Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25

2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

ID,Gene,Variation,Class
0,FAM58A,Truncating Mutations,1
1,CBL,W802*,2
2,CBL,Q249E,2
...

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y

ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

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 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.model_selection 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
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/externals/six.py:31: Dep
recationWarning: The module is deprecated in version 0.21 and will be r
emoved in version 0.23 since we've dropped support for Python 2.7. Plea
se rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

```
Number of data points :  3321
Number of features :  4
```

```
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

Out[2]:

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

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

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

3.1.2. Reading Text Data

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

```
Number of data points :  3321
Number of features :  2
```

```
Features :  ['ID' 'TEXT']
```

Out[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...

3.1.3. Preprocessing of text

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 t
he data
            if not word in stop_words:
                string += word + " "
```

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

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_tim
e, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 161.10056 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	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

In [7]:
```
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

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

In [10]:
```
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 o
f output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
```

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

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

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
])
```

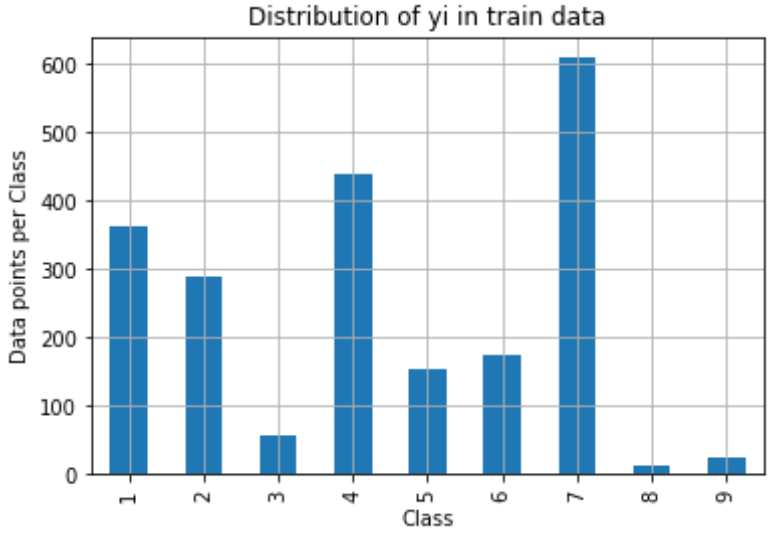
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

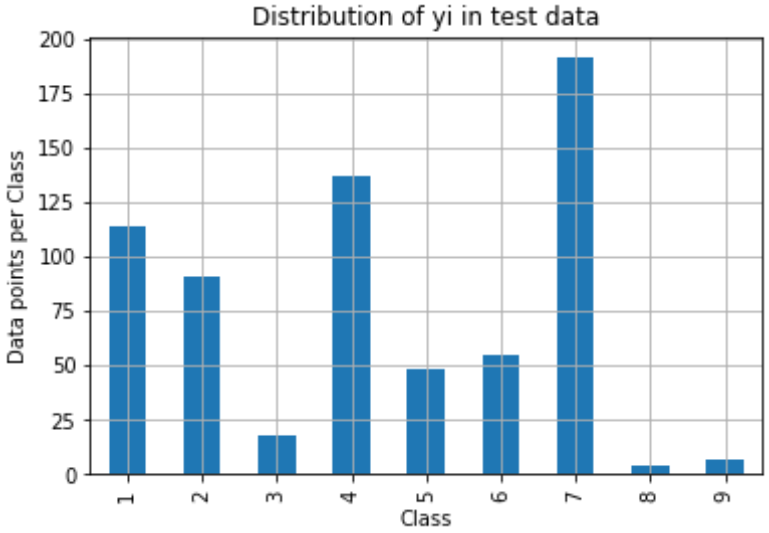
In [12]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index
()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
```

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


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

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

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
```

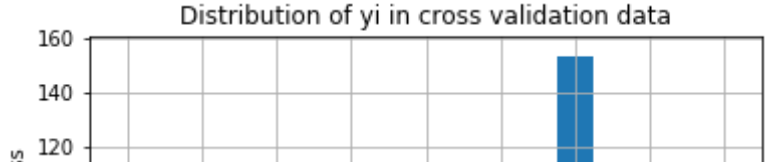
```
# -(train_class_distribution.values): the minus sign will give us in de
creasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribut
ion.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.sha
pe[0]*100), 3), '%)')
```

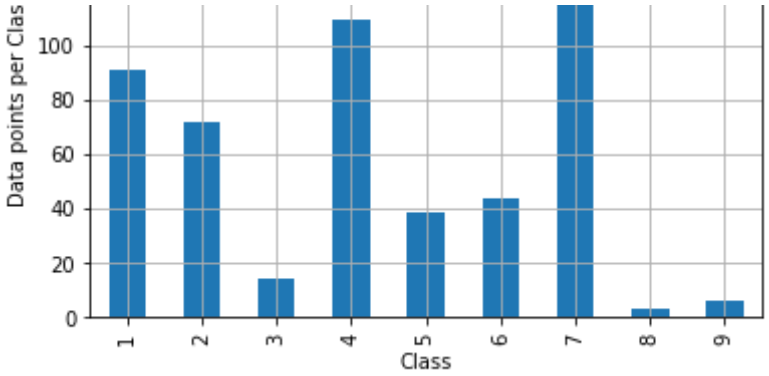



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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




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






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

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [13]:
```
# 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 cl
```

```
ass i are predicted class j

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 correspo
nds to rows in two diamensional array
    # 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 element
s in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
nds to rows in two diamensional array
    # 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=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
```

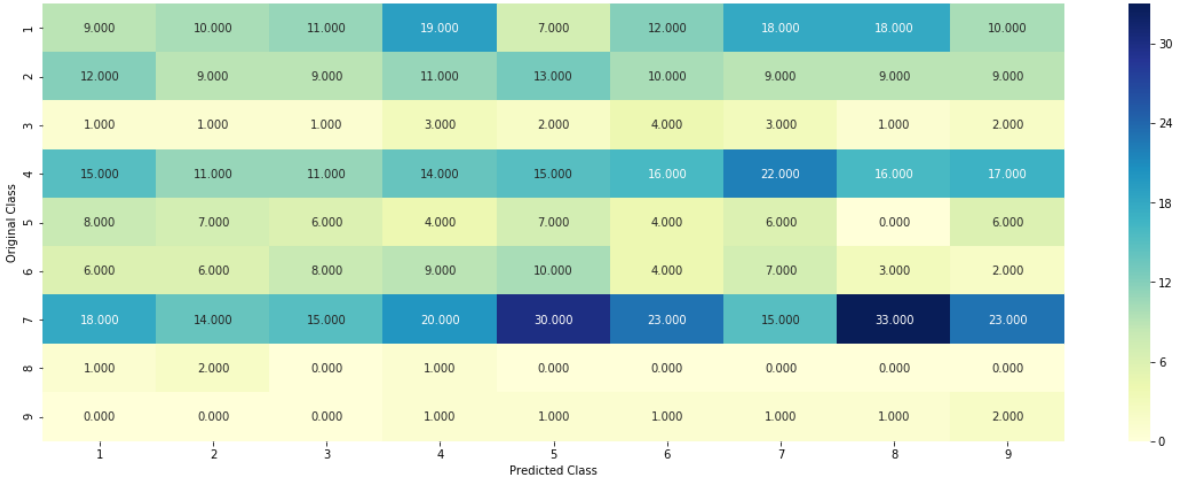
```
    plt.show()

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

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

In [14]:

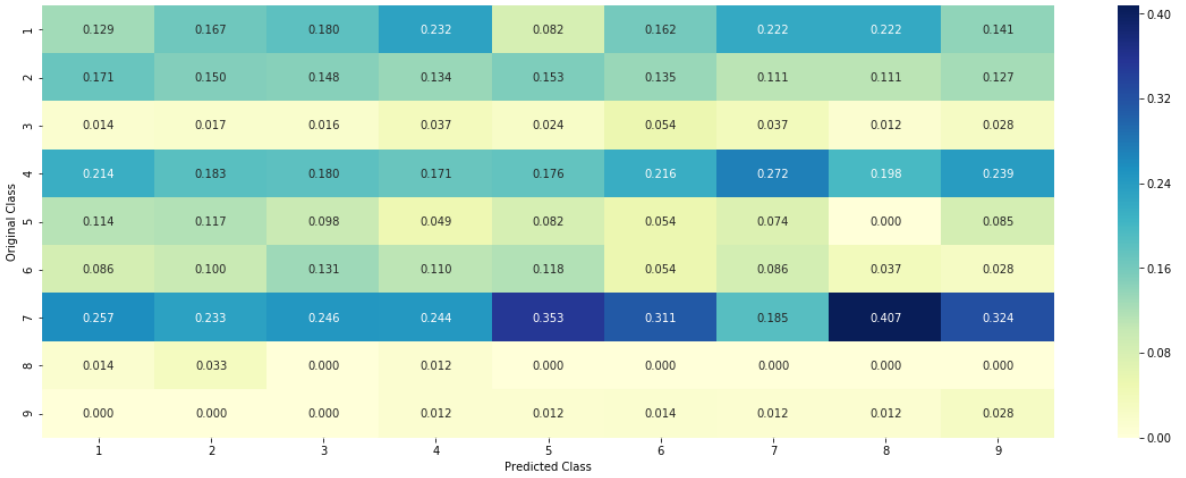
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y
_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
```

```
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_p
redicted_y, eps=1e-15))

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

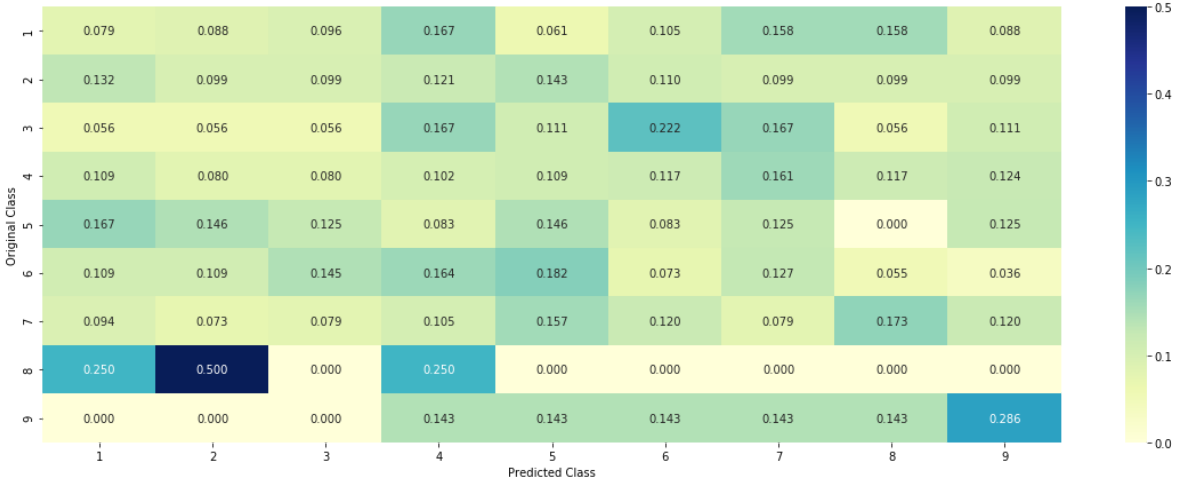
```
Log loss on Cross Validation Data using Random Model 2.50201599110303
Log loss on Test Data using Random Model 2.520887788013496
-------------------- Confusion matrix --------------------
```




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



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



3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
```

```
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_f
ea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
```

```
    # Amplification                             43
    # Fusions                                   22
    # Overexpression                             3
    # E17K                                       3
    # Q61L                                       3
    # S222D                                      2
    # P130S                                      2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

            # cls_cnt.shape[0](numerator) will contain the number of ti
```

```
me that particular feature occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90
*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.06818181818181818177, 0.06818181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.0606060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
    #      ...
```

```
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- (numerator + 10*alpha) / (denominator + 90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
```

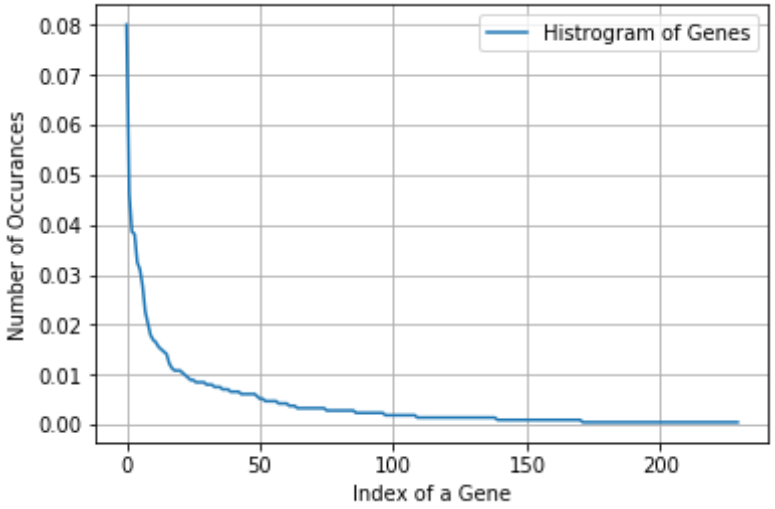
```
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 230
BRCA1     170
TP53       98
PTEN       82
EGFR       81
BRCA2      69
KIT        66
BRAF       59
ERBB2      48
ALK        43
CDKN2A     38
Name: Gene, dtype: int64
```

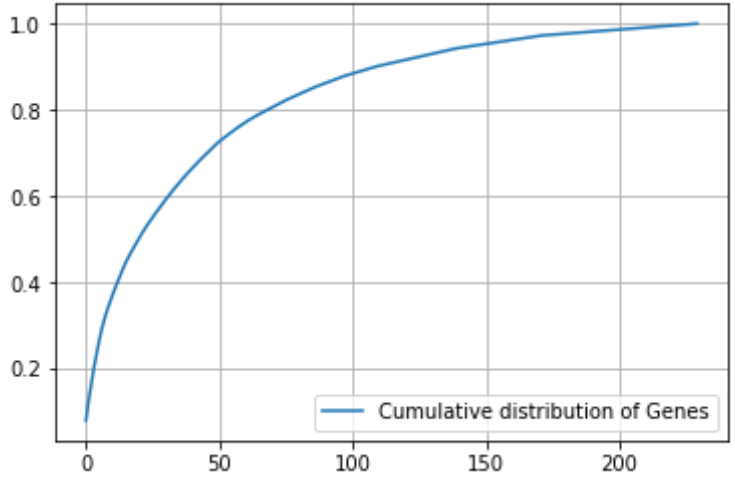
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

```
Ans: There are 230 different categories of genes in the train data, and
they are distibuted as follows
```

