CancerDiagnosis-Assignment-1

July 24, 2018

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
```

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: Thi
  "This module will be removed in 0.20.", DeprecationWarning)
```

3.1. Reading Data

In [5]:
```
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[5]:

	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 [6]:
```
# note the seprator in this file
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[6]:

	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 [7]:
```
import nltk
nltk.download('stopwords')
import re
```

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

```
In [0]: # loading stop words from nltk library
        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
```

```
In [9]: #merging both gene_variations and text data based on ID
        result = pd.merge(data, data_text,on='ID', how='left')
        result.head()
```

```
Out[9]:    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

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

3.1.4. Test, Train and Cross Validation Split

```
In [0]: 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 varaib
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, t
# split the train data into train and cross validation by maintaining same distributio
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, t
```

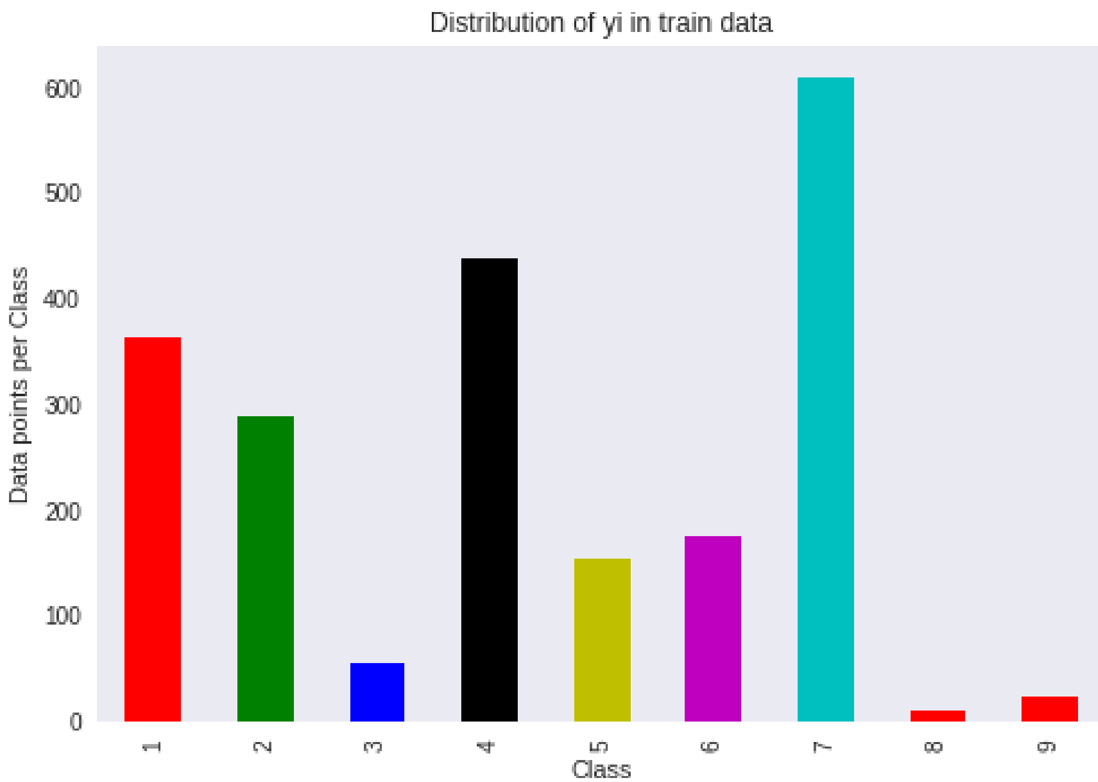
In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

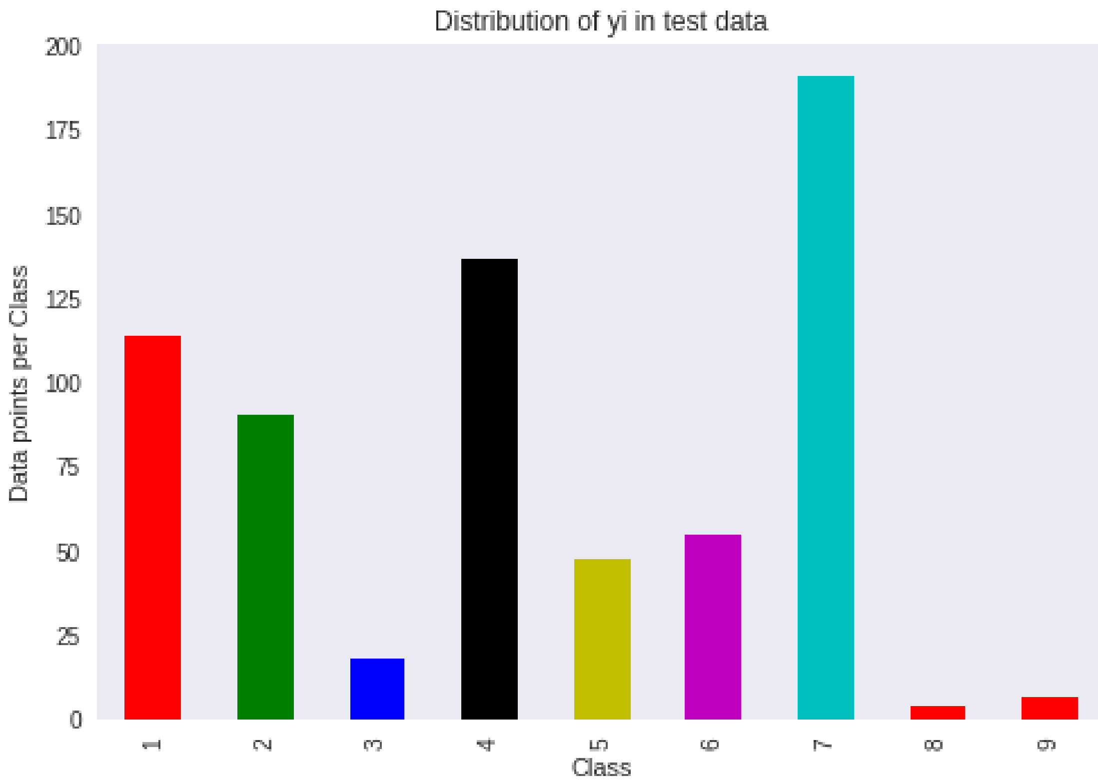
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:
```
# it returns a dict, keys as class labels and values as the number of data points in
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors =  ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar', color=my_colors)
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 =  ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
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]

print('-'*80)
my_colors =  ['r', 'g', 'b', 'k', 'y', 'm', 'c']
```

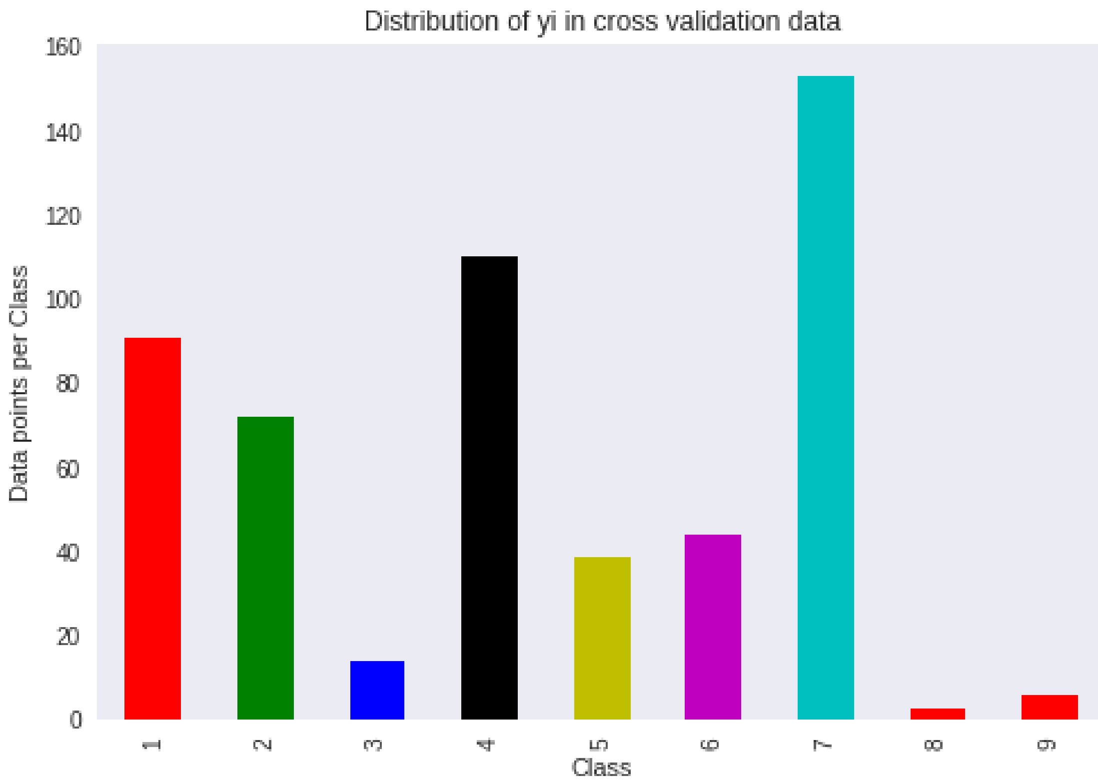
```python
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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],
```




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




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




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

1 Tf-idf Vectorization

In [0]:
```
# Gene and Variation feature
gene_vectorizer = TfidfVectorizer()
train_gene_feature_tfidfCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tfidfCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tfidfCoding = gene_vectorizer.transform(cv_df['Gene'])
```

```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_tfidfCoding = variation_vectorizer.fit_transform(train_df['Vari
test_variation_feature_tfidfCoding = variation_vectorizer.transform(test_df['Variation
cv_variation_feature_tfidfCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
In [14]: # building a CountVectorizer with all the words that occured minimum 3 times in train
         text_vectorizer = TfidfVectorizer()
         train_text_feature_tfidfCoding = text_vectorizer.fit_transform(train_df['TEXT'].values
         # we use the same vectorizer that was trained on train data
         test_text_feature_tfidfCoding = text_vectorizer.transform(test_df['TEXT'].values.asty
         # we use the same vectorizer that was trained on train data
         cv_text_feature_tfidfCoding = text_vectorizer.transform(cv_df['TEXT'].values.astype('
         # 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_tfidfCoding.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 : 128482
```

1.0.1 Stacking Features

```
In [0]: train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_fea
        test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_featu
        cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding,cv_variation_feature_tfi

        train_x_tfidfCoding = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfCodi
        train_y = np.array(list(train_df['Class']))

        test_x_tfidfCoding = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCoding)
        test_y = np.array(list(test_df['Class']))

        cv_x_tfidfCoding = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding)).tocs
        cv_y = np.array(list(cv_df['Class']))
```

2 Machine Learning Models

```
In [0]: # This function plots the confusion matrices given y_i, y_i_hat.
        def plot_confusion_matrix(test_y, predict_y):
            C = confusion_matrix(test_y, predict_y)
            # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predi
```

