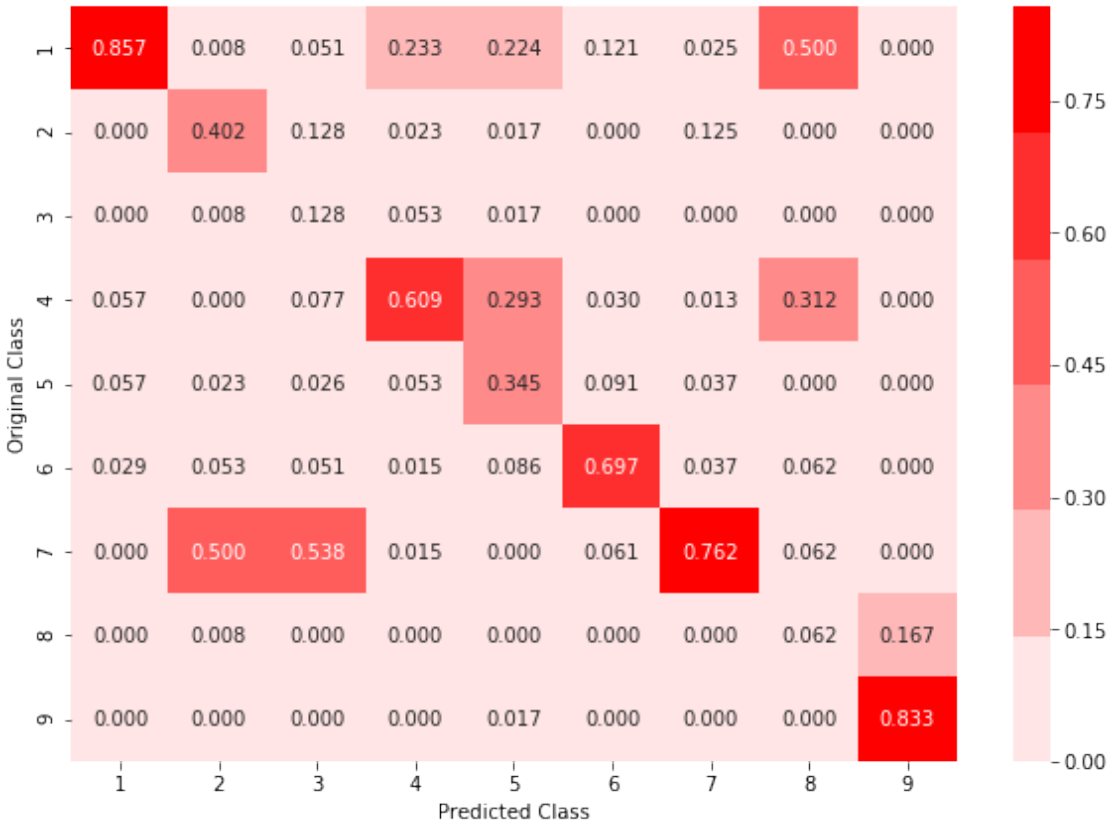
In [95]:
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alp
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding
```