In [18]:
```
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 [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
```

```
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
'''gene_vectorizer_ngram = CountVectorizer(ngram_range=(1,2))
train_gene_feature_ngram = gene_vectorizer_ngram.fit_transform(train_df
['Gene'])
test_gene_feature_ngram = gene_vectorizer_ngram.transform(test_df['Gen
e'])
cv_gene_feature_ngram = gene_vectorizer_ngram.transform(cv_df['Gen
e'])'''
```

Out[23]:
```
"gene_vectorizer_ngram = CountVectorizer(ngram_range=(1,2))\ntrain_gene
_feature_ngram = gene_vectorizer_ngram.fit_transform(train_df['Gene'])
\ntest_gene_feature_ngram = gene_vectorizer_ngram.transform(test_df['Ge
ne'])\ncv_gene_feature_ngram = gene_vectorizer_ngram.transform(cv_df['G
ene'])"
```

In [24]:
```
#TFIDF Vectorization of Gene feature
'''from sklearn.feature_selection import SelectKBest
count_vect=TfidfVectorizer()
train_gene_feature_tfidf = count_vect.fit_transform(train_df['Gene'])
print(train_gene_feature_tfidf.shape[1])
test_gene_feature_tfidf = count_vect.transform(test_df['Gene'])
cv_gene_feature_tfidf = count_vect.transform(cv_df['Gene'])
train_gene_feature_tfidf = SelectKBest(k='all').fit_transform(train_gen
e_feature_tfidf, y_train)
test_gene_feature_tfidf = SelectKBest(k='all').fit_transform(test_gene_
feature_tfidf, y_test)
cv_gene_feature_tfidf = SelectKBest(k='all').fit_transform(cv_gene_feat
ure_tfidf, y_cv)'''
```

Out[24]:
```
"from sklearn.feature_selection import SelectKBest\ncount_vect=TfidfVec
torizer()\ntrain_gene_feature_tfidf = count_vect.fit_transform(train_df
['Gene'])\nprint(train_gene_feature_tfidf.shape[1])\ntest_gene_feature_
tfidf = count_vect.transform(test_df['Gene'])\ncv_gene_feature_tfidf =
count_vect.transform(cv_df['Gene'])\ntrain_gene_feature_tfidf = SelectK
Best(k='all').fit_transform(train_gene_feature_tfidf, y_train)\ntest_ge
ne_feature_tfidf = SelectKBest(k='all').fit_transform(test_gene_feature
```

```
_tfidf, y_test)\ncv_gene_feature_tfidf = SelectKBest(k='all').fit_trans
form(cv_gene_feature_tfidf, y_cv)"
```

In [25]:
```
train_df['Gene'].head()
```

Out[25]:
```
2836    BRCA2
1319     MLH1
1287     HRAS
2188     PTEN
2441    BRCA1
Name: Gene, dtype: object
```

In [26]:
```
gene_vectorizer.get_feature_names()
```

Out[26]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid2',
 'arid5b',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
```

```
                'bcl2l11',
                'bcor',
                'braf',
                'brca1',
                'brca2',
                'brd4',
                'brip1',
                'btk',
                'card11',
                'carm1',
                'casp8',
                'cbl',
                'ccnd1',
                'ccnd3',
                'ccne1',
                'cdh1',
                'cdk12',
                'cdk4',
                'cdk6',
                'cdkn1a',
                'cdkn1b',
                'cdkn2a',
                'cdkn2b',
                'cdkn2c',
                'cebpa',
                'chek2',
                'cic',
                'crebbp',
                'ctcf',
                'ctnnb1',
                'ddr2',
                'dicer1',
                'dusp4',
                'egfr',
                'eif1ax',
                'elf3',
                'ep300',
                'epas1',
                'erbb2',
```

```
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fanca',
                'fat1',
                'fbxw7',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxo1',
                'foxp1',
                'fubp1',
                'gna11',
                'gnaq',
                'gnas',
                'h3f3a',
                'hist1h1c',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikbke',
                'ikzf1',
                'il7r',
                'jak1',
```

```
                'jak2',
                'jun',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
                'kmt2b',
                'kmt2c',
                'kmt2d',
                'knstrn',
                'kras',
                'lats1',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'mdm2',
                'med12',
                'mef2b',
                'men1',
                'met',
                'mga',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'myod1',
                'ncor1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
```

```
                'notch1',
                'notch2',
                'npm1',
                'nras',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pax8',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pim1',
                'pms1',
                'pms2',
                'pole',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
                'rab35',
                'rac1',
                'rad21',
                'rad50',
                'rad51b',
                'rad51c',
                'raf1',
                'rara',
                'rasa1',
```

```
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rit1',
                'rnf43',
                'ros1',
                'rras2',
                'runx1',
                'rxra',
                'rybp',
                'sdhb',
                'sdhc',
                'setd2',
                'sf3b1',
                'shoc2',
                'smad2',
                'smad3',
                'smad4',
                'smarca4',
                'smarcb1',
                'smo',
                'sos1',
                'sox9',
                'spop',
                'src',
                'srsf2',
                'stag2',
                'stat3',
                'stk11',
                'tcf3',
                'tcf7l2',
                'tert',
                'tet1',
                'tet2',
                'tgfbr1',
                'tgfbr2',
                'tmprss2',
```

```
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [27]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 230)
```

In [28]:
```
'''print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_tfidf.shape)'''
```

Out[28]:
```
'print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_tfidf.shape)'
```

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

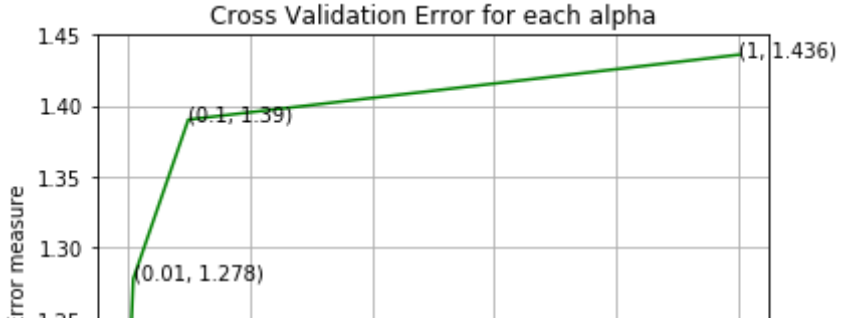
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

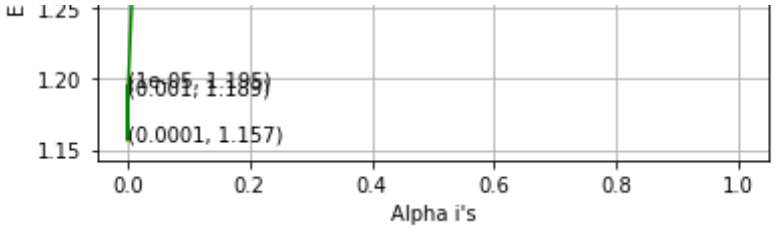
In [29]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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

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


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

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

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

```
For values of alpha =  1e-05 The log loss is: 1.194792632767055
For values of alpha =  0.0001 The log loss is: 1.156619923667089
For values of alpha =  0.001 The log loss is: 1.1888528054321927
For values of alpha =  0.01 The log loss is: 1.2781844317845446
For values of alpha =  0.1 The log loss is: 1.3901950872115807
For values of alpha =  1 The log loss is: 1.4358437313931807
```






```
For values of best alpha =  0.0001 The train log loss is: 1.00582265867
34222
For values of best alpha =  0.0001 The cross validation log loss is: 1.
156619923667089
For values of best alpha =  0.0001 The test log loss is: 1.201383821365
866
```

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

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

In [30]:
```
print("Q6. How many data points in Test and CV datasets are covered by
 the ", unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  23
0  genes in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
```

```
2. In cross validation data 511 out of  532 : 96.05263157894737
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [31]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

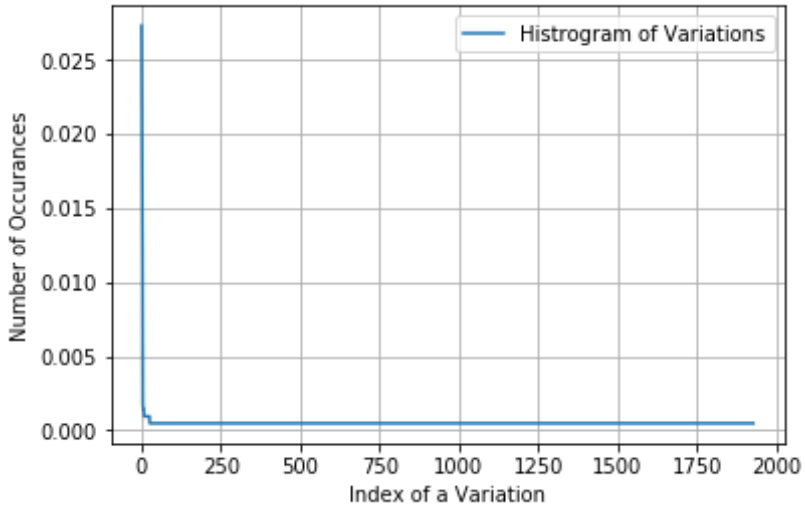
```
Number of Unique Variations : 1928
Truncating_Mutations         58
Deletion                     48
Amplification                42
Fusions                      25
Overexpression                4
E17K                          3
G12V                          3
T58I                          3
TMPRSS2-ETV1_Fusion           2
Promoter_Hypermethylation     2
Name: Variation, dtype: int64
```

In [32]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

```
Ans: There are 1928 different categories of variations in the train dat
a, and they are distibuted as follows
```

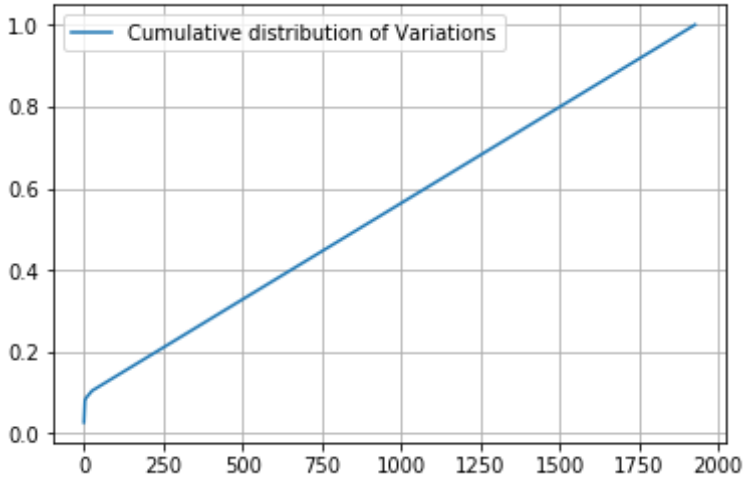
In [33]:
```
s = sum(unique_variations.values);
```

```
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 [34]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02730697 0.04990584 0.06967985 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [35]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [36]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [37]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [38]:
```
'''variation_vectorizer_ngram = CountVectorizer(ngram_range=(1,2))
train_variation_feature_ngram = variation_vectorizer_ngram.fit_transfor
m(train_df['Variation'])
test_variation_feature_ngram = variation_vectorizer_ngram.transform(tes
t_df['Variation'])
cv_variation_feature_ngram = variation_vectorizer_ngram.transform(cv_df
['Variation'])'''
```

Out[38]:
```
"variation_vectorizer_ngram = CountVectorizer(ngram_range=(1,2))\ntrain
_variation_feature_ngram = variation_vectorizer_ngram.fit_transform(tra
in_df['Variation'])\ntest_variation_feature_ngram = variation_vectorize
r_ngram.transform(test_df['Variation'])\ncv_variation_feature_ngram = v
ariation_vectorizer_ngram.transform(cv_df['Variation'])"
```

In [39]:
```
'''count_vect_variation=TfidfVectorizer()
train_variation_feature_tfidf = count_vect_variation.fit_transform(trai
n_df['Variation'])
print(train_variation_feature_tfidf.shape[1])
test_variation_feature_tfidf = count_vect_variation.transform(test_df
['Variation'])
cv_variation_feature_tfidf = count_vect_variation.transform(cv_df['Vari
ation'])
```

```
train_variation_feature_tfidf = SelectKBest(k=1000).fit_transform(train
_variation_feature_tfidf, y_train)
test_variation_feature_tfidf = SelectKBest(k=1000).fit_transform(test_v
ariation_feature_tfidf, y_test)
cv_variation_feature_tfidf = SelectKBest(k=1000).fit_transform(cv_varia
tion_feature_tfidf, y_cv)'''
```

Out[39]:
```
"count_vect_variation=TfidfVectorizer()\ntrain_variation_feature_tfidf
= count_vect_variation.fit_transform(train_df['Variation'])\nprint(trai
n_variation_feature_tfidf.shape[1])\ntest_variation_feature_tfidf = cou
nt_vect_variation.transform(test_df['Variation'])\ncv_variation_feature
_tfidf = count_vect_variation.transform(cv_df['Variation'])\ntrain_vari
ation_feature_tfidf = SelectKBest(k=1000).fit_transform(train_variation
_feature_tfidf, y_train)\ntest_variation_feature_tfidf = SelectKBest(k=
1000).fit_transform(test_variation_feature_tfidf, y_test)\ncv_variation
_feature_tfidf = SelectKBest(k=1000).fit_transform(cv_variation_feature
_tfidf, y_cv)"
```

In [40]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature: (2124, 1967)
```

In [41]:
```
'''print("train_variation_feature_onehotEncoded is converted feature us
ing the onne-hot encoding method. The shape of Variation feature:", tra
in_variation_feature_tfidf.shape)'''
```

Out[41]:
```
'print("train_variation_feature_onehotEncoded is converted feature usin
g the onne-hot encoding method. The shape of Variation feature:", train
_variation_feature_tfidf.shape)'
```

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

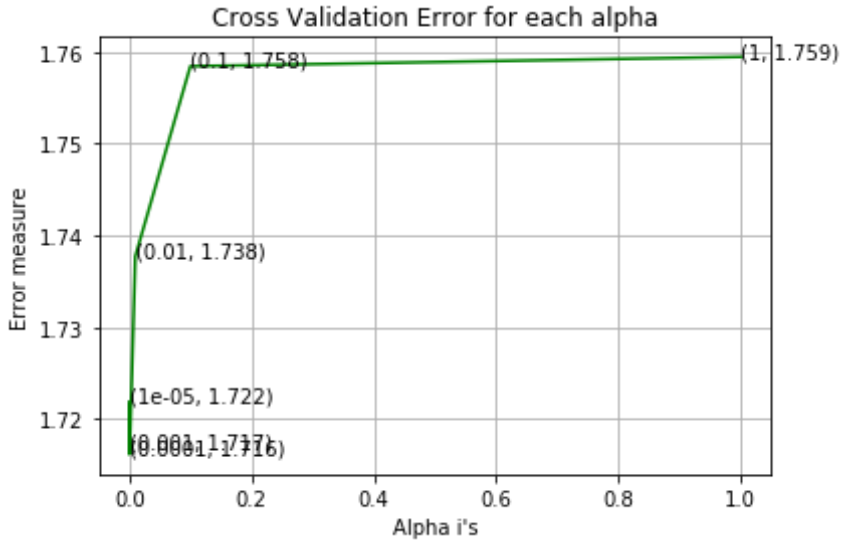
Let's build a model just like the earlier!