```
A =(((C.T)/(C.sum(axis=1))).T)
#divid each element of the confusion matrix with the sum of elements in that colum

# C = [[1, 2],
#      [3, 4]]
# C.T = [[1, 3],
#         [2, 4]]
# C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in
# C.sum(axix =1) = [[3, 7]]
# ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
#                            [2/3, 4/7]]

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

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

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

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

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

```
plt.show()
```

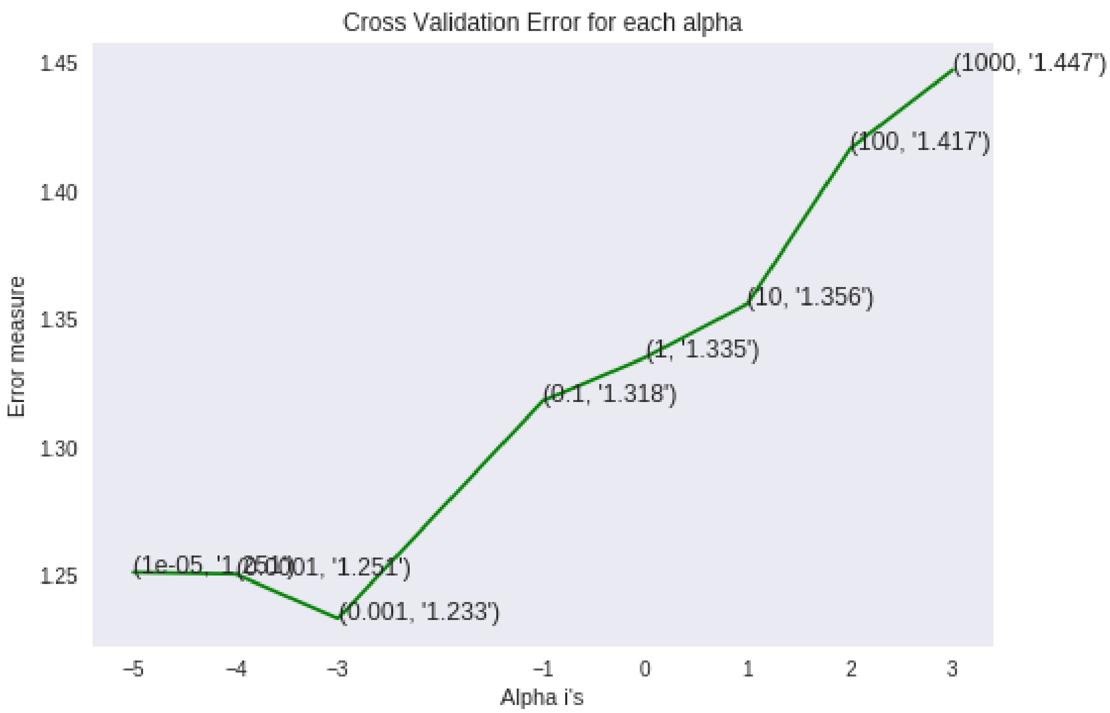
In [0]:
```
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # 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)
```

2.0.1 Naive Bayes

In [18]:
```
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_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    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_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
```

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

```
for alpha = 1e-05
Log Loss : 1.2512110912809997
for alpha = 0.0001
Log Loss : 1.250554910049138
for alpha = 0.001
Log Loss : 1.233236496243429
for alpha = 0.1
Log Loss : 1.318132622621344
for alpha = 1
Log Loss : 1.3350811204639685
for alpha = 10
Log Loss : 1.3561332758753608
for alpha = 100
Log Loss : 1.416674894001924
for alpha = 1000
Log Loss : 1.4472772860629453
```

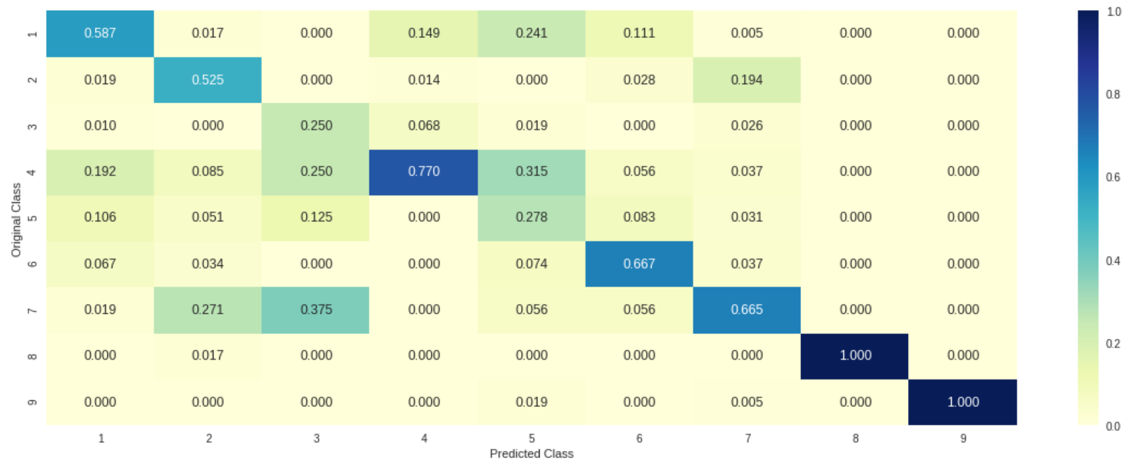


```
For values of best alpha =  0.001 The train log loss is: 0.6000841868355332
For values of best alpha =  0.001 The cross validation log loss is: 1.233236496243429
For values of best alpha =  0.001 The test log loss is: 1.103381147513165
```

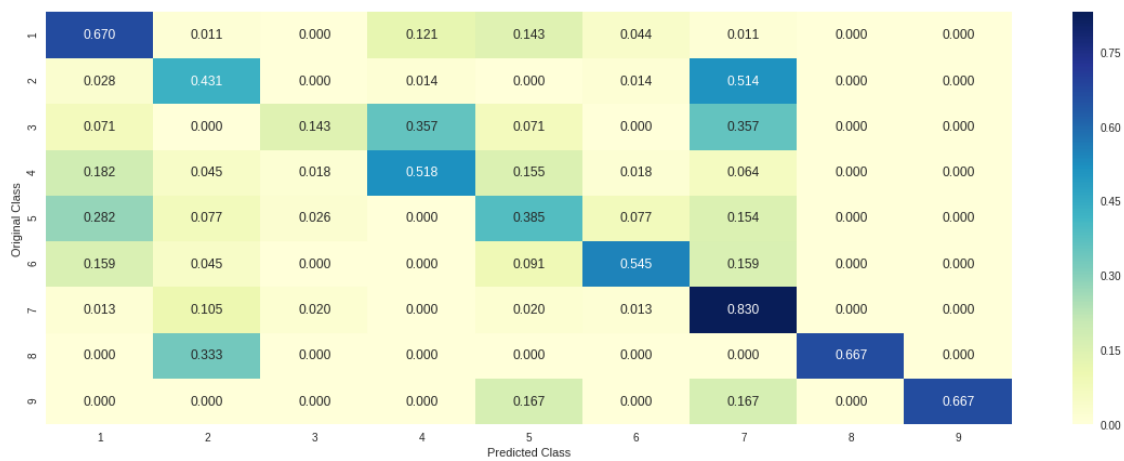
In [22]:
```
##error and confusioon matrix


        clf = MultinomialNB(alpha=0.001)
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
        # 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_tfid
        plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidfCoding.toarray()))
```

```
Log Loss : 1.2513115805283286
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```


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




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



Feature Imortance

In [0]:
```python
# 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 = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer()
```

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

            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(wo
                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 [{}]".forma
                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(wo

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

In [26]:
```
test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCo
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 : 6
Predicted Class Probabilities: [[0.075  0.0595 0.0106 0.0805 0.0362 0.635  0.0976 0.0029 0.002
Actual Class : 6
--------------------------------------------------
0 Text feature [the] present in test data point [True]
2 Text feature [of] present in test data point [True]
3 Text feature [and] present in test data point [True]
5 Text feature [in] present in test data point [True]
```

```
7 Text feature [to] present in test data point [True]
8 Text feature [with] present in test data point [True]
9 Text feature [for] present in test data point [True]
10 Text feature [variants] present in test data point [True]
11 Text feature [that] present in test data point [True]
12 Text feature [deleterious] present in test data point [True]
15 Text feature [were] present in test data point [True]
18 Text feature [mutations] present in test data point [True]
19 Text feature [is] present in test data point [True]
20 Text feature [as] present in test data point [True]
21 Text feature [or] present in test data point [True]
22 Text feature [are] present in test data point [True]
23 Text feature [by] present in test data point [True]
24 Text feature [odds] present in test data point [True]
27 Text feature [we] present in test data point [True]
28 Text feature [was] present in test data point [True]
30 Text feature [be] present in test data point [True]
31 Text feature [this] present in test data point [True]
32 Text feature [from] present in test data point [True]
33 Text feature [these] present in test data point [True]
35 Text feature [on] present in test data point [True]
36 Text feature [neutral] present in test data point [True]
37 Text feature [mutation] present in test data point [True]
38 Text feature [cancer] present in test data point [True]
40 Text feature [variant] present in test data point [True]
42 Text feature [at] present in test data point [True]
46 Text feature [history] present in test data point [True]
48 Text feature [not] present in test data point [True]
49 Text feature [family] present in test data point [True]
50 Text feature [data] present in test data point [True]
52 Text feature [causality] present in test data point [True]
56 Text feature [have] present in test data point [True]
57 Text feature [individuals] present in test data point [True]
59 Text feature [missense] present in test data point [True]
60 Text feature [analysis] present in test data point [True]
63 Text feature [breast] present in test data point [True]
65 Text feature [an] present in test data point [True]
67 Text feature [cells] present in test data point [True]
68 Text feature [ovarian] present in test data point [True]
69 Text feature [classified] present in test data point [True]
70 Text feature [sequence] present in test data point [True]
71 Text feature [type] present in test data point [True]
72 Text feature [classification] present in test data point [True]
76 Text feature [protein] present in test data point [True]
77 Text feature [activity] present in test data point [True]
78 Text feature [domain] present in test data point [True]
79 Text feature [tumor] present in test data point [True]
80 Text feature [which] present in test data point [True]
```

```
81 Text feature [table] present in test data point [True]
82 Text feature [favor] present in test data point [True]
83 Text feature [used] present in test data point [True]
85 Text feature [likelihood] present in test data point [True]
87 Text feature [using] present in test data point [True]
89 Text feature [risk] present in test data point [True]
91 Text feature [all] present in test data point [True]
92 Text feature [our] present in test data point [True]
93 Text feature [10] present in test data point [True]
94 Text feature [been] present in test data point [True]
95 Text feature [two] present in test data point [True]
96 Text feature [et] present in test data point [True]
98 Text feature [wild] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