```
Log loss : 1.2890227521058697
```

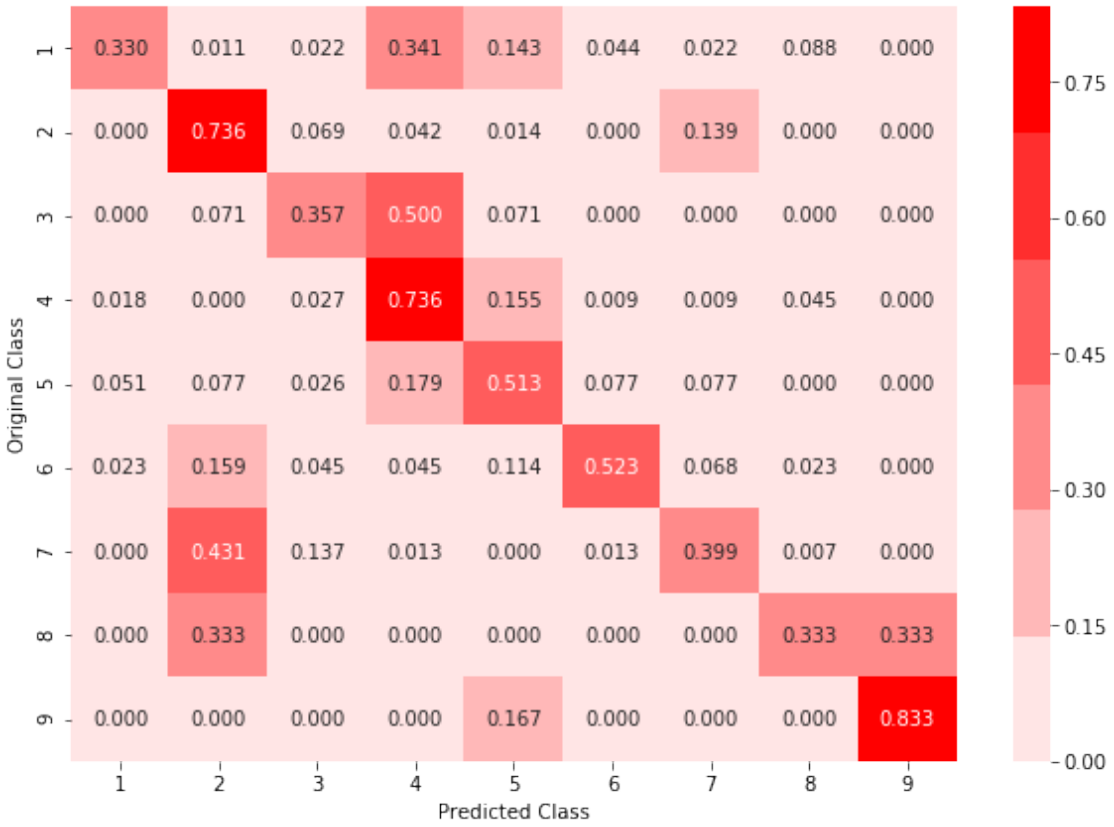
```
Number of mis-classified points : 0.4755639097744361
-------------------- Confusion matrix --------------------
```




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



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



In [96]:
```
# Feature Importance
# Correctly Classified point
```

In [97]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
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_response
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
```

```
            print("Variation is important feature")
        else:
            print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0176 0.1101 0.3042 0.0145 0.0229 0.0371 0.4091 0.0604 0.024
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

```
In [98]: # Incorrectly Classified point
```

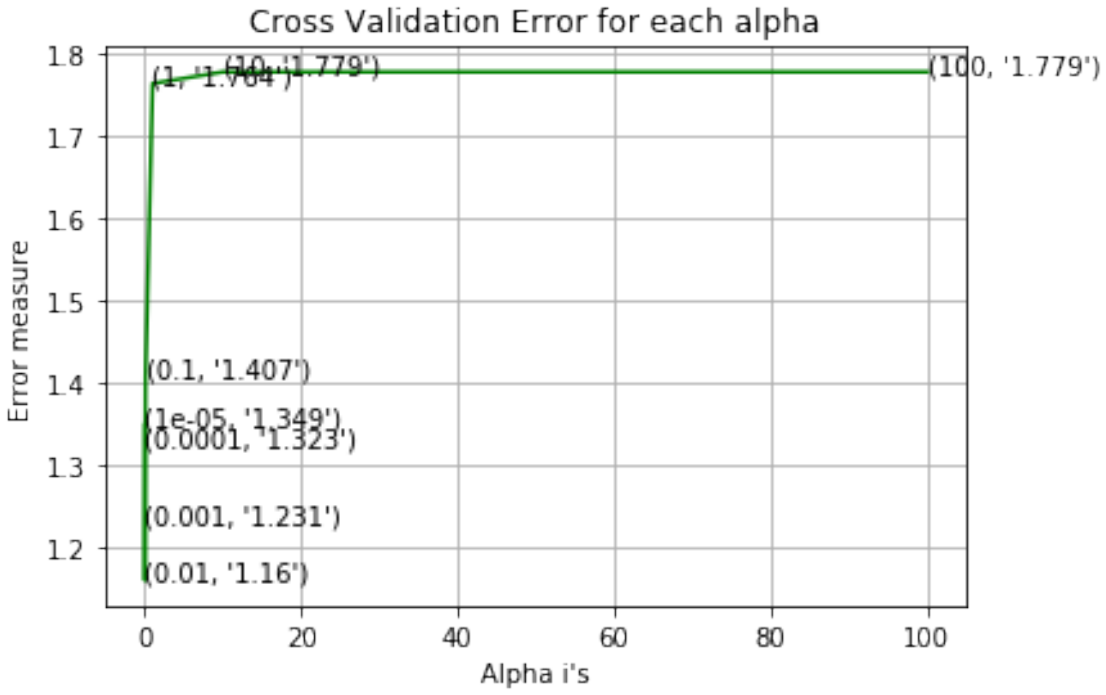
```
In [99]: test_point_index = 100
         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1)
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respons
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         for i in indices:
             if i<9:
```