In [42]:
```
alpha = [10 ** x for x in range(-5, 1)]
```

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

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

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


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

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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7218411743722626
For values of alpha =  0.0001 The log loss is: 1.7161288571158513
For values of alpha =  0.001 The log loss is: 1.7168367695592122
For values of alpha =  0.01 The log loss is: 1.7377242628844551
For values of alpha =  0.1 The log loss is: 1.7584582110192342
For values of alpha =  1 The log loss is: 1.7594819481408266
```




```
For values of best alpha =  0.0001 The train log loss is: 0.73204815707
56048
For values of best alpha =  0.0001 The cross validation log loss is: 1.
7161288571158513
For values of best alpha =  0.0001 The test log loss is: 1.705107484615
9038
```

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

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

In [43]:
```
print("Q12. How many data points are covered by total ", unique_variati
```

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

```
Q12. How many data points are covered by total  1928  genes in test and
cross validation data sets?
Ans
1. In test data 58 out of 665 : 8.721804511278195
2. In cross validation data 66 out of  532 : 12.406015037593985
```

3.2.3 Univariate Analysis on Text Feature

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

In [44]:
```
# 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 [45]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [46]:
```
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

```
Total number of unique words in train data : 52806
```

In [47]:
```
text_vectorizer_ngram = CountVectorizer(ngram_range=(1,4),min_df=3,max_
features=2000)
train_text_feature_ngram = text_vectorizer_ngram.fit_transform(train_df
['TEXT'])
cv_text_feature_ngram = text_vectorizer_ngram.transform(cv_df['TEXT'])
test_text_feature_ngram = text_vectorizer_ngram.transform(test_df['TEX
T'])
```

In [48]:
```
from sklearn.feature_selection import SelectKBest
count_vect_text=TfidfVectorizer(ngram_range=(1,4),min_df=5,max_features
=2000)
train_text_feature_tfidf = count_vect_text.fit_transform(train_df['TEX
T'])
print(train_text_feature_tfidf.shape[1])
test_text_feature_tfidf = count_vect_text.transform(test_df['TEXT'])
cv_text_feature_tfidf = count_vect_text.transform(cv_df['TEXT'])
#train_text_feature_tfidf = SelectKBest(k=1000).fit_transform(train_tex
t_feature_tfidf, y_train)
#test_text_feature_tfidf = SelectKBest(k=1000).transform(test_text_feat
ure_tfidf)
#cv_text_feature_tfidf = SelectKBest(k=1000).transform(cv_text_feature_
tfidf)
```

```
2000
```

In [49]:
```
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 [50]:
```
#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 [51]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [52]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)
train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)
# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEX
T'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCodi
ng, axis=0)
test_text_feature_tfidf= normalize(test_text_feature_tfidf, 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)
cv_text_feature_tfidf = normalize(cv_text_feature_tfidf, axis=0)
```

In [53]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [54]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 5300, 4: 3474, 6: 2936, 5: 2792, 7: 2220, 8: 1982, 9: 1631,
10: 1365, 12: 1314, 11: 1129, 15: 1003, 16: 973, 14: 947, 13: 733, 24:
700, 18: 664, 21: 584, 17: 546, 20: 530, 19: 456, 22: 435, 30: 416, 28:
412, 39: 365, 23: 355, 25: 353, 27: 339, 32: 319, 26: 299, 29: 278, 35:
275, 51: 274, 36: 270, 33: 270, 31: 261, 34: 225, 42: 224, 48: 221, 40:
220, 41: 194, 47: 188, 37: 183, 38: 178, 45: 175, 44: 170, 50: 165, 52:
161, 43: 157, 49: 156, 56: 151, 60: 149, 46: 147, 59: 142, 55: 141, 54:
141, 53: 132, 57: 127, 66: 115, 72: 114, 67: 111, 61: 111, 58: 109, 62:
108, 64: 107, 63: 104, 69: 103, 78: 97, 73: 94, 68: 94, 70: 90, 96: 88,
65: 88, 77: 86, 75: 85, 94: 82, 71: 82, 81: 77, 86: 75, 85: 75, 76: 75,
84: 73, 80: 72, 88: 71, 74: 68, 99: 65, 89: 64, 87: 63, 95: 62, 79: 62,
104: 60, 98: 60, 91: 60, 100: 57, 83: 57, 82: 57, 128: 55, 110: 54, 12
0: 53, 102: 53, 112: 52, 90: 52, 116: 50, 103: 50, 139: 49, 101: 49, 9
7: 49, 92: 49, 119: 47, 105: 47, 93: 47, 126: 46, 117: 46, 114: 46, 10
9: 45, 108: 45, 107: 45, 113: 44, 106: 44, 115: 42, 137: 41, 144: 40, 1
41: 40, 127: 40, 111: 40, 125: 39, 124: 39, 123: 38, 153: 37, 136: 37,
134: 37, 133: 37, 165: 36, 135: 36, 154: 34, 132: 34, 122: 34, 118: 34,
156: 33, 140: 33, 131: 33, 172: 32, 129: 32, 121: 32, 170: 31, 166: 31,
160: 31, 147: 31, 130: 31, 143: 30, 138: 29, 210: 28, 174: 28, 151: 28,
159: 27, 152: 27, 149: 27, 148: 27, 145: 27, 192: 26, 185: 26, 183: 26,
163: 26, 161: 26, 142: 26, 209: 25, 194: 25, 175: 25, 173: 25, 164: 25,
150: 25, 146: 25, 280: 24, 208: 24, 157: 24, 237: 23, 236: 23, 197: 23,
193: 23, 178: 23, 155: 23, 196: 22, 195: 22, 190: 22, 189: 22, 180: 22,
167: 22, 223: 21, 198: 21, 191: 21, 171: 21, 169: 21, 278: 20, 230: 20,
212: 20, 201: 20, 181: 20, 179: 20, 244: 19, 219: 19, 204: 19, 184: 19,
266: 18, 229: 18, 205: 18, 202: 18, 199: 18, 188: 18, 187: 18, 177: 18,
162: 18, 287: 17, 250: 17, 240: 17, 232: 17, 225: 17, 224: 17, 221: 17,
220: 17, 218: 17, 211: 17, 182: 17, 158: 17, 394: 16, 345: 16, 288: 16,
```

276: 16, 269: 16, 256: 16, 235: 16, 234: 16, 228: 16, 222: 16, 215: 16,
203: 16, 200: 16, 168: 16, 332: 15, 296: 15, 295: 15, 291: 15, 284: 15,
260: 15, 258: 15, 249: 15, 245: 15, 226: 15, 217: 15, 207: 15, 176: 15,
325: 14, 267: 14, 265: 14, 264: 14, 242: 14, 241: 14, 238: 14, 317: 13,
315: 13, 312: 13, 300: 13, 289: 13, 277: 13, 262: 13, 248: 13, 239: 13,
214: 13, 186: 13, 392: 12, 372: 12, 336: 12, 334: 12, 327: 12, 311: 12,
310: 12, 302: 12, 299: 12, 297: 12, 290: 12, 286: 12, 282: 12, 279: 12,
275: 12, 255: 12, 253: 12, 233: 12, 231: 12, 213: 12, 206: 12, 440: 11,
396: 11, 369: 11, 362: 11, 361: 11, 356: 11, 355: 11, 341: 11, 324: 11,
322: 11, 313: 11, 308: 11, 303: 11, 293: 11, 259: 11, 254: 11, 252: 11,
246: 11, 216: 11, 473: 10, 407: 10, 402: 10, 358: 10, 353: 10, 344: 10,
343: 10, 339: 10, 331: 10, 321: 10, 318: 10, 309: 10, 274: 10, 272: 10,
247: 10, 243: 10, 455: 9, 451: 9, 438: 9, 435: 9, 432: 9, 419: 9, 408:
9, 373: 9, 357: 9, 351: 9, 349: 9, 346: 9, 337: 9, 326: 9, 323: 9, 316:
9, 314: 9, 285: 9, 283: 9, 281: 9, 273: 9, 271: 9, 268: 9, 263: 9, 251:
9, 539: 8, 502: 8, 481: 8, 478: 8, 460: 8, 457: 8, 437: 8, 436: 8, 431:
8, 429: 8, 412: 8, 404: 8, 399: 8, 388: 8, 376: 8, 375: 8, 354: 8, 350:
8, 342: 8, 329: 8, 319: 8, 306: 8, 261: 8, 227: 8, 944: 7, 675: 7, 674:
7, 642: 7, 632: 7, 628: 7, 619: 7, 510: 7, 506: 7, 468: 7, 454: 7, 449:
7, 445: 7, 421: 7, 415: 7, 414: 7, 413: 7, 410: 7, 397: 7, 384: 7, 381:
7, 379: 7, 370: 7, 363: 7, 352: 7, 348: 7, 338: 7, 330: 7, 307: 7, 305:
7, 304: 7, 298: 7, 294: 7, 270: 7, 959: 6, 886: 6, 830: 6, 824: 6, 739:
6, 714: 6, 653: 6, 650: 6, 638: 6, 636: 6, 630: 6, 599: 6, 587: 6, 582:
6, 576: 6, 575: 6, 570: 6, 543: 6, 538: 6, 537: 6, 531: 6, 520: 6, 519:
6, 515: 6, 505: 6, 499: 6, 498: 6, 497: 6, 496: 6, 494: 6, 485: 6, 479:
6, 471: 6, 465: 6, 462: 6, 453: 6, 447: 6, 444: 6, 423: 6, 418: 6, 416:
6, 409: 6, 405: 6, 401: 6, 386: 6, 385: 6, 383: 6, 382: 6, 380: 6, 374:
6, 366: 6, 333: 6, 328: 6, 320: 6, 292: 6, 257: 6, 1449: 5, 1249: 5, 94
5: 5, 902: 5, 897: 5, 871: 5, 850: 5, 826: 5, 822: 5, 819: 5, 795: 5, 7
81: 5, 773: 5, 771: 5, 766: 5, 741: 5, 726: 5, 698: 5, 673: 5, 664: 5,
639: 5, 622: 5, 620: 5, 601: 5, 597: 5, 585: 5, 580: 5, 571: 5, 568: 5,
563: 5, 560: 5, 556: 5, 554: 5, 550: 5, 549: 5, 545: 5, 544: 5, 534: 5,
529: 5, 525: 5, 523: 5, 514: 5, 492: 5, 482: 5, 476: 5, 475: 5, 474: 5,
469: 5, 466: 5, 461: 5, 452: 5, 450: 5, 446: 5, 442: 5, 433: 5, 425: 5,
424: 5, 411: 5, 403: 5, 395: 5, 393: 5, 390: 5, 387: 5, 371: 5, 365: 5,
359: 5, 347: 5, 340: 5, 2353: 4, 2258: 4, 2135: 4, 1422: 4, 1367: 4, 13
30: 4, 1296: 4, 1285: 4, 1262: 4, 1254: 4, 1230: 4, 1224: 4, 1181: 4, 1
152: 4, 1151: 4, 1124: 4, 1108: 4, 1033: 4, 1029: 4, 998: 4, 996: 4, 97
5: 4, 951: 4, 949: 4, 939: 4, 923: 4, 916: 4, 909: 4, 908: 4, 893: 4, 8