In [27]:
```
test_point_index = 25
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCo
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: [[0.2643 0.0573 0.01   0.4976 0.0348 0.0357 0.0949 0.0028 0.002
Actual Class : 4
--------------------------------------------------
0 Text feature [the] present in test data point [True]
1 Text feature [of] present in test data point [True]
2 Text feature [and] present in test data point [True]
3 Text feature [in] present in test data point [True]
5 Text feature [to] present in test data point [True]
9 Text feature [with] present in test data point [True]
10 Text feature [that] present in test data point [True]
11 Text feature [mutations] present in test data point [True]
13 Text feature [for] present in test data point [True]
14 Text feature [were] present in test data point [True]
15 Text feature [by] present in test data point [True]
16 Text feature [is] present in test data point [True]
17 Text feature [as] present in test data point [True]
18 Text feature [was] present in test data point [True]
20 Text feature [cells] present in test data point [True]
21 Text feature [or] present in test data point [True]
23 Text feature [are] present in test data point [True]
26 Text feature [we] present in test data point [True]
```

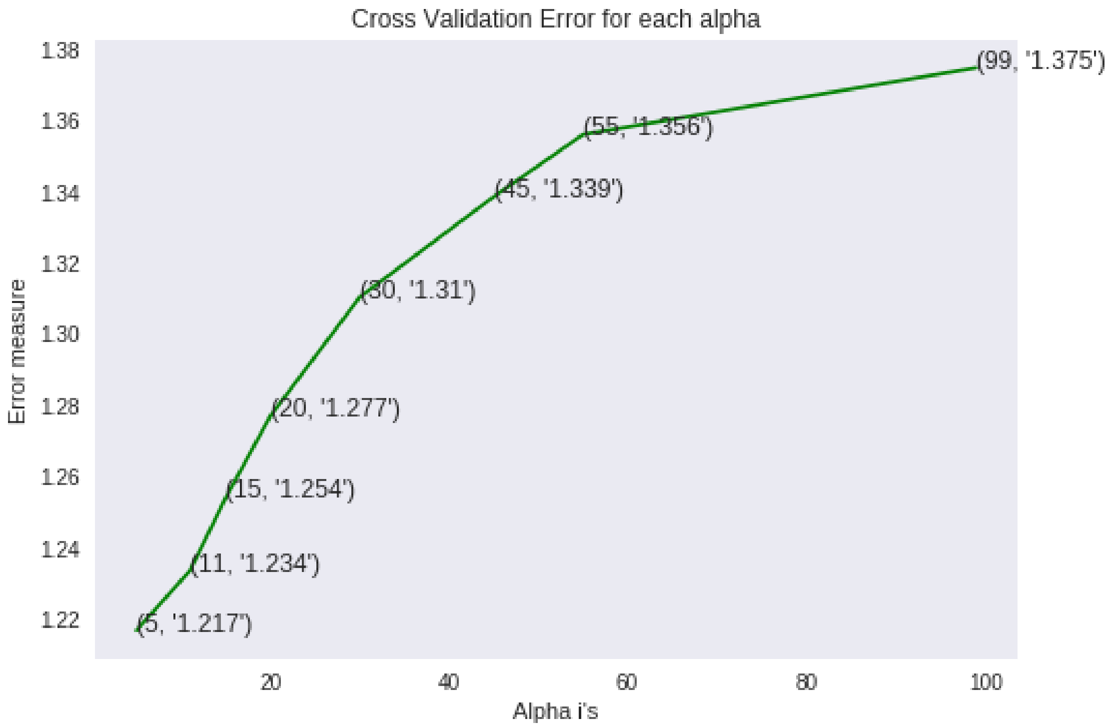
```
27 Text feature [et] present in test data point [True]
29 Text feature [at] present in test data point [True]
30 Text feature [this] present in test data point [True]
31 Text feature [activity] present in test data point [True]
32 Text feature [from] present in test data point [True]
33 Text feature [protein] present in test data point [True]
40 Text feature [variants] present in test data point [True]
41 Text feature [on] present in test data point [True]
43 Text feature [cell] present in test data point [True]
44 Text feature [these] present in test data point [True]
46 Text feature [mutation] present in test data point [True]
48 Text feature [not] present in test data point [True]
53 Text feature [be] present in test data point [True]
54 Text feature [mutants] present in test data point [True]
57 Text feature [binding] present in test data point [True]
58 Text feature [type] present in test data point [True]
59 Text feature [mutant] present in test data point [True]
60 Text feature [have] present in test data point [True]
61 Text feature [an] present in test data point [True]
63 Text feature [expression] present in test data point [True]
65 Text feature [tumor] present in test data point [True]
70 Text feature [wild] present in test data point [True]
71 Text feature [proteins] present in test data point [True]
72 Text feature [which] present in test data point [True]
73 Text feature [missense] present in test data point [True]
76 Text feature [been] present in test data point [True]
80 Text feature [function] present in test data point [True]
88 Text feature [functional] present in test data point [True]
89 Text feature [domain] present in test data point [True]
90 Text feature [also] present in test data point [True]
91 Text feature [all] present in test data point [True]
93 Text feature [using] present in test data point [True]
94 Text feature [gene] present in test data point [True]
96 Text feature [analysis] present in test data point [True]
98 Text feature [two] present in test data point [True]
Out of the top  100  features  53 are present in query point
```

3 K nearest neighbour

In [28]:
```
alpha = [5, 11, 15, 20, 30, 45, 55, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
            sig_clf.fit(train_x_tfidfCoding, train_y)
            sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
            cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
            # 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_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
for alpha = 5
Log Loss : 1.2168235852288414
for alpha = 11
Log Loss : 1.233581169667012
for alpha = 15
Log Loss : 1.25406241861012
for alpha = 20
Log Loss : 1.2769914721938076
for alpha = 30
Log Loss : 1.3103396569723358
for alpha = 45
Log Loss : 1.3385420589564987
for alpha = 55
Log Loss : 1.355994148267625
for alpha = 99
Log Loss : 1.3747314171127818
```

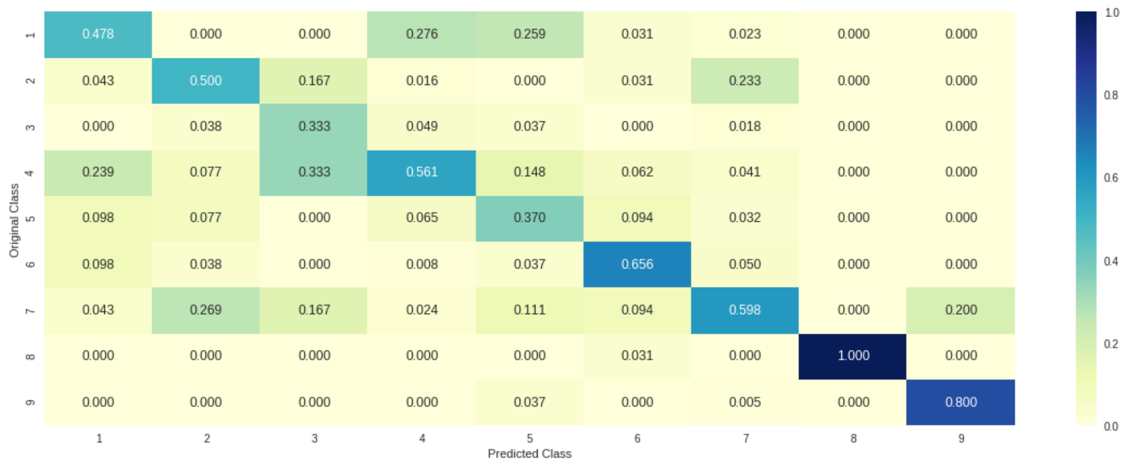



```
For values of best alpha =  5 The train log loss is: 0.8767975906447277
For values of best alpha =  5 The cross validation log loss is: 1.2168235852288414
For values of best alpha =  5 The test log loss is: 1.1272631651080673
```

In [33]:
```
#testing
clf = KNeighborsClassifier(n_neighbors=5,algorithm='brute')
predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y

#plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidfCoding.toarray()))
```

```
Log loss : 1.2168235852288414
Number of mis-classified points : 0.44360902255639095
-------------------- Confusion matrix --------------------
```

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



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

3.0.1 Feature importance

In [34]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 50
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to cla
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 6
The  5  nearest neighbours of the test points belongs to classes [1 5 1 5 5]
Fequency of nearest points : Counter({5: 3, 1: 2})
```

In [36]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 95
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
```

```
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to cla
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 4
The  5  nearest neighbours of the test points belongs to classes [6 4 4 4 5]
Fequency of nearest points : Counter({4: 3, 6: 1, 5: 1})
```

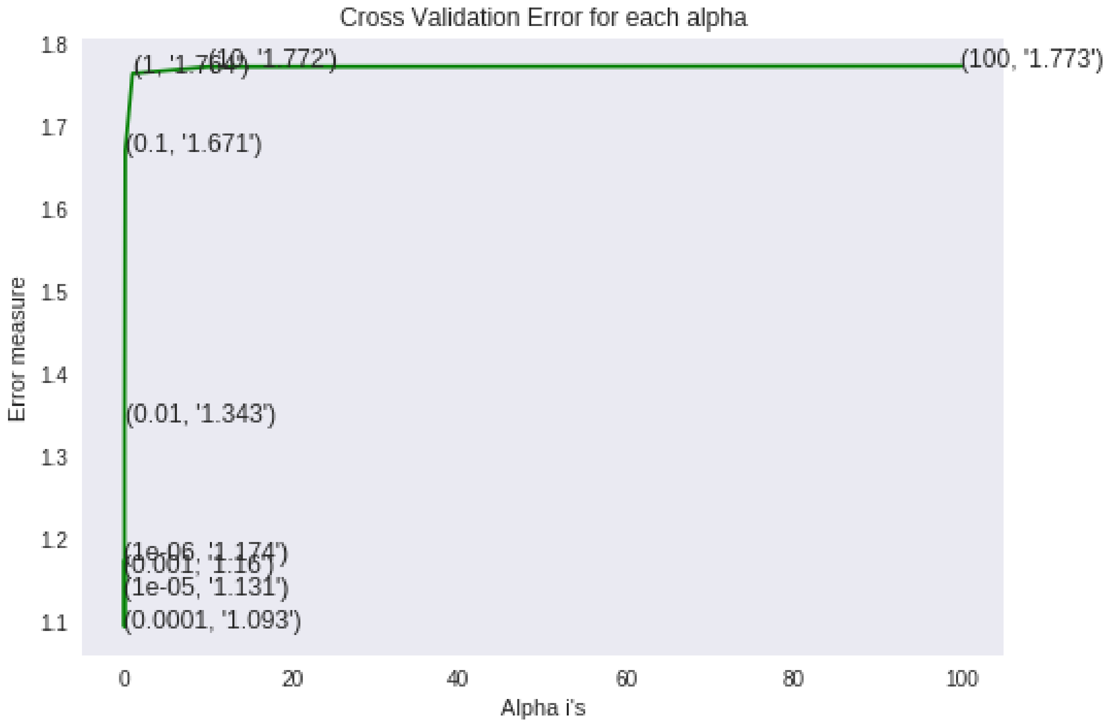
4 Logistic Regression

4.0.1 With Class Balancing

In [37]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    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 = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