```
            print("Gene is important feature")
        elif i<18:
            print("Variation is important feature")
        else:
            print("Text is important feature")

Predicted Class : 8
Predicted Class Probabilities: [[0.0504 0.1705 0.1642 0.0587 0.0606 0.0629 0.0665 0.2801 0.086
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

```
In [100]: # Linear Support Vector Machines
          # Hyper paramter tuning

In [101]: alpha = [10 ** x for x in range(-5, 3)]
          cv_log_error_array = []
          for i in alpha:
              print("for C =", i)
          #     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
              clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge
```

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

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


        best_alpha = np.argmin(cv_log_error_array)
        # clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
        clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)

        predict_y = sig_clf.predict_proba(train_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
        predict_y = sig_clf.predict_proba(test_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l
```

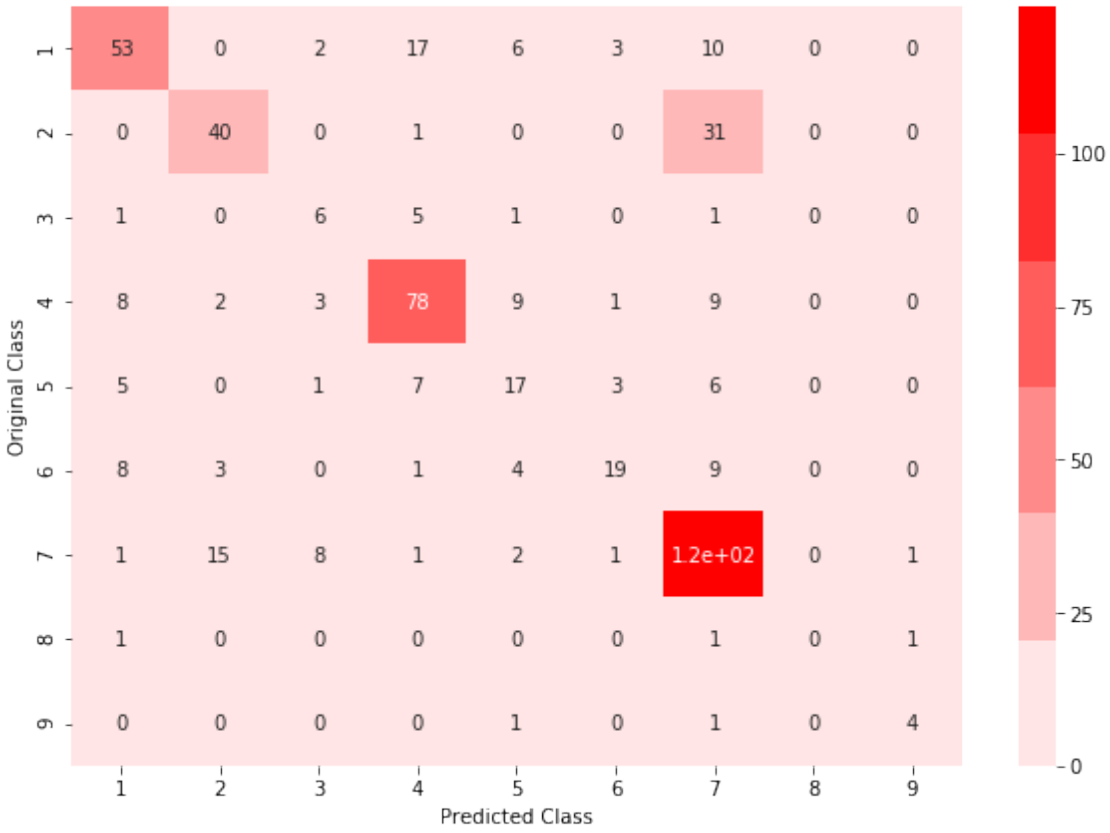
```
for C = 1e-05
Log Loss : 1.3493448357943634
for C = 0.0001
Log Loss : 1.3233749726898993
for C = 0.001
Log Loss : 1.2310610382678415
for C = 0.01
Log Loss : 1.160096704866313
for C = 0.1
Log Loss : 1.4074641667836936
for C = 1
Log Loss : 1.7644408079327045
for C = 10
Log Loss : 1.7785855166692548
for C = 100
Log Loss : 1.778585616887523
```



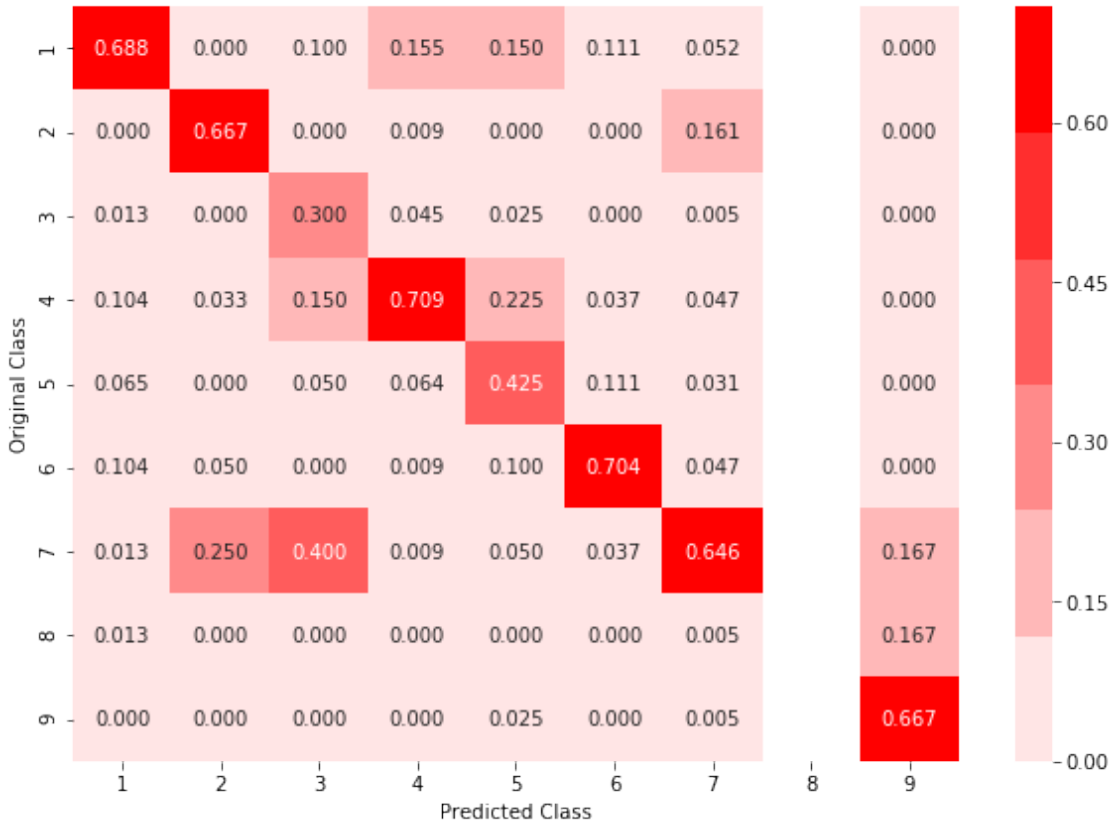