```
81: 4, 872: 4, 870: 4, 854: 4, 847: 4, 840: 4, 838: 4, 835: 4, 816: 4,
807: 4, 805: 4, 799: 4, 786: 4, 782: 4, 780: 4, 777: 4, 761: 4, 756: 4,
751: 4, 746: 4, 728: 4, 727: 4, 712: 4, 711: 4, 708: 4, 705: 4, 703: 4,
702: 4, 697: 4, 691: 4, 690: 4, 672: 4, 666: 4, 665: 4, 663: 4, 662: 4,
659: 4, 654: 4, 644: 4, 643: 4, 634: 4, 627: 4, 625: 4, 614: 4, 611: 4,
610: 4, 609: 4, 607: 4, 602: 4, 600: 4, 598: 4, 596: 4, 594: 4, 591: 4,
581: 4, 578: 4, 574: 4, 565: 4, 558: 4, 557: 4, 546: 4, 542: 4, 541: 4,
533: 4, 532: 4, 528: 4, 527: 4, 522: 4, 521: 4, 518: 4, 517: 4, 516: 4,
512: 4, 511: 4, 509: 4, 508: 4, 507: 4, 504: 4, 495: 4, 490: 4, 489: 4,
487: 4, 484: 4, 464: 4, 458: 4, 443: 4, 441: 4, 434: 4, 430: 4, 428: 4,
427: 4, 422: 4, 406: 4, 400: 4, 389: 4, 368: 4, 367: 4, 335: 4, 301: 4,
3347: 3, 3264: 3, 2647: 3, 2637: 3, 2557: 3, 2550: 3, 2495: 3, 2412: 3,
2390: 3, 2112: 3, 2061: 3, 1988: 3, 1871: 3, 1858: 3, 1818: 3, 1817: 3,
1785: 3, 1775: 3, 1771: 3, 1741: 3, 1733: 3, 1699: 3, 1655: 3, 1608: 3,
1538: 3, 1518: 3, 1514: 3, 1478: 3, 1477: 3, 1475: 3, 1460: 3, 1459: 3,
1440: 3, 1433: 3, 1403: 3, 1387: 3, 1354: 3, 1344: 3, 1313: 3, 1304: 3,
1297: 3, 1292: 3, 1291: 3, 1269: 3, 1261: 3, 1256: 3, 1247: 3, 1239: 3,
1234: 3, 1233: 3, 1229: 3, 1194: 3, 1189: 3, 1187: 3, 1183: 3, 1180: 3,
1175: 3, 1147: 3, 1136: 3, 1132: 3, 1111: 3, 1106: 3, 1096: 3, 1086: 3,
1080: 3, 1077: 3, 1076: 3, 1075: 3, 1063: 3, 1061: 3, 1055: 3, 1052: 3,
1050: 3, 1046: 3, 1044: 3, 1038: 3, 1032: 3, 1023: 3, 1022: 3, 1015: 3,
1006: 3, 1002: 3, 991: 3, 984: 3, 970: 3, 968: 3, 946: 3, 942: 3, 938:
3, 936: 3, 934: 3, 929: 3, 917: 3, 911: 3, 907: 3, 900: 3, 896: 3, 892:
3, 891: 3, 884: 3, 882: 3, 876: 3, 873: 3, 867: 3, 864: 3, 863: 3, 859:
3, 856: 3, 843: 3, 832: 3, 823: 3, 814: 3, 810: 3, 809: 3, 803: 3, 796:
3, 794: 3, 792: 3, 779: 3, 770: 3, 769: 3, 767: 3, 765: 3, 759: 3, 754:
3, 750: 3, 748: 3, 736: 3, 733: 3, 731: 3, 723: 3, 721: 3, 718: 3, 717:
3, 709: 3, 696: 3, 695: 3, 693: 3, 692: 3, 687: 3, 683: 3, 682: 3, 678:
3, 677: 3, 676: 3, 670: 3, 669: 3, 657: 3, 656: 3, 655: 3, 648: 3, 640:
3, 635: 3, 626: 3, 624: 3, 621: 3, 617: 3, 615: 3, 606: 3, 605: 3, 603:
3, 593: 3, 586: 3, 584: 3, 573: 3, 569: 3, 566: 3, 562: 3, 555: 3, 553:
3, 551: 3, 548: 3, 536: 3, 503: 3, 501: 3, 500: 3, 493: 3, 491: 3, 488:
3, 483: 3, 480: 3, 477: 3, 472: 3, 470: 3, 459: 3, 456: 3, 448: 3, 426:
3, 417: 3, 398: 3, 391: 3, 378: 3, 377: 3, 360: 3, 12232: 2, 12196: 2,
12088: 2, 9775: 2, 9637: 2, 7868: 2, 7421: 2, 7227: 2, 7052: 2, 6872:
2, 5400: 2, 5342: 2, 5218: 2, 5182: 2, 5105: 2, 4940: 2, 4768: 2, 4701:
2, 4435: 2, 4423: 2, 4232: 2, 4169: 2, 4139: 2, 4112: 2, 4067: 2, 3943:
2, 3734: 2, 3705: 2, 3605: 2, 3566: 2, 3513: 2, 3497: 2, 3481: 2, 3425:
2, 3376: 2, 3292: 2, 3249: 2, 3247: 2, 3155: 2, 3135: 2, 3101: 2, 3080:
```

```
2, 3040: 2, 3011: 2, 2911: 2, 2883: 2, 2879: 2, 2851: 2, 2801: 2, 2798:
2, 2730: 2, 2654: 2, 2650: 2, 2628: 2, 2598: 2, 2595: 2, 2584: 2, 2565:
2, 2560: 2, 2553: 2, 2547: 2, 2529: 2, 2509: 2, 2497: 2, 2493: 2, 2482:
2, 2471: 2, 2460: 2, 2433: 2, 2401: 2, 2393: 2, 2387: 2, 2376: 2, 2360:
2, 2357: 2, 2347: 2, 2308: 2, 2303: 2, 2264: 2, 2243: 2, 2230: 2, 2225:
2, 2213: 2, 2198: 2, 2187: 2, 2185: 2, 2168: 2, 2162: 2, 2133: 2, 2120:
2, 2116: 2, 2108: 2, 2098: 2, 2081: 2, 2075: 2, 2068: 2, 2057: 2, 2039:
2, 2030: 2, 2026: 2, 2025: 2, 1995: 2, 1980: 2, 1974: 2, 1971: 2, 1970:
2, 1969: 2, 1967: 2, 1963: 2, 1949: 2, 1936: 2, 1928: 2, 1914: 2, 1904:
2, 1901: 2, 1892: 2, 1891: 2, 1889: 2, 1888: 2, 1877: 2, 1869: 2, 1851:
2, 1849: 2, 1838: 2, 1823: 2, 1822: 2, 1819: 2, 1810: 2, 1803: 2, 1801:
2, 1757: 2, 1738: 2, 1732: 2, 1731: 2, 1727: 2, 1722: 2, 1720: 2, 1714:
2, 1704: 2, 1702: 2, 1697: 2, 1688: 2, 1686: 2, 1672: 2, 1656: 2, 1647:
2, 1644: 2, 1635: 2, 1633: 2, 1632: 2, 1610: 2, 1607: 2, 1605: 2, 1601:
2, 1593: 2, 1587: 2, 1581: 2, 1579: 2, 1570: 2, 1556: 2, 1554: 2, 1551:
2, 1539: 2, 1537: 2, 1531: 2, 1527: 2, 1510: 2, 1504: 2, 1498: 2, 1496:
2, 1492: 2, 1490: 2, 1489: 2, 1486: 2, 1479: 2, 1465: 2, 1444: 2, 1438:
2, 1436: 2, 1426: 2, 1424: 2, 1412: 2, 1411: 2, 1409: 2, 1404: 2, 1401:
2, 1400: 2, 1394: 2, 1378: 2, 1370: 2, 1362: 2, 1358: 2, 1355: 2, 1347:
2, 1346: 2, 1342: 2, 1340: 2, 1339: 2, 1338: 2, 1337: 2, 1329: 2, 1327:
2, 1320: 2, 1318: 2, 1309: 2, 1307: 2, 1305: 2, 1300: 2, 1284: 2, 1279:
2, 1278: 2, 1273: 2, 1271: 2, 1265: 2, 1259: 2, 1258: 2, 1257: 2, 1253:
2, 1250: 2, 1226: 2, 1223: 2, 1217: 2, 1215: 2, 1214: 2, 1211: 2, 1204:
2, 1203: 2, 1201: 2, 1200: 2, 1199: 2, 1196: 2, 1190: 2, 1186: 2, 1184:
2, 1177: 2, 1176: 2, 1174: 2, 1173: 2, 1172: 2, 1171: 2, 1166: 2, 1163:
2, 1162: 2, 1154: 2, 1150: 2, 1145: 2, 1138: 2, 1135: 2, 1134: 2, 1133:
2, 1131: 2, 1126: 2, 1123: 2, 1119: 2, 1118: 2, 1110: 2, 1099: 2, 1092:
2, 1091: 2, 1090: 2, 1087: 2, 1072: 2, 1059: 2, 1058: 2, 1056: 2, 1053:
2, 1049: 2, 1042: 2, 1040: 2, 1018: 2, 1016: 2, 1014: 2, 1013: 2, 994:
2, 990: 2, 989: 2, 985: 2, 980: 2, 978: 2, 969: 2, 964: 2, 963: 2, 962:
2, 958: 2, 948: 2, 941: 2, 937: 2, 933: 2, 932: 2, 925: 2, 913: 2, 912:
2, 889: 2, 885: 2, 878: 2, 866: 2, 857: 2, 849: 2, 848: 2, 844: 2, 841:
2, 839: 2, 837: 2, 834: 2, 829: 2, 827: 2, 821: 2, 820: 2, 812: 2, 811:
2, 806: 2, 802: 2, 801: 2, 800: 2, 798: 2, 791: 2, 787: 2, 785: 2, 784:
2, 776: 2, 774: 2, 772: 2, 763: 2, 762: 2, 760: 2, 758: 2, 753: 2, 749:
2, 747: 2, 744: 2, 743: 2, 740: 2, 734: 2, 724: 2, 719: 2, 716: 2, 715:
2, 710: 2, 707: 2, 706: 2, 704: 2, 701: 2, 699: 2, 689: 2, 686: 2, 681:
2, 680: 2, 668: 2, 661: 2, 658: 2, 652: 2, 649: 2, 646: 2, 631: 2, 623:
2, 618: 2, 616: 2, 613: 2, 604: 2, 592: 2, 589: 2, 588: 2, 583: 2, 579:
```

```
2, 567: 2, 552: 2, 547: 2, 535: 2, 524: 2, 513: 2, 486: 2, 463: 2, 439:
2, 364: 2, 149467: 1, 119638: 1, 80383: 1, 67865: 1, 67774: 1, 66625:
1, 65280: 1, 63136: 1, 62594: 1, 54026: 1, 52697: 1, 50009: 1, 47802:
1, 45806: 1, 45746: 1, 44083: 1, 42000: 1, 41726: 1, 41630: 1, 41451:
1, 40558: 1, 40163: 1, 39478: 1, 39234: 1, 38007: 1, 36951: 1, 36018:
1, 34919: 1, 34450: 1, 33278: 1, 32653: 1, 32559: 1, 32528: 1, 31863:
1, 31636: 1, 31511: 1, 28625: 1, 27554: 1, 26677: 1, 26029: 1, 25841:
1, 25670: 1, 25400: 1, 24615: 1, 24516: 1, 24256: 1, 24101: 1, 24015:
1, 23923: 1, 23770: 1, 23386: 1, 23329: 1, 22846: 1, 22593: 1, 21783:
1, 21726: 1, 21459: 1, 21442: 1, 21241: 1, 20808: 1, 20492: 1, 20267:
1, 19758: 1, 19647: 1, 19519: 1, 19298: 1, 19262: 1, 19000: 1, 18839:
1, 18726: 1, 18506: 1, 18477: 1, 18347: 1, 18125: 1, 17930: 1, 17913:
1, 17820: 1, 17695: 1, 17599: 1, 17575: 1, 17508: 1, 17432: 1, 17426:
1, 17234: 1, 17056: 1, 17016: 1, 16929: 1, 16907: 1, 16830: 1, 16715:
1, 16641: 1, 16625: 1, 16523: 1, 16486: 1, 16426: 1, 16342: 1, 15949:
1, 15913: 1, 15825: 1, 15689: 1, 15662: 1, 15500: 1, 15366: 1, 15312:
1, 15181: 1, 15111: 1, 15102: 1, 14984: 1, 14888: 1, 14794: 1, 14789:
1, 14783: 1, 14659: 1, 14609: 1, 14506: 1, 14397: 1, 14298: 1, 14235:
1, 13923: 1, 13867: 1, 13830: 1, 13642: 1, 13383: 1, 13343: 1, 13268:
1, 13254: 1, 13210: 1, 13116: 1, 13045: 1, 12997: 1, 12975: 1, 12868:
1, 12855: 1, 12795: 1, 12731: 1, 12669: 1, 12637: 1, 12609: 1, 12555:
1, 12552: 1, 12453: 1, 12381: 1, 12378: 1, 12357: 1, 12344: 1, 12260:
1, 12228: 1, 12175: 1, 12154: 1, 12137: 1, 12056: 1, 12018: 1, 12007:
1, 11994: 1, 11905: 1, 11660: 1, 11652: 1, 11644: 1, 11627: 1, 11625:
1, 11620: 1, 11489: 1, 11472: 1, 11448: 1, 11326: 1, 11229: 1, 11220:
1, 11219: 1, 11134: 1, 11132: 1, 11065: 1, 10962: 1, 10959: 1, 10938:
1, 10872: 1, 10862: 1, 10859: 1, 10810: 1, 10775: 1, 10732: 1, 10705:
1, 10587: 1, 10568: 1, 10270: 1, 10264: 1, 10203: 1, 10149: 1, 10105:
1, 10097: 1, 10092: 1, 10052: 1, 10050: 1, 9854: 1, 9847: 1, 9774: 1, 9
740: 1, 9660: 1, 9649: 1, 9643: 1, 9614: 1, 9560: 1, 9542: 1, 9531: 1,
9479: 1, 9453: 1, 9350: 1, 9342: 1, 9283: 1, 9277: 1, 9252: 1, 9238: 1,
9188: 1, 9176: 1, 9120: 1, 9118: 1, 9090: 1, 9076: 1, 9002: 1, 8965: 1,
8933: 1, 8919: 1, 8899: 1, 8869: 1, 8817: 1, 8811: 1, 8803: 1, 8801: 1,
8783: 1, 8773: 1, 8769: 1, 8762: 1, 8714: 1, 8661: 1, 8644: 1, 8626: 1,
8563: 1, 8529: 1, 8517: 1, 8439: 1, 8367: 1, 8353: 1, 8312: 1, 8292: 1,
8289: 1, 8266: 1, 8209: 1, 8169: 1, 8147: 1, 8120: 1, 8095: 1, 8093: 1,
8091: 1, 8063: 1, 8009: 1, 7987: 1, 7945: 1, 7942: 1, 7935: 1, 7932: 1,
7926: 1, 7880: 1, 7879: 1, 7867: 1, 7852: 1, 7837: 1, 7830: 1, 7796: 1,
7780: 1, 7774: 1, 7715: 1, 7705: 1, 7703: 1, 7690: 1, 7629: 1, 7577: 1,
```