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

```
for alpha = 1e-06
Log Loss : 1.174085592435071
for alpha = 1e-05
Log Loss : 1.1312737373316668
for alpha = 0.0001
Log Loss : 1.0931957547758493
for alpha = 0.001
Log Loss : 1.160297246672394
for alpha = 0.01
Log Loss : 1.3430805607641998
for alpha = 0.1
Log Loss : 1.6705349112984504
for alpha = 1
Log Loss : 1.7637334454676454
for alpha = 10
Log Loss : 1.7718063353408449
for alpha = 100
Log Loss : 1.7726335300955434
```




```
For values of best alpha =  0.0001 The train log loss is: 0.40621785221940354
For values of best alpha =  0.0001 The cross validation log loss is: 1.0931957547758493
```

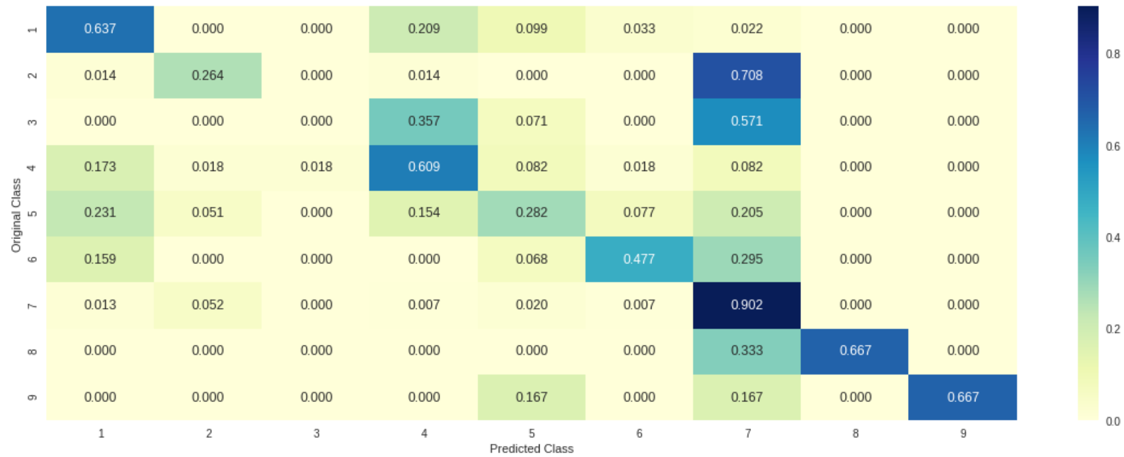
```
For values of best alpha =  0.0001 The test log loss is: 0.9499571788606144
```

```
In [38]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y
```

```
Log loss : 1.0931957547758493
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```


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


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



4.0.2 Feature Importance

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

```
In [42]: clf = SGDClassifier(class_weight='balanced', alpha=0.0001, penalty='l2', loss='log', r
         clf.fit(train_x_tfidfCoding,train_y)
         test_point_index = 1
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCod
         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 : 6
Predicted Class Probabilities: [[0.1327 0.0072 0.0044 0.1805 0.1148 0.5523 0.0058 0.0011 0.001
Actual Class : 5
--------------------------------------------------
72 Text feature [deleterious] present in test data point [True]
116 Text feature [odds] present in test data point [True]
152 Text feature [individuals] present in test data point [True]
182 Text feature [history] present in test data point [True]
197 Text feature [wildtype] present in test data point [True]
206 Text feature [binding] present in test data point [True]
234 Text feature [family] present in test data point [True]
238 Text feature [values] present in test data point [True]
239 Text feature [interaction] present in test data point [True]
243 Text feature [polymorphism] present in test data point [True]
245 Text feature [models] present in test data point [True]
248 Text feature [classified] present in test data point [True]
251 Text feature [site] present in test data point [True]
268 Text feature [are] present in test data point [True]
282 Text feature [substitutions] present in test data point [True]
284 Text feature [residues] present in test data point [True]
291 Text feature [57] present in test data point [True]
313 Text feature [homozygous] present in test data point [True]
316 Text feature [classification] present in test data point [True]
334 Text feature [ethnic] present in test data point [True]
341 Text feature [significant] present in test data point [True]
360 Text feature [missense] present in test data point [True]
385 Text feature [ring] present in test data point [True]
393 Text feature [risk] present in test data point [True]
403 Text feature [copy] present in test data point [True]
424 Text feature [breast] present in test data point [True]
437 Text feature [personal] present in test data point [True]
443 Text feature [favor] present in test data point [True]
444 Text feature [studies] present in test data point [True]
459 Text feature [mutations] present in test data point [True]
460 Text feature [identified] present in test data point [True]
462 Text feature [substitution] present in test data point [True]
467 Text feature [basis] present in test data point [True]
469 Text feature [there] present in test data point [True]
475 Text feature [conformation] present in test data point [True]
484 Text feature [cosegregation] present in test data point [True]
487 Text feature [induce] present in test data point [True]
489 Text feature [active] present in test data point [True]
493 Text feature [evidence] present in test data point [True]
Out of the top  500  features  39 are present in query point
```

```
In [44]: clf = SGDClassifier(class_weight='balanced', alpha=0.0001, penalty='l2', loss='log', r
         clf.fit(train_x_tfidfCoding,train_y)
         test_point_index = 26
         no_feature = 500
         predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCod
         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
```

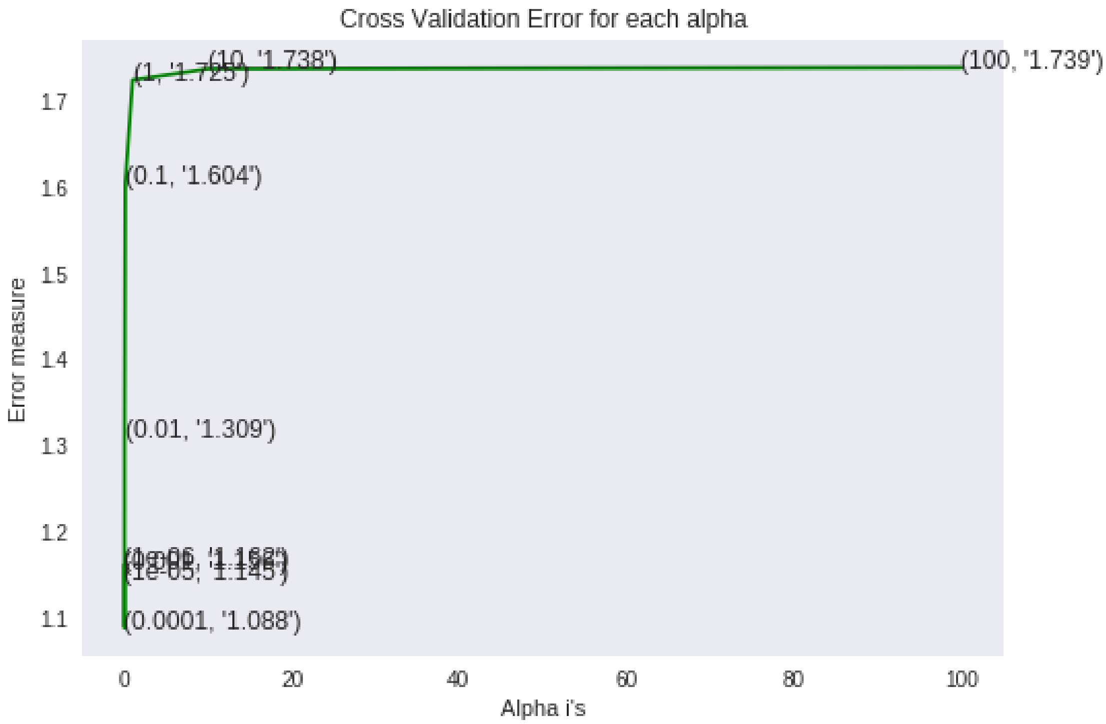
```
Predicted Class : 1
Predicted Class Probabilities: [[0.7561 0.0387 0.0022 0.1383 0.0087 0.0315 0.0145 0.0058 0.004
Actual Class : 1
--------------------------------------------------
165 Text feature [structure] present in test data point [True]
211 Text feature [transcriptional] present in test data point [True]
257 Text feature [hotspot] present in test data point [True]
293 Text feature [surface] present in test data point [True]
324 Text feature [manuscript] present in test data point [True]
344 Text feature [binding] present in test data point [True]
374 Text feature [function] present in test data point [True]
432 Text feature [residues] present in test data point [True]
464 Text feature [type] present in test data point [True]
470 Text feature [domains] present in test data point [True]
482 Text feature [splicing] present in test data point [True]
491 Text feature [panel] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

4.0.3 Without class balancing

```
In [45]: alpha = [10 ** x for x in range(-6, 3)]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             #clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear
             clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf.fit(train_x_tfidfCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidfCoding, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
             # 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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1615031562998066
for alpha = 1e-05
Log Loss : 1.1450179904963886
for alpha = 0.0001
Log Loss : 1.088007192453907
for alpha = 0.001
Log Loss : 1.1555548867608998
for alpha = 0.01
Log Loss : 1.308690431434055
for alpha = 0.1
Log Loss : 1.6039838810498346
for alpha = 1
Log Loss : 1.7245375069573017
for alpha = 10
Log Loss : 1.7376865831359651
for alpha = 100
Log Loss : 1.7391087878414682
```