```
For values of best alpha =  0.01 The train log loss is: 0.7443526375441353
For values of best alpha =  0.01 The cross validation log loss is: 1.160096704866313
For values of best alpha =  0.01 The test log loss is: 1.1345685011524151
```

```
In [102]: # Testing model with best hyper parameters

In [103]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
          predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_
```

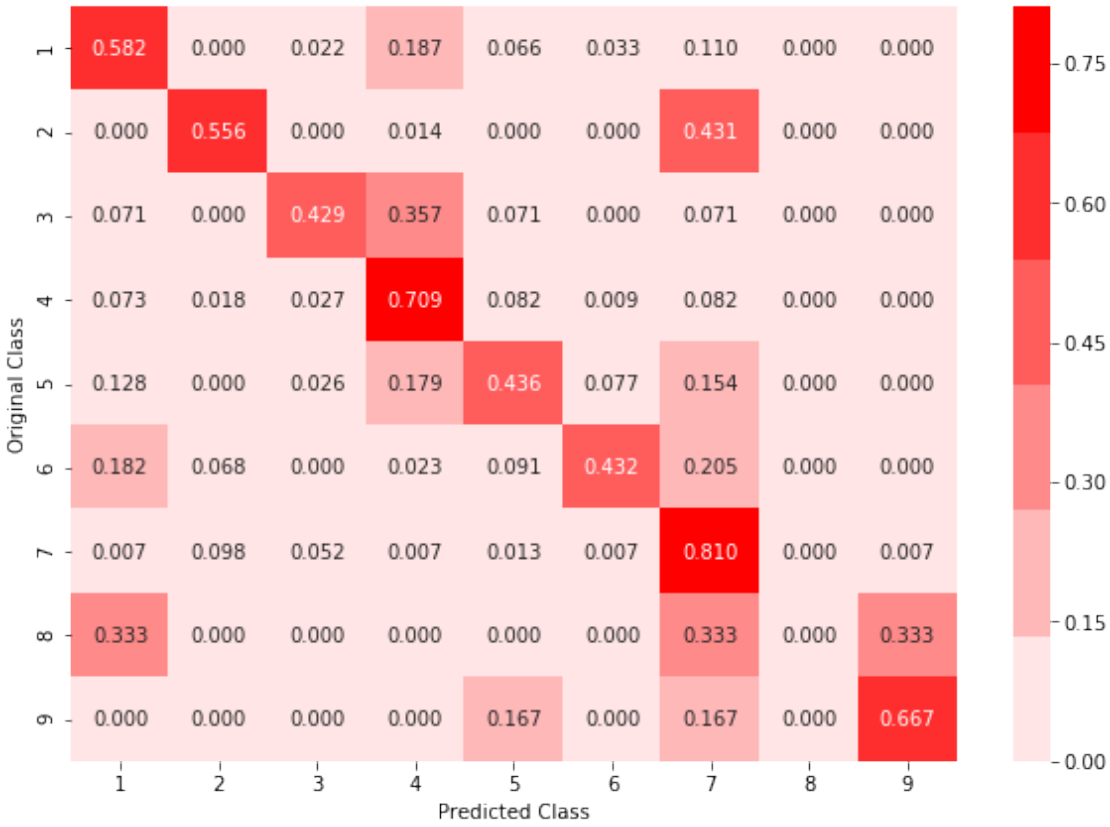
```
Log loss : 1.160096704866313
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```



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



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




```
In [104]: # Feature Importance

In [105]: # For Correctly classified point

In [106]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
          clf.fit(train_x_onehotCoding,train_y)
          test_point_index = 1
          # 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_onehotC
          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

Predicted Class : 7
Predicted Class Probabilities: [[0.087  0.0647 0.0106 0.1199 0.0412 0.0241 0.644  0.0041 0.0045
Actual Class : 7
--------------------------------------------------
```

```
32 Text feature [constitutively] present in test data point [True]
37 Text feature [constitutive] present in test data point [True]
48 Text feature [stat] present in test data point [True]
62 Text feature [expressing] present in test data point [True]
63 Text feature [interleukin] present in test data point [True]
64 Text feature [stat5] present in test data point [True]
77 Text feature [activating] present in test data point [True]
86 Text feature [cdnas] present in test data point [True]
90 Text feature [jak] present in test data point [True]
104 Text feature [activated] present in test data point [True]
113 Text feature [technology] present in test data point [True]
131 Text feature [ligand] present in test data point [True]
253 Text feature [serum] present in test data point [True]
273 Text feature [downstream] present in test data point [True]
383 Text feature [activation] present in test data point [True]
442 Text feature [proliferation] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

```
In [107]: # For Incorrectly classified point

In [108]: 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_onehot
          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['Gen
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1869 0.1248 0.0183 0.2662 0.0559 0.0584 0.2774 0.0051 0.007
Actual Class : 1
--------------------------------------------------
32 Text feature [constitutively] present in test data point [True]
36 Text feature [3t3] present in test data point [True]
37 Text feature [constitutive] present in test data point [True]
62 Text feature [expressing] present in test data point [True]
77 Text feature [activating] present in test data point [True]
104 Text feature [activated] present in test data point [True]
112 Text feature [noncanonical] present in test data point [True]
131 Text feature [ligand] present in test data point [True]
273 Text feature [downstream] present in test data point [True]
294 Text feature [extracellular] present in test data point [True]
383 Text feature [activation] present in test data point [True]
442 Text feature [proliferation] present in test data point [True]
484 Text feature [egf] present in test data point [True]
```

```
Out of the top  500  features  13 are present in query point
```