```
7562: 1, 7524: 1, 7509: 1, 7506: 1, 7504: 1, 7486: 1, 7465: 1, 7412: 1,
7398: 1, 7396: 1, 7374: 1, 7364: 1, 7351: 1, 7312: 1, 7281: 1, 7273: 1,
7248: 1, 7137: 1, 7129: 1, 7095: 1, 7073: 1, 7048: 1, 7047: 1, 7045: 1,
7025: 1, 7018: 1, 7014: 1, 7004: 1, 6959: 1, 6943: 1, 6932: 1, 6923: 1,
6915: 1, 6914: 1, 6906: 1, 6866: 1, 6864: 1, 6847: 1, 6840: 1, 6827: 1,
6799: 1, 6761: 1, 6735: 1, 6716: 1, 6709: 1, 6708: 1, 6699: 1, 6682: 1,
6659: 1, 6647: 1, 6642: 1, 6640: 1, 6633: 1, 6607: 1, 6565: 1, 6556: 1,
6544: 1, 6532: 1, 6513: 1, 6497: 1, 6483: 1, 6454: 1, 6447: 1, 6445: 1,
6426: 1, 6425: 1, 6421: 1, 6410: 1, 6370: 1, 6363: 1, 6341: 1, 6330: 1,
6327: 1, 6322: 1, 6317: 1, 6297: 1, 6282: 1, 6277: 1, 6260: 1, 6245: 1,
6229: 1, 6211: 1, 6199: 1, 6187: 1, 6172: 1, 6143: 1, 6138: 1, 6075: 1,
6072: 1, 6069: 1, 6050: 1, 6049: 1, 5990: 1, 5974: 1, 5966: 1, 5961: 1,
5943: 1, 5925: 1, 5924: 1, 5917: 1, 5905: 1, 5904: 1, 5884: 1, 5882: 1,
5874: 1, 5867: 1, 5866: 1, 5864: 1, 5862: 1, 5840: 1, 5836: 1, 5810: 1,
5801: 1, 5797: 1, 5791: 1, 5781: 1, 5778: 1, 5768: 1, 5729: 1, 5724: 1,
5718: 1, 5717: 1, 5697: 1, 5680: 1, 5658: 1, 5637: 1, 5622: 1, 5603: 1,
5575: 1, 5551: 1, 5515: 1, 5511: 1, 5470: 1, 5468: 1, 5465: 1, 5450: 1,
5447: 1, 5446: 1, 5445: 1, 5432: 1, 5430: 1, 5425: 1, 5414: 1, 5392: 1,
5383: 1, 5373: 1, 5366: 1, 5364: 1, 5353: 1, 5338: 1, 5337: 1, 5336: 1,
5327: 1, 5315: 1, 5312: 1, 5303: 1, 5295: 1, 5290: 1, 5287: 1, 5286: 1,
5270: 1, 5249: 1, 5244: 1, 5168: 1, 5158: 1, 5127: 1, 5123: 1, 5107: 1,
5097: 1, 5086: 1, 5069: 1, 5056: 1, 5047: 1, 5038: 1, 5022: 1, 5004: 1,
4976: 1, 4958: 1, 4935: 1, 4928: 1, 4870: 1, 4868: 1, 4856: 1, 4855: 1,
4849: 1, 4845: 1, 4842: 1, 4839: 1, 4835: 1, 4823: 1, 4820: 1, 4816: 1,
4809: 1, 4804: 1, 4801: 1, 4776: 1, 4773: 1, 4763: 1, 4760: 1, 4759: 1,
4752: 1, 4749: 1, 4746: 1, 4731: 1, 4718: 1, 4716: 1, 4698: 1, 4693: 1,
4692: 1, 4689: 1, 4687: 1, 4671: 1, 4667: 1, 4663: 1, 4656: 1, 4639: 1,
4631: 1, 4616: 1, 4610: 1, 4590: 1, 4572: 1, 4557: 1, 4551: 1, 4548: 1,
4546: 1, 4515: 1, 4494: 1, 4493: 1, 4485: 1, 4484: 1, 4476: 1, 4448: 1,
4447: 1, 4440: 1, 4433: 1, 4413: 1, 4407: 1, 4391: 1, 4390: 1, 4376: 1,
4373: 1, 4371: 1, 4370: 1, 4369: 1, 4364: 1, 4363: 1, 4345: 1, 4335: 1,
4329: 1, 4323: 1, 4309: 1, 4306: 1, 4303: 1, 4293: 1, 4288: 1, 4280: 1,
4276: 1, 4275: 1, 4272: 1, 4270: 1, 4235: 1, 4229: 1, 4219: 1, 4204: 1,
4201: 1, 4199: 1, 4197: 1, 4193: 1, 4180: 1, 4178: 1, 4176: 1, 4157: 1,
4143: 1, 4138: 1, 4136: 1, 4135: 1, 4132: 1, 4130: 1, 4118: 1, 4115: 1,
4109: 1, 4101: 1, 4094: 1, 4086: 1, 4082: 1, 4073: 1, 4068: 1, 4066: 1,
4065: 1, 4051: 1, 4045: 1, 4040: 1, 4030: 1, 4020: 1, 4016: 1, 4010: 1,
4008: 1, 3990: 1, 3988: 1, 3965: 1, 3962: 1, 3958: 1, 3956: 1, 3953: 1,
3949: 1, 3944: 1, 3932: 1, 3924: 1, 3919: 1, 3917: 1, 3913: 1, 3902: 1,
```

```
3897: 1, 3882: 1, 3881: 1, 3868: 1, 3862: 1, 3859: 1, 3848: 1, 3832: 1,
3829: 1, 3828: 1, 3827: 1, 3818: 1, 3815: 1, 3810: 1, 3800: 1, 3781: 1,
3780: 1, 3774: 1, 3772: 1, 3758: 1, 3754: 1, 3740: 1, 3739: 1, 3726: 1,
3724: 1, 3722: 1, 3721: 1, 3713: 1, 3711: 1, 3709: 1, 3707: 1, 3701: 1,
3677: 1, 3675: 1, 3662: 1, 3661: 1, 3656: 1, 3651: 1, 3644: 1, 3643: 1,
3636: 1, 3631: 1, 3626: 1, 3621: 1, 3620: 1, 3617: 1, 3616: 1, 3597: 1,
3596: 1, 3595: 1, 3594: 1, 3592: 1, 3587: 1, 3584: 1, 3582: 1, 3581: 1,
3570: 1, 3565: 1, 3558: 1, 3556: 1, 3553: 1, 3552: 1, 3541: 1, 3538: 1,
3537: 1, 3533: 1, 3527: 1, 3521: 1, 3517: 1, 3514: 1, 3512: 1, 3511: 1,
3509: 1, 3507: 1, 3503: 1, 3495: 1, 3492: 1, 3486: 1, 3468: 1, 3466: 1,
3463: 1, 3462: 1, 3454: 1, 3447: 1, 3441: 1, 3437: 1, 3435: 1, 3430: 1,
3420: 1, 3419: 1, 3417: 1, 3414: 1, 3410: 1, 3409: 1, 3406: 1, 3402: 1,
3394: 1, 3391: 1, 3389: 1, 3382: 1, 3380: 1, 3378: 1, 3369: 1, 3362: 1,
3360: 1, 3358: 1, 3357: 1, 3355: 1, 3353: 1, 3350: 1, 3343: 1, 3338: 1,
3334: 1, 3331: 1, 3330: 1, 3327: 1, 3326: 1, 3323: 1, 3322: 1, 3309: 1,
3288: 1, 3284: 1, 3282: 1, 3280: 1, 3277: 1, 3268: 1, 3265: 1, 3262: 1,
3252: 1, 3246: 1, 3242: 1, 3241: 1, 3238: 1, 3237: 1, 3234: 1, 3233: 1,
3230: 1, 3227: 1, 3222: 1, 3215: 1, 3205: 1, 3204: 1, 3188: 1, 3186: 1,
3184: 1, 3166: 1, 3163: 1, 3161: 1, 3144: 1, 3138: 1, 3129: 1, 3128: 1,
3127: 1, 3108: 1, 3106: 1, 3105: 1, 3104: 1, 3103: 1, 3102: 1, 3099: 1,
3094: 1, 3093: 1, 3087: 1, 3076: 1, 3073: 1, 3070: 1, 3068: 1, 3064: 1,
3062: 1, 3058: 1, 3057: 1, 3055: 1, 3054: 1, 3052: 1, 3048: 1, 3047: 1,
3044: 1, 3043: 1, 3039: 1, 3029: 1, 3024: 1, 3022: 1, 3020: 1, 3014: 1,
3012: 1, 3009: 1, 3008: 1, 3006: 1, 3002: 1, 2996: 1, 2992: 1, 2991: 1,
2987: 1, 2986: 1, 2980: 1, 2967: 1, 2961: 1, 2958: 1, 2955: 1, 2950: 1,
2949: 1, 2946: 1, 2942: 1, 2926: 1, 2925: 1, 2924: 1, 2922: 1, 2920: 1,
2915: 1, 2914: 1, 2909: 1, 2905: 1, 2897: 1, 2896: 1, 2886: 1, 2885: 1,
2876: 1, 2863: 1, 2855: 1, 2852: 1, 2848: 1, 2840: 1, 2834: 1, 2828: 1,
2827: 1, 2825: 1, 2824: 1, 2822: 1, 2821: 1, 2819: 1, 2812: 1, 2807: 1,
2782: 1, 2767: 1, 2754: 1, 2748: 1, 2744: 1, 2740: 1, 2739: 1, 2738: 1,
2734: 1, 2726: 1, 2718: 1, 2714: 1, 2710: 1, 2707: 1, 2703: 1, 2687: 1,
2686: 1, 2685: 1, 2684: 1, 2682: 1, 2678: 1, 2673: 1, 2666: 1, 2661: 1,
2657: 1, 2645: 1, 2639: 1, 2632: 1, 2630: 1, 2617: 1, 2613: 1, 2610: 1,
2604: 1, 2599: 1, 2596: 1, 2594: 1, 2591: 1, 2588: 1, 2581: 1, 2580: 1,
2575: 1, 2570: 1, 2562: 1, 2558: 1, 2556: 1, 2540: 1, 2528: 1, 2526: 1,
2517: 1, 2511: 1, 2510: 1, 2507: 1, 2502: 1, 2500: 1, 2499: 1, 2496: 1,
2494: 1, 2488: 1, 2484: 1, 2483: 1, 2477: 1, 2467: 1, 2464: 1, 2461: 1,
2456: 1, 2452: 1, 2450: 1, 2449: 1, 2448: 1, 2445: 1, 2441: 1, 2439: 1,
2430: 1, 2426: 1, 2425: 1, 2422: 1, 2418: 1, 2416: 1, 2411: 1, 2409: 1,
```

```
2407: 1, 2399: 1, 2396: 1, 2378: 1, 2372: 1, 2371: 1, 2366: 1, 2365: 1,
2364: 1, 2352: 1, 2348: 1, 2346: 1, 2341: 1, 2338: 1, 2336: 1, 2334: 1,
2333: 1, 2328: 1, 2325: 1, 2321: 1, 2320: 1, 2315: 1, 2314: 1, 2311: 1,
2310: 1, 2306: 1, 2300: 1, 2299: 1, 2298: 1, 2296: 1, 2295: 1, 2291: 1,
2289: 1, 2287: 1, 2284: 1, 2281: 1, 2280: 1, 2278: 1, 2277: 1, 2276: 1,
2272: 1, 2271: 1, 2269: 1, 2261: 1, 2255: 1, 2253: 1, 2244: 1, 2239: 1,
2235: 1, 2234: 1, 2229: 1, 2221: 1, 2218: 1, 2217: 1, 2216: 1, 2215: 1,
2212: 1, 2210: 1, 2200: 1, 2193: 1, 2190: 1, 2188: 1, 2183: 1, 2176: 1,
2174: 1, 2171: 1, 2163: 1, 2161: 1, 2159: 1, 2158: 1, 2154: 1, 2139: 1,
2131: 1, 2127: 1, 2124: 1, 2122: 1, 2121: 1, 2119: 1, 2113: 1, 2110: 1,
2107: 1, 2106: 1, 2103: 1, 2102: 1, 2101: 1, 2094: 1, 2091: 1, 2087: 1,
2080: 1, 2071: 1, 2070: 1, 2067: 1, 2063: 1, 2055: 1, 2053: 1, 2051: 1,
2049: 1, 2045: 1, 2041: 1, 2040: 1, 2038: 1, 2036: 1, 2035: 1, 2033: 1,
2032: 1, 2027: 1, 2021: 1, 2018: 1, 2015: 1, 2012: 1, 2009: 1, 2004: 1,
1997: 1, 1992: 1, 1990: 1, 1989: 1, 1985: 1, 1983: 1, 1982: 1, 1981: 1,
1979: 1, 1977: 1, 1975: 1, 1968: 1, 1966: 1, 1964: 1, 1958: 1, 1954: 1,
1953: 1, 1950: 1, 1944: 1, 1941: 1, 1939: 1, 1937: 1, 1935: 1, 1933: 1,
1930: 1, 1929: 1, 1922: 1, 1921: 1, 1916: 1, 1913: 1, 1912: 1, 1909: 1,
1907: 1, 1905: 1, 1899: 1, 1898: 1, 1897: 1, 1896: 1, 1894: 1, 1893: 1,
1890: 1, 1886: 1, 1885: 1, 1884: 1, 1881: 1, 1880: 1, 1878: 1, 1865: 1,
1863: 1, 1862: 1, 1861: 1, 1859: 1, 1856: 1, 1847: 1, 1843: 1, 1842: 1,
1841: 1, 1840: 1, 1839: 1, 1837: 1, 1833: 1, 1830: 1, 1826: 1, 1825: 1,
1816: 1, 1814: 1, 1813: 1, 1811: 1, 1808: 1, 1797: 1, 1796: 1, 1795: 1,
1793: 1, 1791: 1, 1784: 1, 1783: 1, 1781: 1, 1779: 1, 1777: 1, 1774: 1,
1772: 1, 1770: 1, 1768: 1, 1767: 1, 1766: 1, 1764: 1, 1763: 1, 1759: 1,
1758: 1, 1756: 1, 1754: 1, 1753: 1, 1752: 1, 1750: 1, 1745: 1, 1740: 1,
1739: 1, 1729: 1, 1728: 1, 1721: 1, 1719: 1, 1718: 1, 1716: 1, 1715: 1,
1708: 1, 1707: 1, 1703: 1, 1701: 1, 1696: 1, 1693: 1, 1692: 1, 1690: 1,
1687: 1, 1684: 1, 1683: 1, 1678: 1, 1669: 1, 1668: 1, 1666: 1, 1662: 1,
1659: 1, 1653: 1, 1642: 1, 1640: 1, 1639: 1, 1637: 1, 1634: 1, 1630: 1,
1627: 1, 1626: 1, 1625: 1, 1622: 1, 1619: 1, 1617: 1, 1616: 1, 1615: 1,
1612: 1, 1611: 1, 1609: 1, 1606: 1, 1604: 1, 1602: 1, 1600: 1, 1598: 1,
1595: 1, 1590: 1, 1589: 1, 1588: 1, 1586: 1, 1585: 1, 1584: 1, 1583: 1,
1582: 1, 1578: 1, 1576: 1, 1573: 1, 1572: 1, 1569: 1, 1567: 1, 1564: 1,
1563: 1, 1559: 1, 1558: 1, 1552: 1, 1550: 1, 1549: 1, 1548: 1, 1547: 1,
1544: 1, 1543: 1, 1542: 1, 1541: 1, 1536: 1, 1533: 1, 1532: 1, 1530: 1,
1523: 1, 1522: 1, 1521: 1, 1519: 1, 1517: 1, 1516: 1, 1515: 1, 1513: 1,
1512: 1, 1511: 1, 1509: 1, 1508: 1, 1506: 1, 1505: 1, 1503: 1, 1501: 1,
1500: 1, 1499: 1, 1497: 1, 1495: 1, 1494: 1, 1493: 1, 1482: 1, 1476: 1,
```