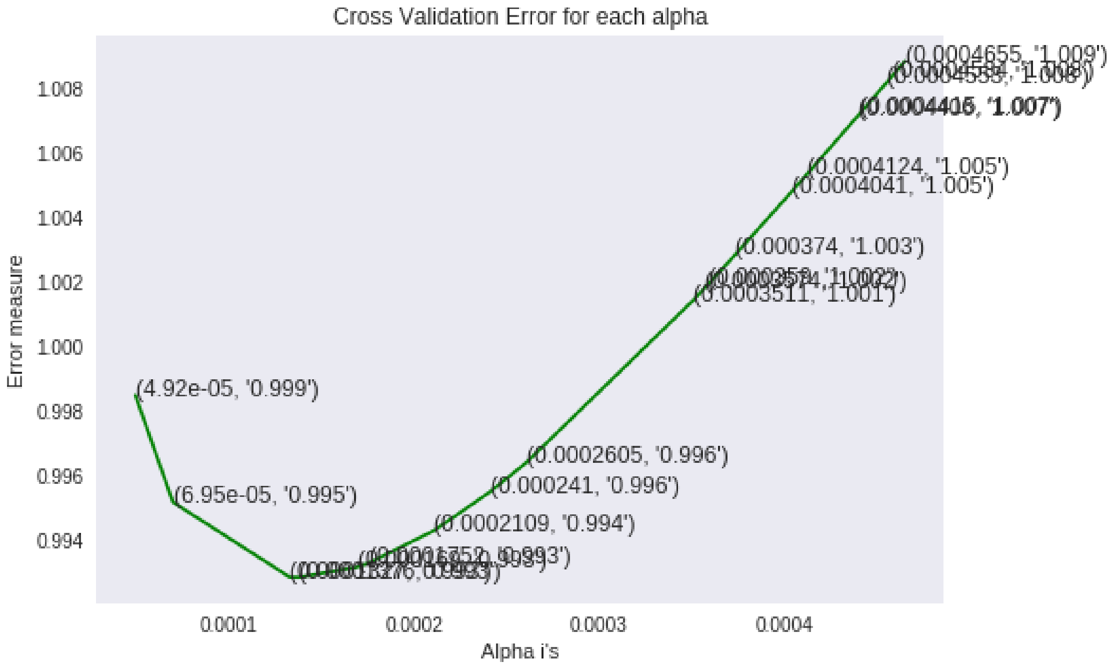

```
For values of best alpha =  0.0001 The train log loss is: 0.39574785564034176
For values of best alpha =  0.0001 The cross validation log loss is: 1.088007192453907
For values of best alpha =  0.0001 The test log loss is: 0.9500785220261735
```

```
In [20]: alpha = np.random.uniform(0.00002,0.0005,20)
         alpha = np.round(alpha,7)
         alpha.sort()
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             #clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear
             clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
             clf.fit(train_x_tfidfCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidfCoding, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
             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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
for alpha = 4.92e-05
Log Loss : 0.9985082572605175
for alpha = 6.95e-05
Log Loss : 0.9952008423870486
for alpha = 0.0001327
Log Loss : 0.9928827426261161
for alpha = 0.0001376
Log Loss : 0.992878794138204
for alpha = 0.000169
Log Loss : 0.9931891450602036
for alpha = 0.0001752
Log Loss : 0.9933106330823002
for alpha = 0.0002109
Log Loss : 0.9943231046209637
for alpha = 0.000241
Log Loss : 0.9955125357484811
for alpha = 0.0002605
Log Loss : 0.9964095601919558
for alpha = 0.0003511
Log Loss : 1.0014447299007243
for alpha = 0.0003574
Log Loss : 1.0018315490384146
for alpha = 0.000359
Log Loss : 1.0019303226412006
for alpha = 0.000374
Log Loss : 1.0028660240217675
for alpha = 0.0004041
```

```
Log Loss : 1.0047883417142436
for alpha = 0.0004124
Log Loss : 1.0053269130680313
for alpha = 0.0004406
Log Loss : 1.007177376059949
for alpha = 0.0004415
Log Loss : 1.007236873796161
for alpha = 0.0004553
Log Loss : 1.0081518650587897
for alpha = 0.0004584
Log Loss : 1.0083580384477318
for alpha = 0.0004655
Log Loss : 1.008831003338642
```



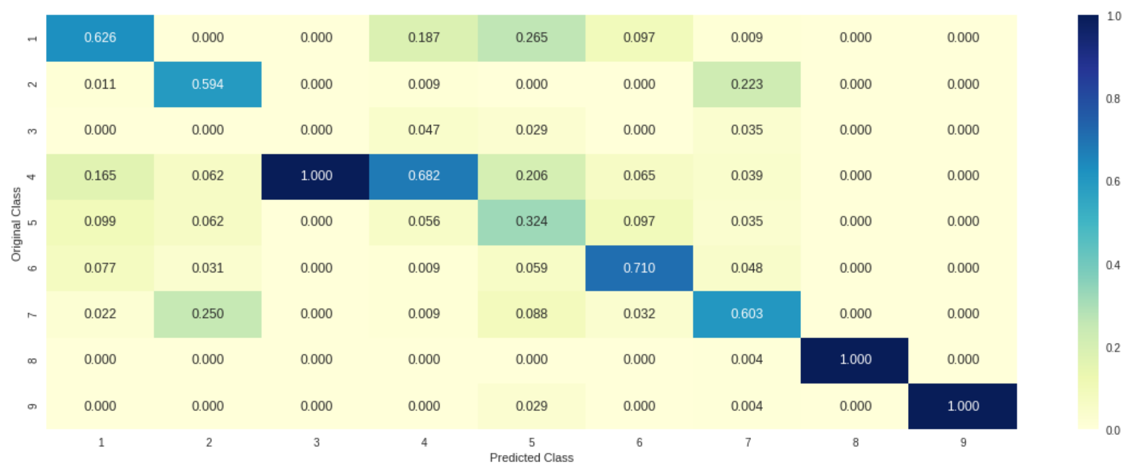

```
For values of best alpha =  0.0001376 The train log loss is: 0.4495380740785141
For values of best alpha =  0.0001376 The cross validation log loss is: 0.992878794138204
For values of best alpha =  0.0001376 The test log loss is: 0.9663254273959704
```

In [47]:
```
#testing
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

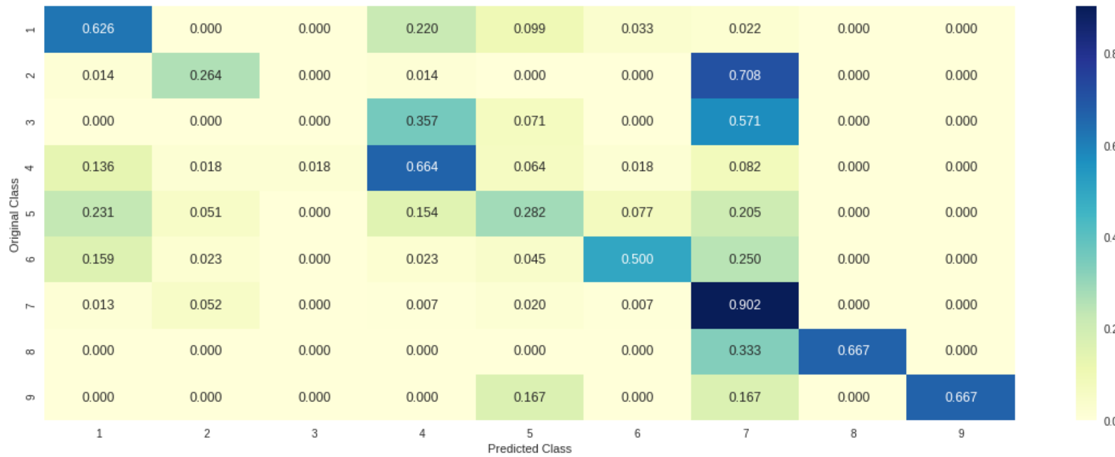
```
Log loss : 1.08788163196963
Number of mis-classified points : 0.38721804511278196
```

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

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



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



In [48]:
```
# from tabulate import tabulate
clf = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 25
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCod
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 : 1
Predicted Class Probabilities: [[0.3731 0.0526 0.0094 0.3369 0.0325 0.0235 0.162  0.0049 0.005
Actual Class : 4
--------------------------------------------------
227 Text feature [transcriptional] present in test data point [True]
291 Text feature [families] present in test data point [True]
340 Text feature [manuscript] present in test data point [True]
352 Text feature [binding] present in test data point [True]
366 Text feature [thrombocytopenia] present in test data point [True]
370 Text feature [function] present in test data point [True]
451 Text feature [type] present in test data point [True]
466 Text feature [skipping] present in test data point [True]
491 Text feature [splicing] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

In [25]:
```
# from tabulate import tabulate
clf = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding,train_y)
```

```
test_point_index = 15
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoc
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
```

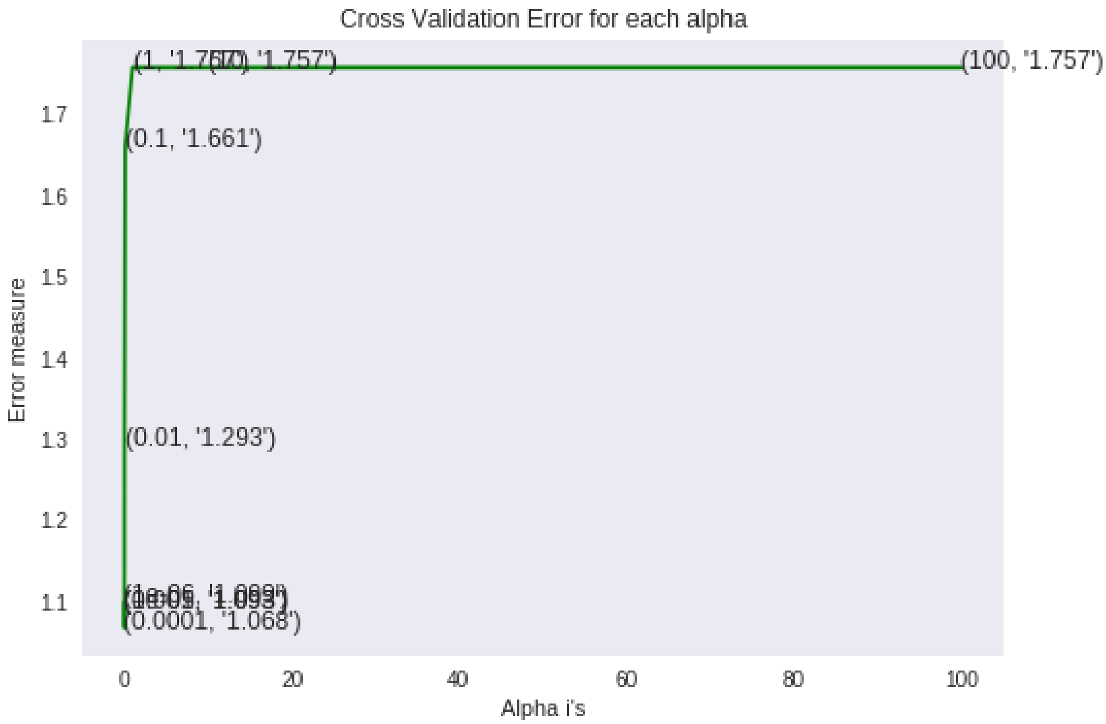
```
Predicted Class : 2
Predicted Class Probabilities: [[2.580e-02 8.027e-01 6.600e-03 2.140e-02 1.730e-02 1.790e-02 1
  3.200e-03 6.000e-04]]
Actual Class : 2
--------------------------------------------------
10 Text feature [patients] present in test data point [True]
187 Text feature [response] present in test data point [True]
271 Text feature [clinical] present in test data point [True]
288 Text feature [treatment] present in test data point [True]
297 Text feature [imatinib] present in test data point [True]
319 Text feature [resistance] present in test data point [True]
340 Text feature [therapy] present in test data point [True]
362 Text feature [months] present in test data point [True]
380 Text feature [patient] present in test data point [True]
391 Text feature [group] present in test data point [True]
404 Text feature [who] present in test data point [True]
409 Text feature [number] present in test data point [True]
423 Text feature [sequencing] present in test data point [True]
427 Text feature [gene] present in test data point [True]
435 Text feature [tumor] present in test data point [True]
446 Text feature [time] present in test data point [True]
455 Text feature [was] present in test data point [True]
466 Text feature [mutational] present in test data point [True]
475 Text feature [samples] present in test data point [True]
Out of the top  500  features  19 are present in query point
```