```
1473: 1, 1472: 1, 1470: 1, 1469: 1, 1468: 1, 1467: 1, 1464: 1, 1463: 1,
1461: 1, 1455: 1, 1452: 1, 1451: 1, 1450: 1, 1443: 1, 1442: 1, 1441: 1,
1435: 1, 1432: 1, 1429: 1, 1428: 1, 1421: 1, 1420: 1, 1418: 1, 1414: 1,
1413: 1, 1408: 1, 1407: 1, 1406: 1, 1399: 1, 1398: 1, 1397: 1, 1392: 1,
1390: 1, 1389: 1, 1385: 1, 1384: 1, 1383: 1, 1382: 1, 1374: 1, 1373: 1,
1372: 1, 1369: 1, 1368: 1, 1366: 1, 1364: 1, 1361: 1, 1360: 1, 1359: 1,
1357: 1, 1352: 1, 1349: 1, 1348: 1, 1341: 1, 1328: 1, 1322: 1, 1321: 1,
1319: 1, 1317: 1, 1315: 1, 1314: 1, 1312: 1, 1306: 1, 1303: 1, 1302: 1,
1301: 1, 1299: 1, 1298: 1, 1295: 1, 1294: 1, 1293: 1, 1283: 1, 1282: 1,
1277: 1, 1276: 1, 1274: 1, 1270: 1, 1268: 1, 1266: 1, 1264: 1, 1263: 1,
1260: 1, 1255: 1, 1251: 1, 1248: 1, 1245: 1, 1237: 1, 1235: 1, 1232: 1,
1231: 1, 1228: 1, 1222: 1, 1218: 1, 1216: 1, 1213: 1, 1212: 1, 1210: 1,
1209: 1, 1208: 1, 1205: 1, 1198: 1, 1197: 1, 1195: 1, 1193: 1, 1191: 1,
1179: 1, 1178: 1, 1170: 1, 1169: 1, 1168: 1, 1167: 1, 1165: 1, 1160: 1,
1158: 1, 1157: 1, 1156: 1, 1149: 1, 1148: 1, 1144: 1, 1141: 1, 1140: 1,
1137: 1, 1129: 1, 1128: 1, 1121: 1, 1120: 1, 1117: 1, 1116: 1, 1115: 1,
1114: 1, 1113: 1, 1107: 1, 1105: 1, 1103: 1, 1100: 1, 1097: 1, 1095: 1,
1094: 1, 1093: 1, 1089: 1, 1088: 1, 1083: 1, 1082: 1, 1081: 1, 1078: 1,
1074: 1, 1071: 1, 1069: 1, 1068: 1, 1067: 1, 1066: 1, 1064: 1, 1054: 1,
1051: 1, 1048: 1, 1047: 1, 1036: 1, 1035: 1, 1030: 1, 1024: 1, 1021: 1,
1020: 1, 1017: 1, 1011: 1, 1010: 1, 1009: 1, 1008: 1, 1004: 1, 1003: 1,
1001: 1, 999: 1, 995: 1, 993: 1, 988: 1, 987: 1, 983: 1, 982: 1, 981:
1, 979: 1, 976: 1, 974: 1, 973: 1, 972: 1, 965: 1, 961: 1, 956: 1, 955:
1, 954: 1, 953: 1, 952: 1, 950: 1, 943: 1, 931: 1, 930: 1, 928: 1, 926:
1, 924: 1, 922: 1, 920: 1, 919: 1, 918: 1, 915: 1, 906: 1, 903: 1, 899:
1, 894: 1, 890: 1, 888: 1, 887: 1, 880: 1, 879: 1, 877: 1, 875: 1, 869:
1, 862: 1, 861: 1, 858: 1, 855: 1, 853: 1, 852: 1, 851: 1, 845: 1, 842:
1, 836: 1, 833: 1, 831: 1, 828: 1, 825: 1, 818: 1, 804: 1, 793: 1, 789:
1, 788: 1, 783: 1, 778: 1, 775: 1, 764: 1, 757: 1, 755: 1, 752: 1, 745:
1, 742: 1, 738: 1, 737: 1, 725: 1, 722: 1, 720: 1, 700: 1, 694: 1, 684:
1, 679: 1, 671: 1, 667: 1, 660: 1, 651: 1, 647: 1, 641: 1, 637: 1, 629:
1, 608: 1, 595: 1, 590: 1, 577: 1, 572: 1, 564: 1, 561: 1, 540: 1, 530:
1, 526: 1, 420: 1})
```

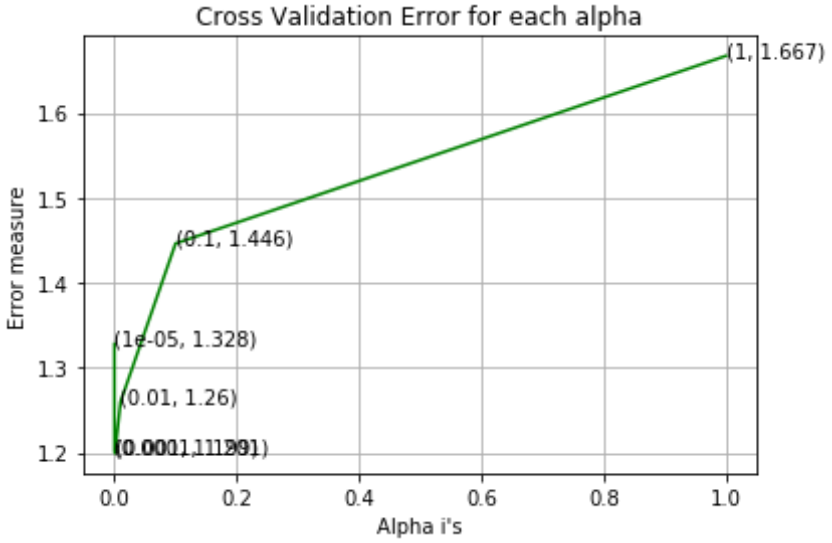
In [55]:

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

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

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

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

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


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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
y[i]))
plt.grid()
```

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3278375195465977
For values of alpha =  0.0001 The log loss is: 1.2009257191674434
For values of alpha =  0.001 The log loss is: 1.1991599867933538
For values of alpha =  0.01 The log loss is: 1.259683313542512
For values of alpha =  0.1 The log loss is: 1.4460939623244384
For values of alpha =  1 The log loss is: 1.6669680722593103
```




```
For values of best alpha =  0.001 The train log loss is: 0.697216842974
4666
For values of best alpha =  0.001 The cross validation log loss is: 1.1
991599867933538
For values of best alpha =  0.001 The test log loss is: 1.1601400364745
131
```

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

Ans. Yes, it seems like!

In [56]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
```

```
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [57]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
 train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appe
ared in train data")
```

```
95.557 % of word of test data appeared in train data
98.03 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [58]:
```
#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 probabilit
ies belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
```

```
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [59]:
```
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 [60]:
```
# 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(word,yes_no))
        elif (v < fea1_len+fea2_len):
```

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

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

Stacking the three types of features

In [61]:
```
# 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,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))
```

```
train_gene_var_tfidf = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_onehotCoding))
test_gene_var_tfidf = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_tfidf = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))


train_gene_var_ngram = hstack((train_gene_feature_onehotCoding,train_va
riation_feature_onehotCoding))
test_gene_var_ngram = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_ngram = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))


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

train_x_tfidf = hstack((train_gene_var_tfidf, train_text_feature_tfidf
)).tocsr()

train_x_ngram = hstack((train_gene_var_ngram, train_text_feature_ngram
)).tocsr()

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

test_x_ngram= hstack((test_gene_var_ngram, test_text_feature_ngram)).to
csr()

test_x_tfidf = hstack((test_gene_var_tfidf, test_text_feature_tfidf)).t
ocsr()

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_o
nehotCoding)).tocsr()
```

```
cv_y = np.array(list(cv_df['Class']))

cv_x_tfidf = hstack((cv_gene_var_tfidf, cv_text_feature_tfidf)).tocsr()

cv_x_ngram = hstack((cv_gene_var_ngram, cv_text_feature_ngram)).tocsr()

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, trai
n_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_t
ext_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [62]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55
003)
(number of data points * number of features) in test data =  (665, 5500
3)
(number of data points * number of features) in cross validation data =
(532, 55003)
```

In [63]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding.shape)
```

```
print("(number of data points * number of features) in test data = ", t
est_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data =
(532, 27)
```

In [64]:
```
print("TFIDF encoded features :")
print("(number of data points * number of features) in train data = ",
train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", t
est_x_tfidf.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_tfidf.shape)
```

```
TFIDF encoded features :
(number of data points * number of features) in train data =  (2124, 41
97)
(number of data points * number of features) in test data =  (665, 419
7)
(number of data points * number of features) in cross validation data =
(532, 4197)
```

In [65]:
```
print("BOW with 1 or 2 grams :")
print("(number of data points * number of features) in train data = ",
train_x_ngram.shape)
print("(number of data points * number of features) in test data = ", t
est_x_ngram.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_ngram.shape)
```

```
BOW with 1 or 2 grams :
(number of data points * number of features) in train data =  (2124, 41
97)
(number of data points * number of features) in test data =  (665, 419
```

```
7)

(number of data points * number of features) in cross validation data =
(532, 4197)
```

4.1. Base Line Model

4.1.1. Naive Bayes

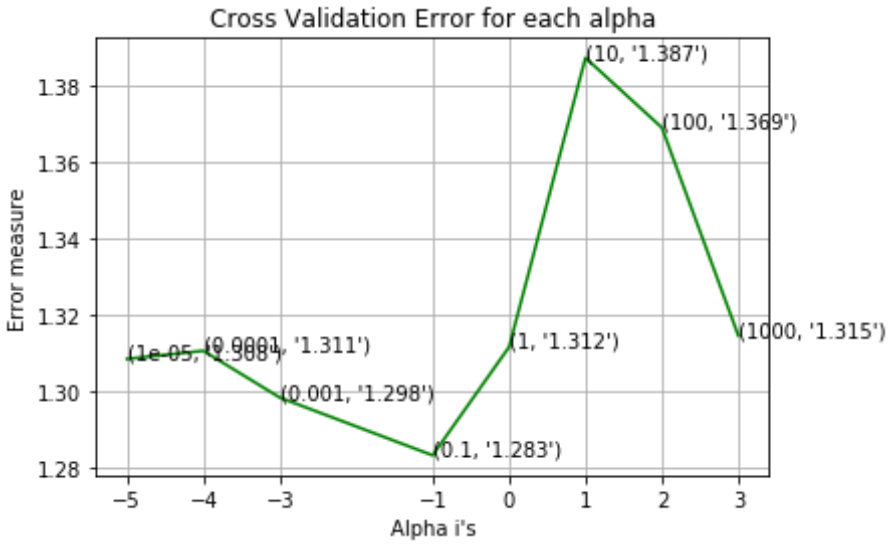
4.1.1.1. Hyper parameter tuning

In [66]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to
 X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test v
ector X.
# ----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/naive-bayes-algorithm-1/
# ----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
```

```
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/naive-bayes-algorithm-1/
# ----------------------


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

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

```
plt.show()


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


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

```
for alpha = 1e-05
Log Loss : 1.3084358395825717
for alpha = 0.0001
Log Loss : 1.3105677060506127
for alpha = 0.001
Log Loss : 1.2983210012188227
for alpha = 0.1
Log Loss : 1.283173798778252
for alpha = 1
Log Loss : 1.311556866895866
for alpha = 10
Log Loss : 1.387372472532212
for alpha = 100
Log Loss : 1.3690721024860462
for alpha = 1000
Log Loss : 1.3145558003952669
```




```
For values of best alpha =  0.1 The train log loss is: 0.914665433629
7331
For values of best alpha =  0.1 The cross validation log loss is: 1.2
83173798778252
For values of best alpha =  0.1 The test log loss is: 1.2719540130293
25
```

4.1.1.2. Testing the model with best hyper paramters

In [67]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
```