5 Linear SVMs

```
In [26]: alpha = [10 ** x for x in range(-6, 3)]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42,class_we
             clf.fit(train_x_tfidfCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidfCoding, train_y)
```

```
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
        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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.099133190797159
for alpha = 1e-05
Log Loss : 1.092928969224864
for alpha = 0.0001
Log Loss : 1.0676203931784294
for alpha = 0.001
Log Loss : 1.0930205880659987
for alpha = 0.01
Log Loss : 1.292599876761465
for alpha = 0.1
Log Loss : 1.660697514751692
for alpha = 1
Log Loss : 1.7572657883770801
for alpha = 10
Log Loss : 1.7572657986577993
for alpha = 100
Log Loss : 1.7572658044559226
```



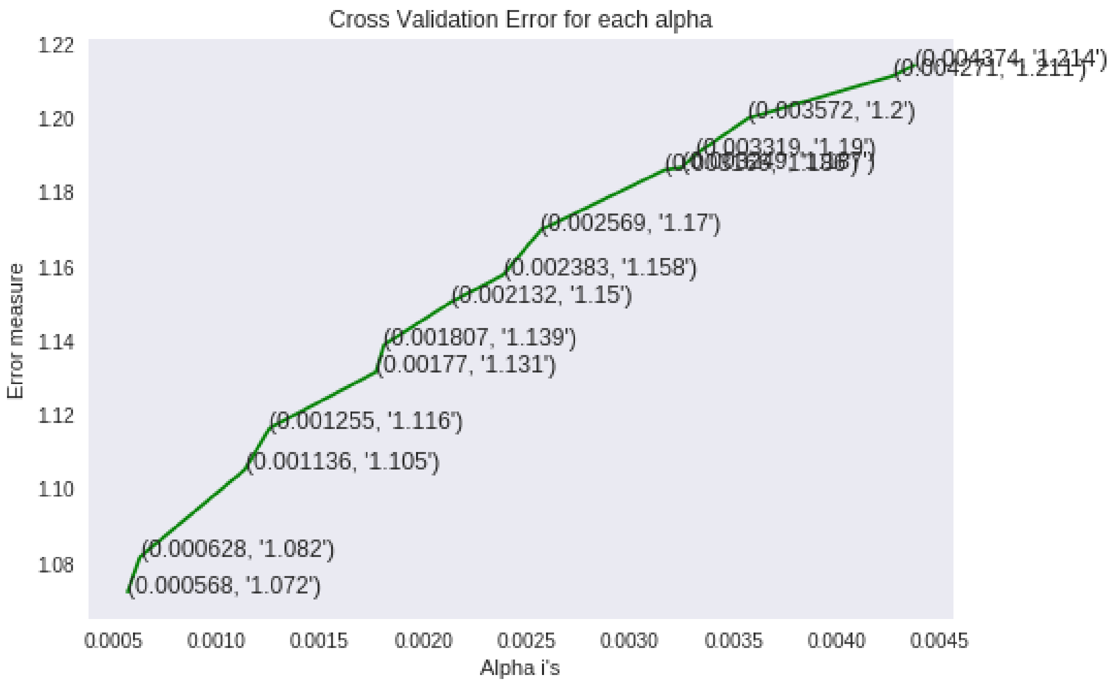

```
For values of best alpha =  0.0001 The train log loss is: 0.4955814764793545
For values of best alpha =  0.0001 The cross validation log loss is: 1.0794117835910462
For values of best alpha =  0.0001 The test log loss is: 1.0530895551479529
```

In [27]:
```
alpha = np.random.uniform(0.0002,0.005,15)
alpha = np.round(alpha,6)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier( alpha=i, penalty='l2', loss='hinge', random_state=42,class_we
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
    # 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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
for alpha = 0.000568
Log Loss : 1.0721049749053675
for alpha = 0.000628
Log Loss : 1.0815821721582675
for alpha = 0.001136
Log Loss : 1.1051661166583087
for alpha = 0.001255
Log Loss : 1.1163814682582782
for alpha = 0.00177
Log Loss : 1.131471044747968
for alpha = 0.001807
Log Loss : 1.138778040443421
for alpha = 0.002132
Log Loss : 1.1503358223402862
for alpha = 0.002383
Log Loss : 1.1576035778845157
for alpha = 0.002569
Log Loss : 1.1699061451891892
for alpha = 0.003166
Log Loss : 1.1860130339950816
for alpha = 0.003249
Log Loss : 1.1867128602705175
for alpha = 0.003319
Log Loss : 1.1903322827993128
for alpha = 0.003572
Log Loss : 1.2000601959371962
for alpha = 0.004271
Log Loss : 1.2114233933234364
```

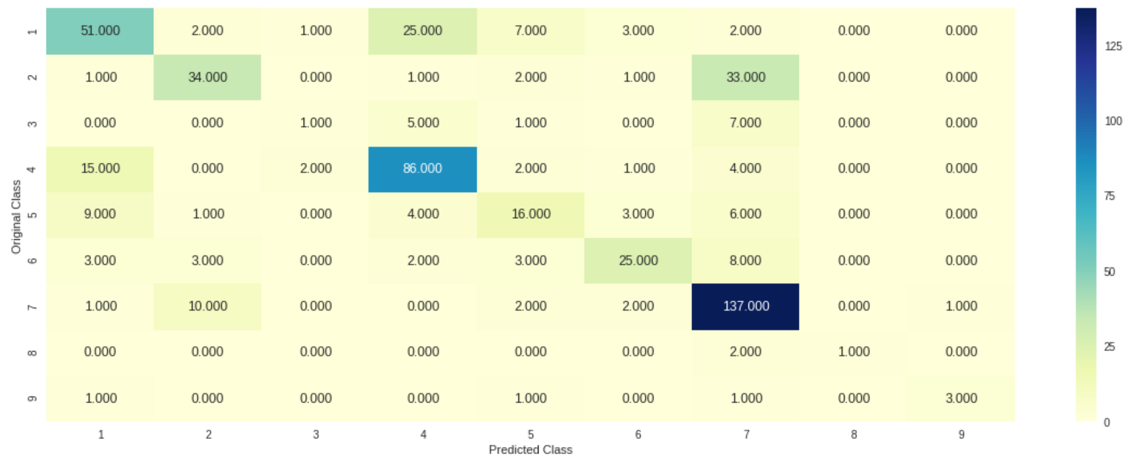
```
for alpha = 0.004374
Log Loss : 1.2143116612042137
```



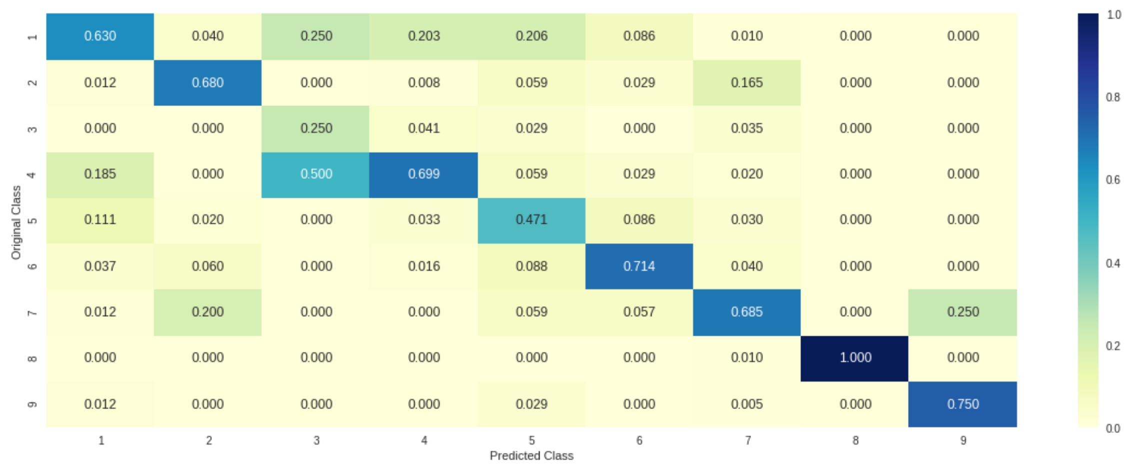

```
For values of best alpha =  0.000568 The train log loss is: 0.4935728353962843
For values of best alpha =  0.000568 The cross validation log loss is: 1.0721049749053675
For values of best alpha =  0.000568 The test log loss is: 1.0467308795505816
```

```
In [28]: ##testing
         clf = SGDClassifier(alpha=0.000568, penalty='l2', loss='hinge', random_state=42,class_
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

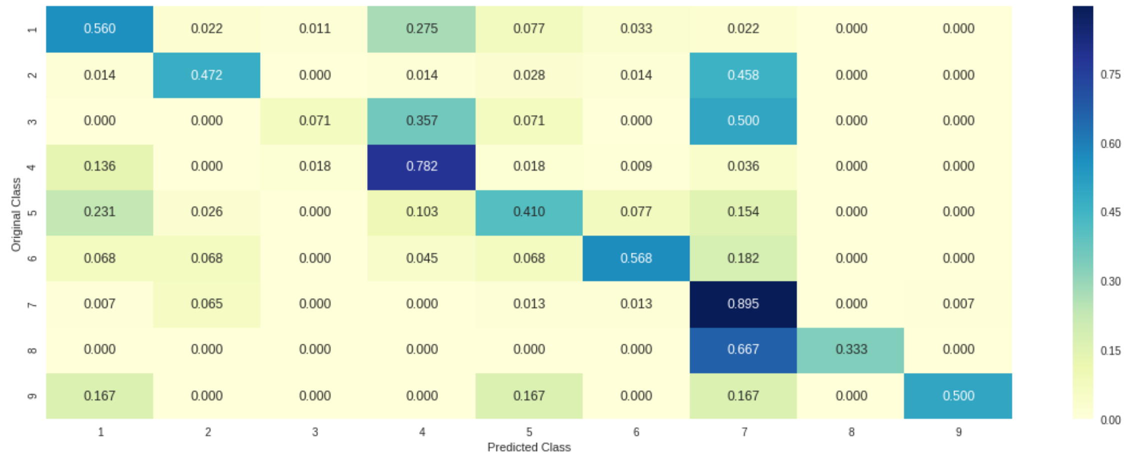
```
Log loss : 1.0721049749053675
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```