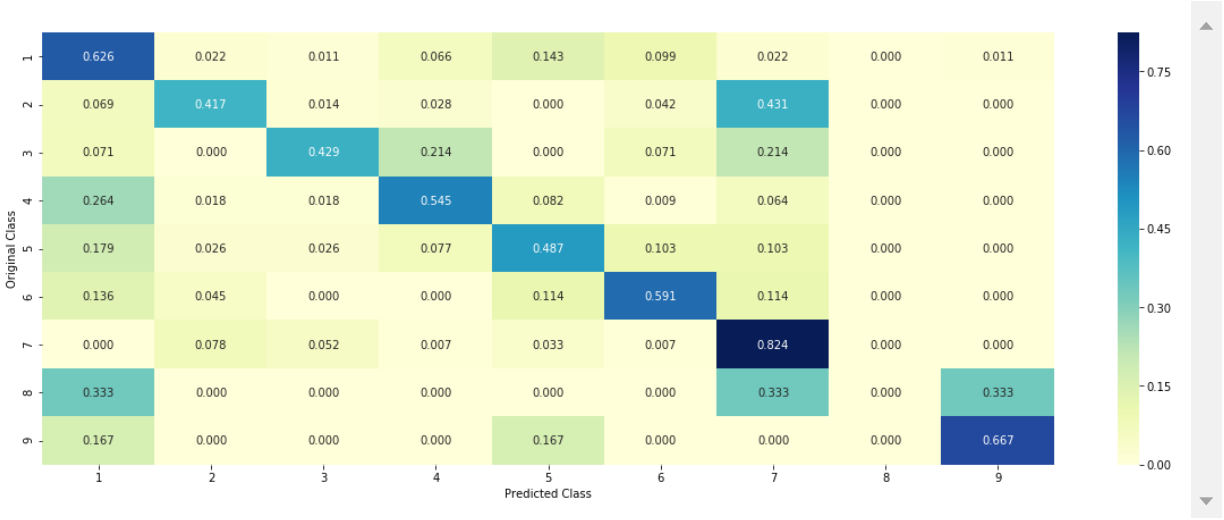
```
ior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to
 X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test v
ector X.
# ----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/naive-bayes-algorithm-1/
# ----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-pro
bability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
```

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

```
Log Loss : 1.283173798778252
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```


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


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



4.1.1.3. Feature Importance, Correctly classified point

In [68]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0714 0.0703 0.0119 0.1031 0.0336 0.5
694 0.1339 0.0035 0.0029]]
Actual Class : 6
--------------------------------------------------
```

```
7 Text feature [i2285v] present in test data point [True]
8 Text feature [brca] present in test data point [True]
9 Text feature [ivs20] present in test data point [True]
10 Text feature [personal] present in test data point [True]
12 Text feature [odds] present in test data point [True]
13 Text feature [falling] present in test data point [True]
16 Text feature [history] present in test data point [True]
18 Text feature [ivs5] present in test data point [True]
19 Text feature [433] present in test data point [True]
Out of the top  100  features  9 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [69]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5502 0.0815 0.0138 0.1196 0.0389 0.0
333 0.1553 0.0041 0.0033]]
Actual Class : 1
--------------------------------------------------
13 Text feature [type] present in test data point [True]
14 Text feature [affect] present in test data point [True]
15 Text feature [one] present in test data point [True]
16 Text feature [dna] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [protein] present in test data point [True]
19 Text feature [two] present in test data point [True]
```

```
19 Text feature [two] present in test data point [True]
20 Text feature [wild] present in test data point [True]
21 Text feature [loss] present in test data point [True]
22 Text feature [reduced] present in test data point [True]
23 Text feature [amino] present in test data point [True]
24 Text feature [present] present in test data point [True]
25 Text feature [four] present in test data point [True]
26 Text feature [acids] present in test data point [True]
27 Text feature [region] present in test data point [True]
28 Text feature [therefore] present in test data point [True]
29 Text feature [nonsense] present in test data point [True]
30 Text feature [three] present in test data point [True]
31 Text feature [sequence] present in test data point [True]
32 Text feature [changes] present in test data point [True]
33 Text feature [possible] present in test data point [True]
34 Text feature [corresponding] present in test data point [True]
36 Text feature [binding] present in test data point [True]
37 Text feature [large] present in test data point [True]
38 Text feature [analysis] present in test data point [True]
39 Text feature [table] present in test data point [True]
40 Text feature [sequences] present in test data point [True]
41 Text feature [least] present in test data point [True]
42 Text feature [used] present in test data point [True]
43 Text feature [effect] present in test data point [True]
44 Text feature [conserved] present in test data point [True]
45 Text feature [identified] present in test data point [True]
46 Text feature [data] present in test data point [True]
47 Text feature [containing] present in test data point [True]
48 Text feature [whether] present in test data point [True]
49 Text feature [frameshift] present in test data point [True]
50 Text feature [likely] present in test data point [True]
51 Text feature [result] present in test data point [True]
52 Text feature [using] present in test data point [True]
53 Text feature [also] present in test data point [True]
54 Text feature [specific] present in test data point [True]
55 Text feature [six] present in test data point [True]
56 Text feature [results] present in test data point [True]
57 Text feature [indicate] present in test data point [True]
58 Text feature [indicating] present in test data point [True]
59 Text feature [five] present in test data point [True]
```

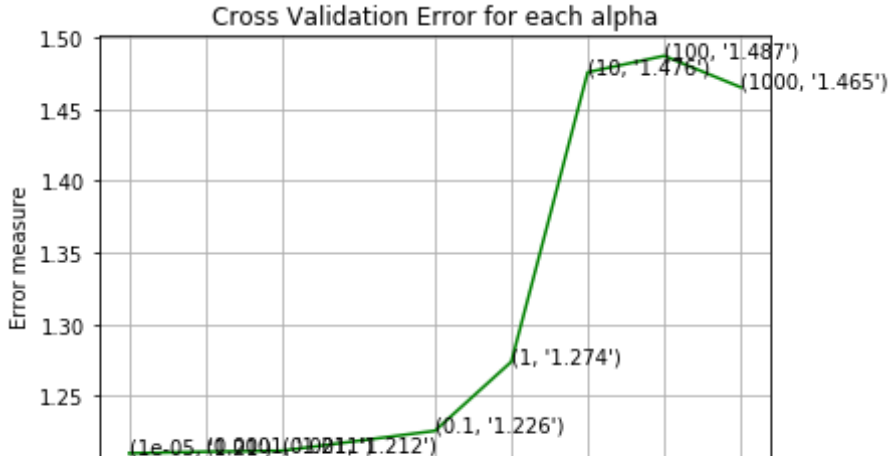
```
60 Text feature [gene] present in test data point [True]
61 Text feature [expected] present in test data point [True]
62 Text feature [shown] present in test data point [True]
64 Text feature [either] present in test data point [True]
65 Text feature [determined] present in test data point [True]
66 Text feature [discussion] present in test data point [True]
67 Text feature [possibility] present in test data point [True]
68 Text feature [involved] present in test data point [True]
70 Text feature [genetic] present in test data point [True]
71 Text feature [affected] present in test data point [True]
72 Text feature [previous] present in test data point [True]
73 Text feature [within] present in test data point [True]
74 Text feature [define] present in test data point [True]
75 Text feature [human] present in test data point [True]
76 Text feature [ability] present in test data point [True]
77 Text feature [control] present in test data point [True]
78 Text feature [addition] present in test data point [True]
79 Text feature [remains] present in test data point [True]
81 Text feature [efficiency] present in test data point [True]
82 Text feature [majority] present in test data point [True]
83 Text feature [proteins] present in test data point [True]
84 Text feature [coding] present in test data point [True]
85 Text feature [functions] present in test data point [True]
86 Text feature [form] present in test data point [True]
88 Text feature [eight] present in test data point [True]
89 Text feature [affecting] present in test data point [True]
90 Text feature [change] present in test data point [True]
91 Text feature [suppressor] present in test data point [True]
92 Text feature [important] present in test data point [True]
93 Text feature [revealed] present in test data point [True]
94 Text feature [additional] present in test data point [True]
95 Text feature [performed] present in test data point [True]
97 Text feature [whereas] present in test data point [True]
98 Text feature [fraction] present in test data point [True]
99 Text feature [different] present in test data point [True]
Out of the top  100  features  81 are present in query point
```

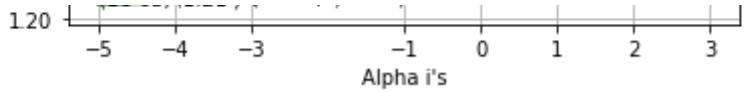
Using TFIDF Vectorization

In [70]:
```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log
```

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

```
for alpha = 1e-05
Log Loss : 1.2099377443690835
for alpha = 0.0001
Log Loss : 1.211047427425721
for alpha = 0.001
Log Loss : 1.2117267151869757
for alpha = 0.1
Log Loss : 1.2255951089219987
for alpha = 1
Log Loss : 1.2737764105985152
for alpha = 10
Log Loss : 1.4756459221536893
for alpha = 100
Log Loss : 1.4870500204852257
for alpha = 1000
Log Loss : 1.4651688302581019
```



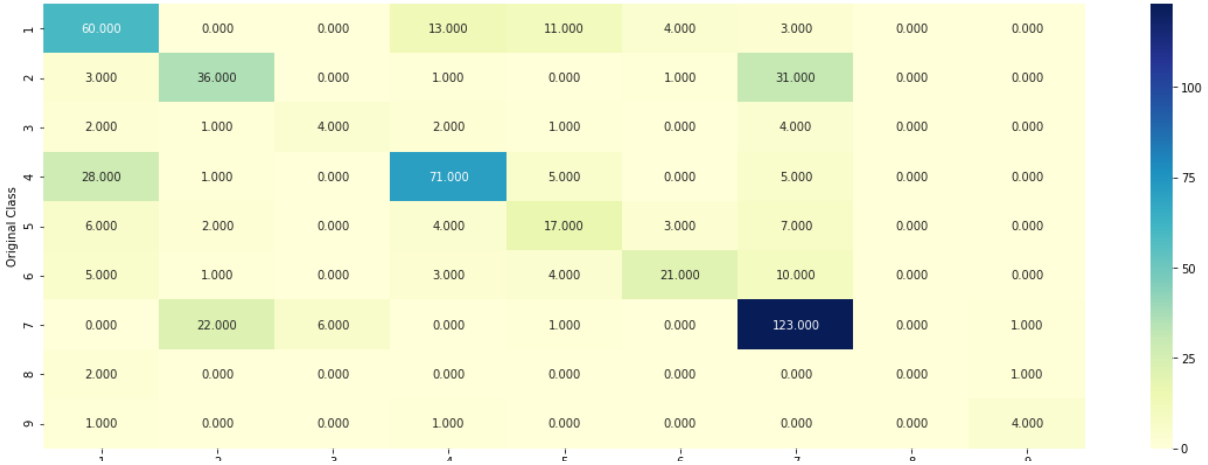



```
For values of best alpha =  1e-05 The train log loss is: 0.557919932452
4332
For values of best alpha =  1e-05 The cross validation log loss is: 1.2
099377443690835
For values of best alpha =  1e-05 The test log loss is: 1.1941911086117
514
```

In [71]:

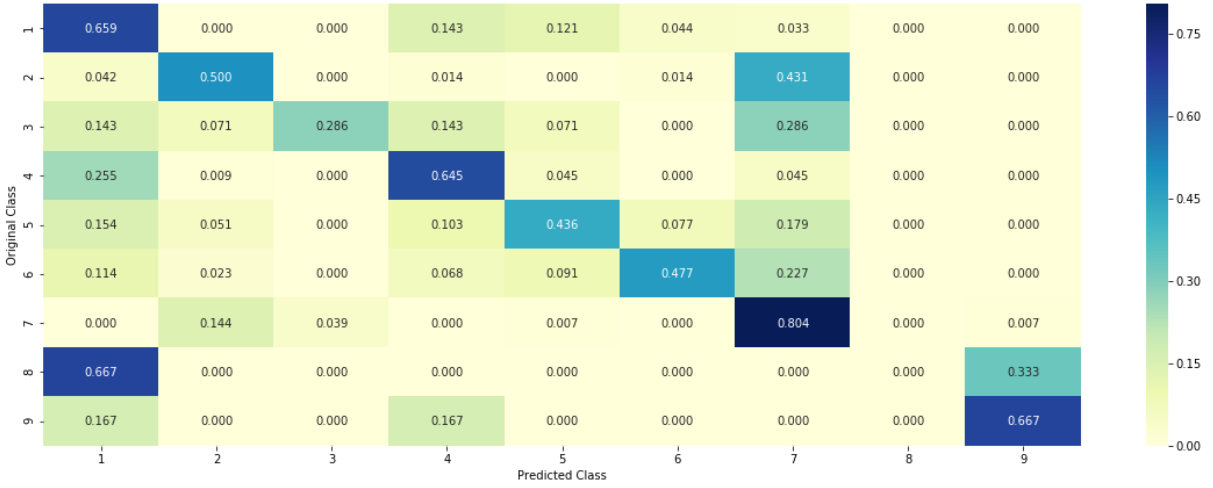
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
# to avoid rounding error while multiplying probabilites we use log-pro
bability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_tfidf)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf.toarray()))
```

```
Log Loss : 1.2099377443690835
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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

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



In [72]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0595 0.0444 0.0116 0.0737 0.0356 0.6
816 0.0879 0.0035 0.0023]]
Actual Class : 6
--------------------------------------------------
31 Text feature [000] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

In [73]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6776 0.0489 0.0128 0.0822 0.0392 0.0
361 0.0968 0.0039 0.0025]]
Actual Class : 1
--------------------------------------------------
16 Text feature [1b] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.2. K Nearest Neighbour Classification

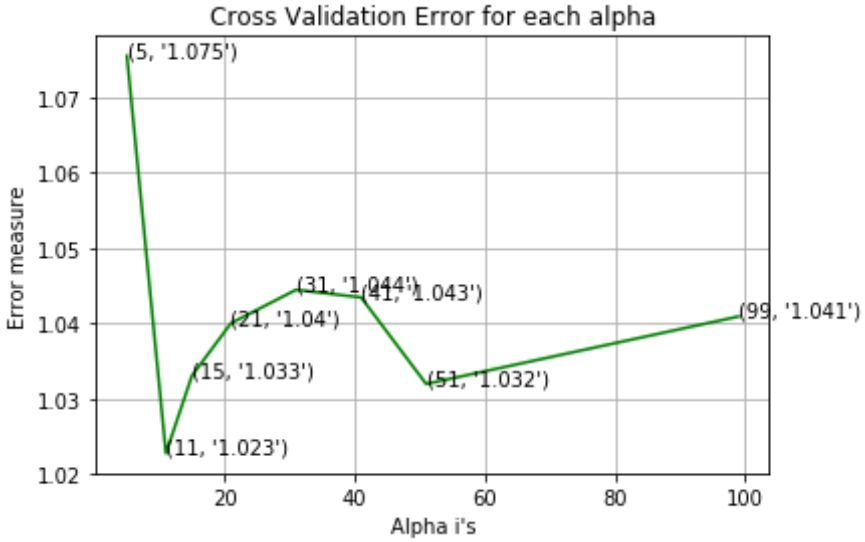
4.2.1. Hyper parameter tuning

In [74]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-examp
le-1/
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
```