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



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

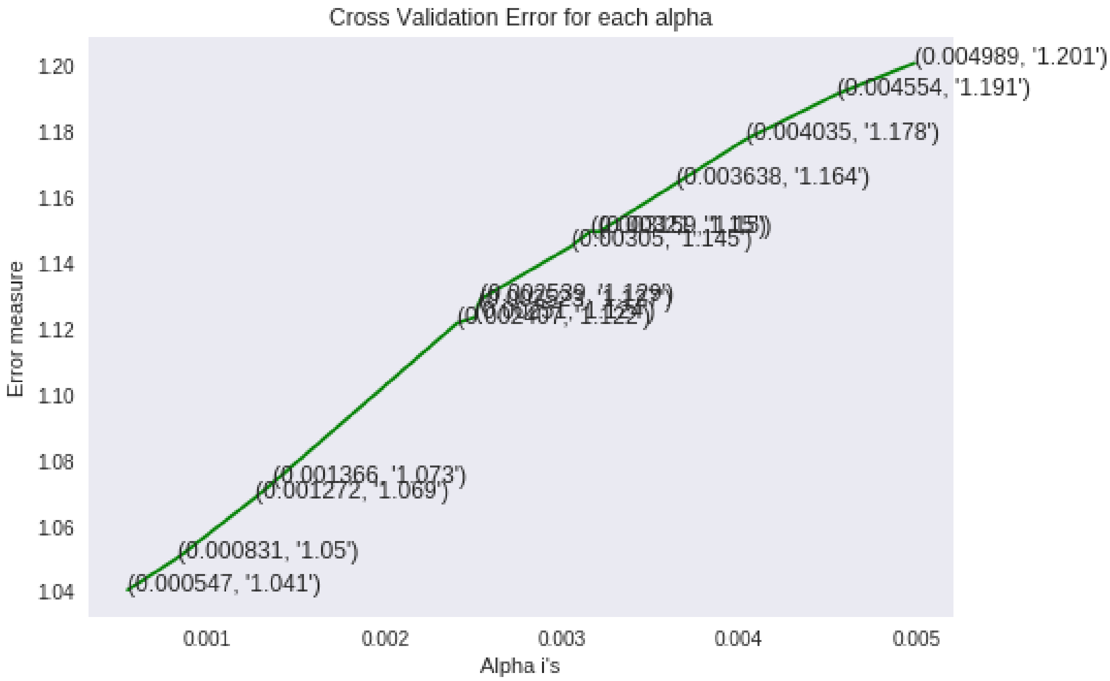


5.0.1 Without class balancing

```
In [29]: alpha = np.random.uniform(0.0002,0.005,15)
         alpha = np.round(alpha,6)
         alpha.sort()
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             clf = SGDClassifier( alpha=i, penalty='l2', loss='hinge', random_state=42)
             clf.fit(train_x_tfidfCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidfCoding, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
             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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
         clf.fit(train_x_tfidfCoding, train_y)
```

```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

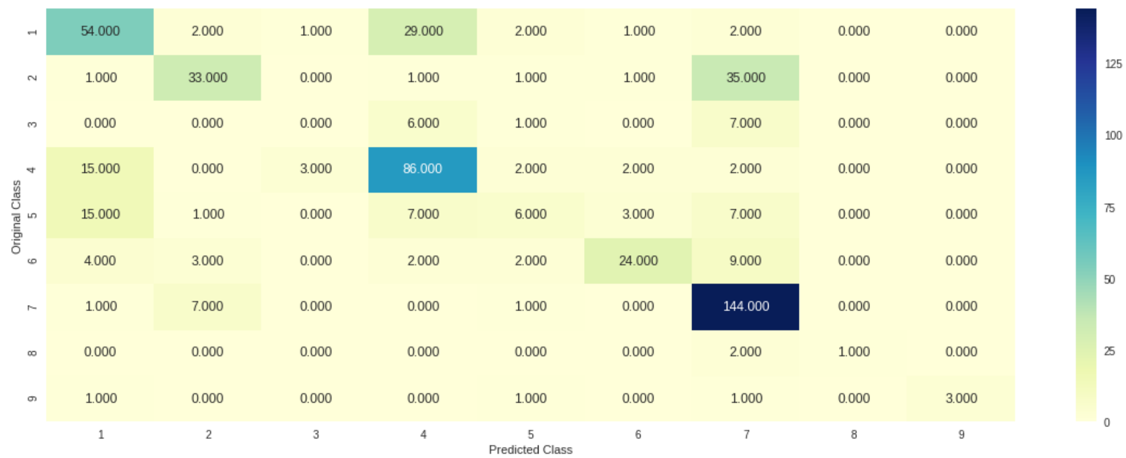
```
for alpha = 0.000547
Log Loss : 1.0405157479414953
for alpha = 0.000831
Log Loss : 1.0503616157096207
for alpha = 0.001272
Log Loss : 1.0688360722037271
for alpha = 0.001366
Log Loss : 1.0730691051632633
for alpha = 0.002407
Log Loss : 1.1216976529516471
for alpha = 0.00251
Log Loss : 1.1235786528866658
for alpha = 0.002523
Log Loss : 1.127000062772096
for alpha = 0.002539
Log Loss : 1.1290274670826514
for alpha = 0.00305
Log Loss : 1.1450333201842517
for alpha = 0.003159
Log Loss : 1.1496900671496142
for alpha = 0.00321
Log Loss : 1.1495704737244987
for alpha = 0.003638
Log Loss : 1.1640338750481505
for alpha = 0.004035
Log Loss : 1.177596743681733
for alpha = 0.004554
Log Loss : 1.1914919649297913
for alpha = 0.004989
Log Loss : 1.2008465989063246
```




```
For values of best alpha =  0.000547 The train log loss is: 0.45957059243562226
For values of best alpha =  0.000547 The cross validation log loss is: 1.0405157479414953
For values of best alpha =  0.000547 The test log loss is: 1.0130955680874625
```

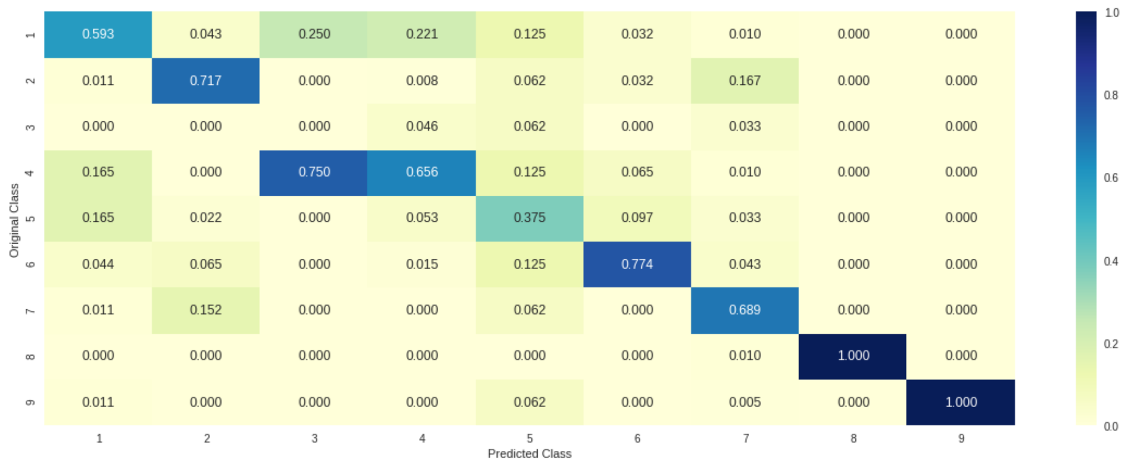
```
In [30]: ##testing
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

```
Log loss : 1.0405157479414953
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```

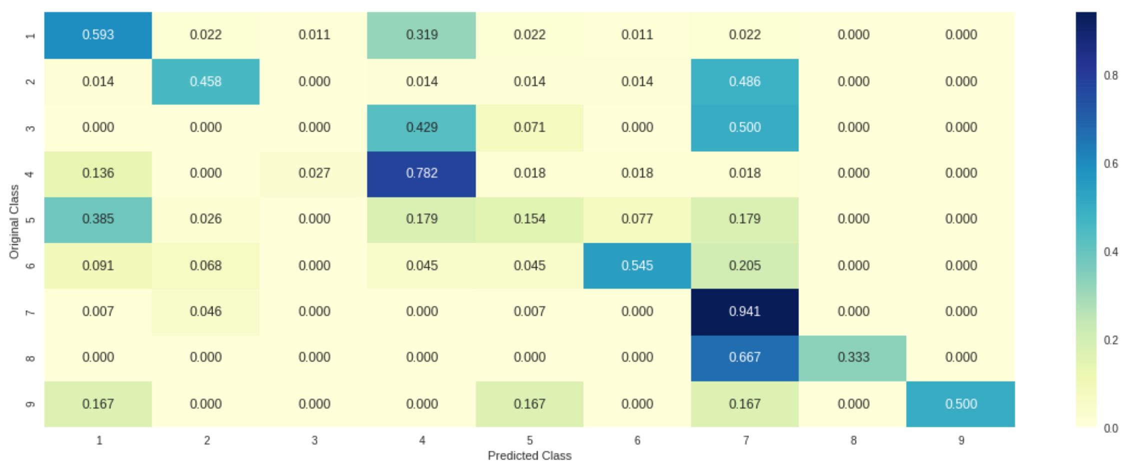


	1	2	3	4	5	6	7	8	9
1	54.000	2.000	1.000	29.000	2.000	1.000	2.000	0.000	0.000
2	1.000	33.000	0.000	1.000	1.000	1.000	35.000	0.000	0.000
3	0.000	0.000	0.000	6.000	1.000	0.000	7.000	0.000	0.000
4	15.000	0.000	3.000	86.000	2.000	2.000	2.000	0.000	0.000
5	15.000	1.000	0.000	7.000	6.000	3.000	7.000	0.000	0.000
6	4.000	3.000	0.000	2.000	2.000	24.000	9.000	0.000	0.000
7	1.000	7.000	0.000	0.000	1.000	0.000	144.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000	0.000
9	1.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	3.000

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




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



6 Feature importance

In [31]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 1
```

```
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCo
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: [[0.0508 0.1205 0.0223 0.0752 0.0526 0.0183 0.6548 0.0043 0.001
Actual Class : 2
--------------------------------------------------
2 Text feature [cells] present in test data point [True]
204 Text feature [activation] present in test data point [True]
220 Text feature [thyroid] present in test data point [True]
225 Text feature [mutant] present in test data point [True]
233 Text feature [ns] present in test data point [True]
243 Text feature [of] present in test data point [True]
424 Text feature [mutants] present in test data point [True]
425 Text feature [cell] present in test data point [True]
435 Text feature [at] present in test data point [True]
439 Text feature [insertion] present in test data point [True]
447 Text feature [codon] present in test data point [True]
465 Text feature [the] present in test data point [True]
471 Text feature [expressing] present in test data point [True]
Out of the top  500  features  13 are present in query point
```

In [32]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 26
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCo
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 : 1
Predicted Class Probabilities: [[0.4584 0.0323 0.0208 0.3907 0.0227 0.0219 0.0481 0.0039 0.001
Actual Class : 1
--------------------------------------------------
302 Text feature [transcriptional] present in test data point [True]
362 Text feature [binding] present in test data point [True]
368 Text feature [function] present in test data point [True]
```

```
371 Text feature [type] present in test data point [True]
412 Text feature [et] present in test data point [True]
493 Text feature [each] present in test data point [True]
498 Text feature [transcription] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

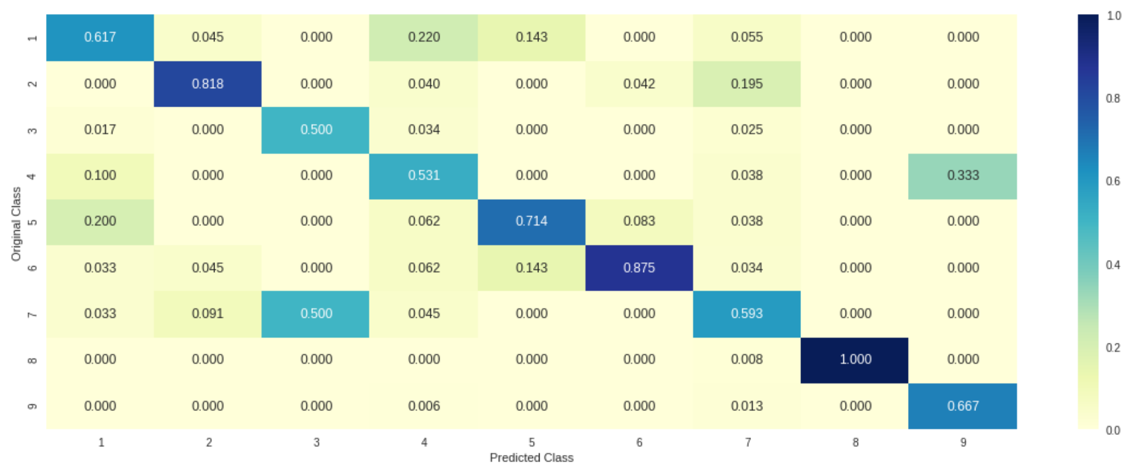
7 Random Forest Classifier

```
In [0]: alpha = [100,200,500,1000,2000]
        max_depth = [5,10,20]
        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_tfidfCoding, train_y)
                sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                sig_clf.fit(train_x_tfidfCoding, train_y)
                sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
                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))

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

        best_alpha = np.argmin(cv_log_error_array)
        clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/3)], criterion='gini', 
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)

        predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/3)],'depth = ',alpha[int
        predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/3)],'depth = ',alpha[int
        predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/3)],'depth = ',alpha[int
```

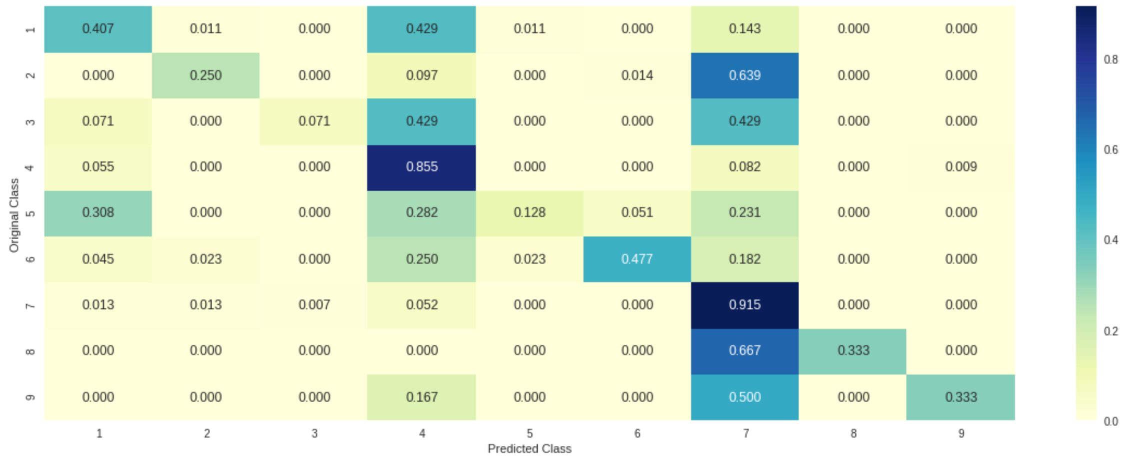
```
In [34]: #test
         clf = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=max_depth[
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

```
Log loss : 1.1721616774021122
Number of mis-classified points : 0.40037593984962405
-------------------- Confusion matrix --------------------
```

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



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




```
In [0]: clf = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=20, random_s
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)

        test_point_index = 1
        no_feature = 100
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.feature_importances_)
        print("-"*50)
        get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_
```

```
In [0]: test_point_index = 29
        no_feature = 100
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.feature_importances_)
        print("-"*50)
        get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_
```

8 Stacking the models

```
In [36]: clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', class_weight='balanced',
         clf1.fit(train_x_tfidfCoding, train_y)
         sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")
```

```
clf2 = SGDClassifier(alpha=0.0003, penalty='l2', loss='hinge', class_weight='balanced
clf2.fit(train_x_tfidfCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = KNeighborsClassifier(n_neighbors=5)
clf3.fit(train_x_tfidfCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_tfidfCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_prob
sig_clf2.fit(train_x_tfidfCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_
sig_clf3.fit(train_x_tfidfCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_t
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classi
    sclf.fit(train_x_tfidfCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 0.98
Support vector machines : Log Loss: 1.03
Naive Bayes : Log Loss: 1.15
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.174
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.001
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.411
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.026
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.128
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.508
```

In [37]:
```
alpha = np.random.uniform(0.005,0.5,10)
alpha = np.round(alpha,5)
alpha.sort()
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classi
    sclf.fit(train_x_tfidfCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_
```

```
            log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
            if best_alpha > log_error:
                best_alpha = log_error
```

```
Stacking Classifer : for the value of alpha: 0.096710 Log Loss: 1.027
Stacking Classifer : for the value of alpha: 0.102950 Log Loss: 1.024
Stacking Classifer : for the value of alpha: 0.135230 Log Loss: 1.016
Stacking Classifer : for the value of alpha: 0.178490 Log Loss: 1.014
Stacking Classifer : for the value of alpha: 0.212420 Log Loss: 1.016
Stacking Classifer : for the value of alpha: 0.216030 Log Loss: 1.016
Stacking Classifer : for the value of alpha: 0.216900 Log Loss: 1.016
Stacking Classifer : for the value of alpha: 0.229120 Log Loss: 1.017
Stacking Classifer : for the value of alpha: 0.320480 Log Loss: 1.030
Stacking Classifer : for the value of alpha: 0.369170 Log Loss: 1.037
```

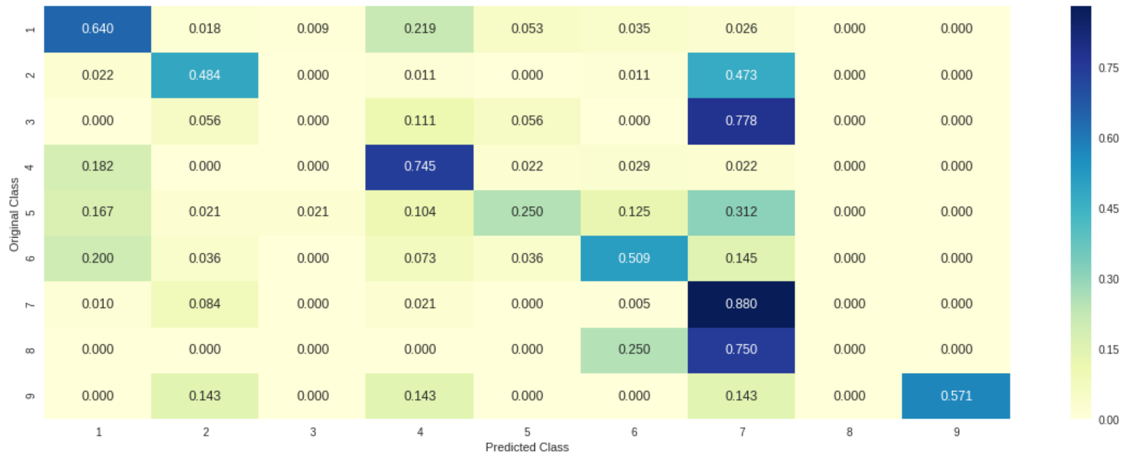
In [39]:

```
#testing
lr = LogisticRegression(C=0.178490)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=
sclf.fit(train_x_tfidfCoding, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_tfidf
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_tfidfCoding))
```

```
Log loss (train) on the stacking classifier : 0.3022920079959014
Log loss (CV) on the stacking classifier : 1.0135557985568862
Log loss (test) on the stacking classifier : 1.0271289216801116
Number of missclassified point : 0.3518796992481203
-------------------- Confusion matrix --------------------
```

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

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

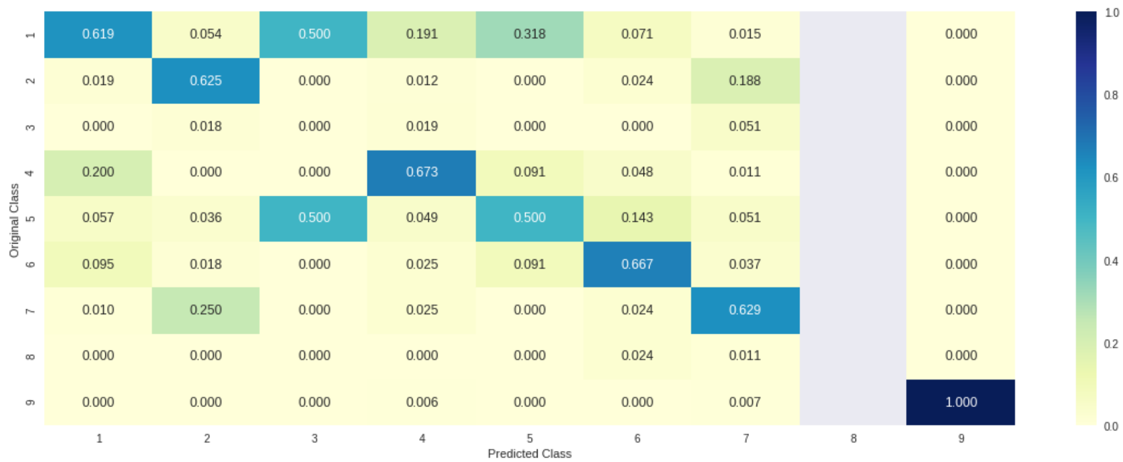


8.1 Maximum Voting Classifier

In [40]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassi
from sklearn.ensemble import VotingClassifier
clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', class_weight='balanced',
clf1.fit(train_x_tfidfCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.0003, penalty='l2', loss='hinge', class_weight='balanced
clf2.fit(train_x_tfidfCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = KNeighborsClassifier(n_neighbors=5)
clf3.fit(train_x_tfidfCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_c
vclf.fit(train_x_tfidfCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_pro
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfidf
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidfCoding))
```

```
Log loss (train) on the VotingClassifier : 0.5650012605803157
Log loss (CV) on the VotingClassifier : 1.0251590276774352
Log loss (test) on the VotingClassifier : 1.013884876965482
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```

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



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

9 Conclusion

	Train loss	CV loss	Test loss
Naive-Bayes	0.600	1.233	1.103
K-NN	0.876	1.216	1.127
Logistic Regression(With Class Balancing)	0.406	1.093	0.949
Logistic Regression(Without Class Balancing)	0.440	1.099	0.966
Linear SVM(With Class Balancing)	0.493	1.072	1.046
Linear SVM(Without Class Balancing)	0.459	1.040	1.013
Random Forest	0.503	1.068	1.115