```
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------


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

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

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

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

```
for alpha = 5
Log Loss : 1.0754840534912007
for alpha = 11
Log Loss : 1.0227570587714927
for alpha = 15
Log Loss : 1.0330592197843973
for alpha = 21
Log Loss : 1.040016550748663
for alpha = 31
Log Loss : 1.044445280564074
for alpha = 41
Log Loss : 1.0434371413195695
for alpha = 51
Log Loss : 1.0319586608868954
for alpha = 99
Log Loss : 1.0409030253550136
```




```
For values of best alpha =  11 The train log loss is: 0.6495482670309
317
For values of best alpha =  11 The cross validation log loss is: 1.02
27570587714927
For values of best alpha =  11 The test log loss is: 1.04444097869581
44
```

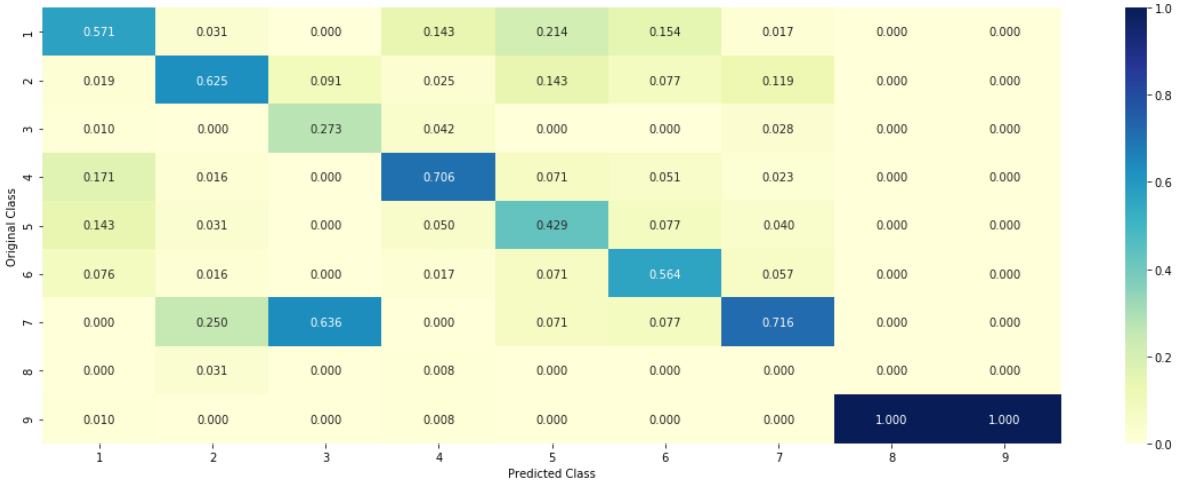
4.2.2. Testing the model with best hyper paramters

In [75]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
```

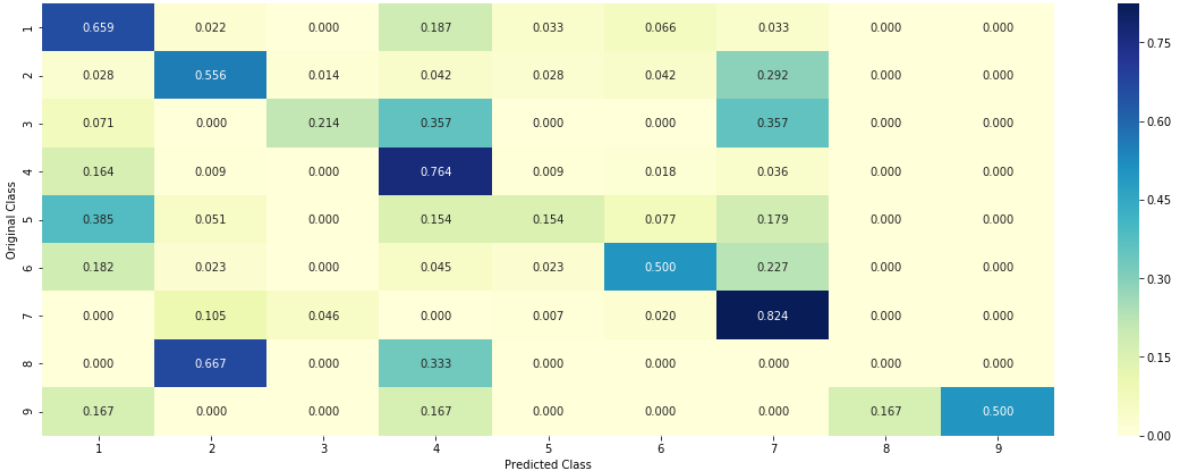
```
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.0227570587714927
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```


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




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



4.2.3.Sample Query point -1

In [76]:
```
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].resh
ape(1, -1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points
 belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 6
The  11  nearest neighbours of the test points belongs to classes [6 6
6 6 6 6 6 6 6 6 6]
Fequency of nearest points : Counter({6: 11})
```

4.2.4. Sample Query Point-2

In [77]:

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

```
Predicted Class : 1
Actual Class : 1
```

```
Actual class : 1
```

```
the k value for knn is 11 and the nearest neighbours of the test points
belongs to classes [1 1 1 1 1 4 4 4 6 1 1]
Fequency of nearest points : Counter({1: 7, 4: 3, 6: 1})
```

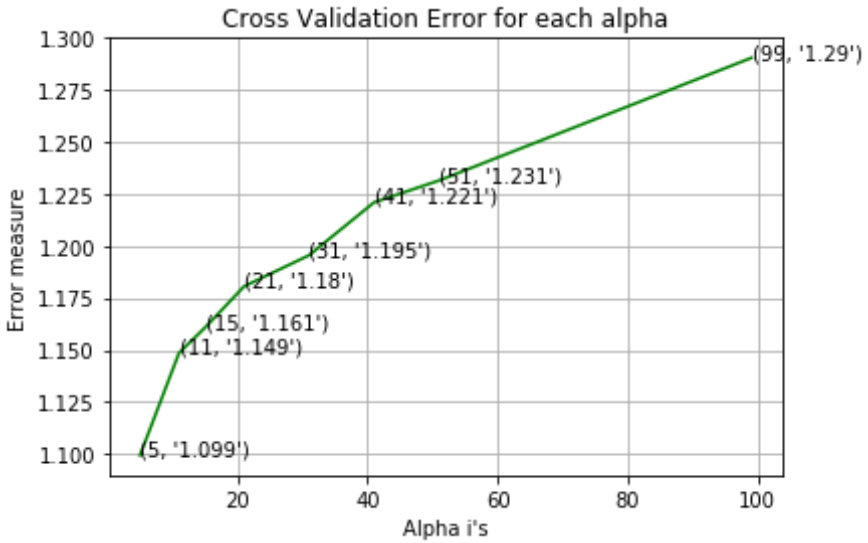
Using TFIDF Vectorization

In [78]:

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

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


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

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

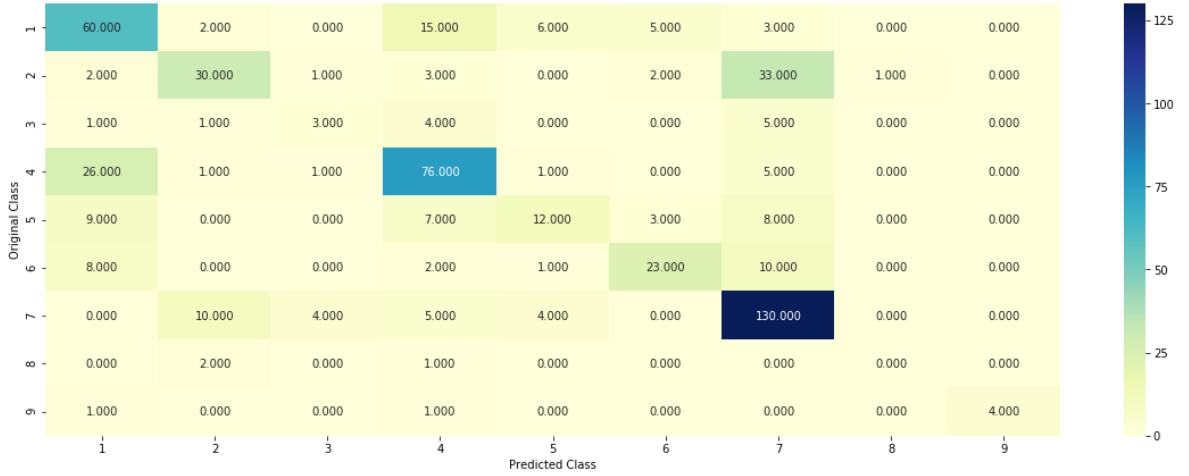
```
for alpha = 5
Log Loss : 1.09945206855152
for alpha = 11
Log Loss : 1.1485343198725266
for alpha = 15
Log Loss : 1.160694517426222
for alpha = 21
Log Loss : 1.1804701372028543
for alpha = 31
Log Loss : 1.1954101071915006
for alpha = 41
Log Loss : 1.2208037973773316
for alpha = 51
Log Loss : 1.2312854238890563
for alpha = 99
Log Loss : 1.2901020568347985
```




```
For values of best alpha =  5 The train log loss is: 0.9076965460280116
For values of best alpha =  5 The cross validation log loss is: 1.09945
206855152
For values of best alpha =  5 The test log loss is: 1.1049203460348955
```

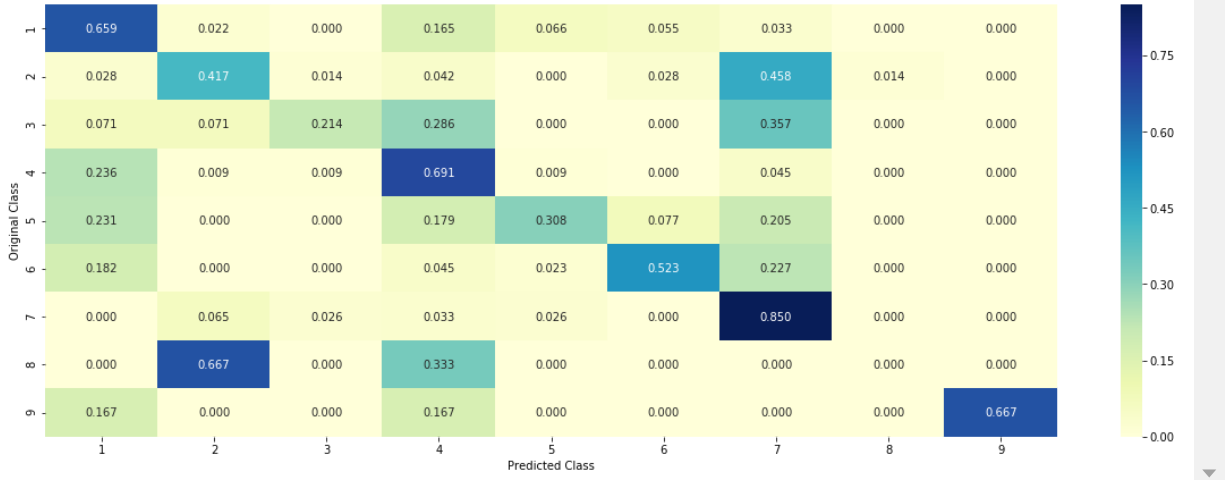
In [79]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, c
v_y, clf)
```

```
Log loss : 1.09945206855152
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```



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

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



In [80]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

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

In [81]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
```

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

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_tfidf[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_tfidf[test_point_index].reshape(1, -1
), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

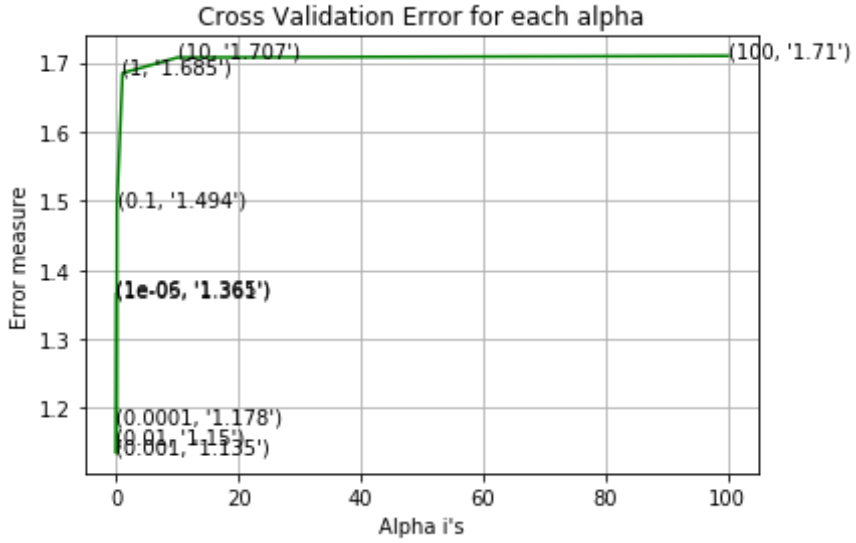
In [82]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
```

```
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/geometric-intuition-1/
#-------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-----------------------------------
# video link:
#-----------------------------------

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

```
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.364653422455021
```

```
for alpha = 1e-05
Log Loss : 1.36131257859915
for alpha = 0.0001

Log Loss : 1.178011949086007
for alpha = 0.001
Log Loss : 1.134652365301984
for alpha = 0.01
Log Loss : 1.1499655928736663
for alpha = 0.1
Log Loss : 1.4937302367899856
for alpha = 1
Log Loss : 1.6847480316683545
for alpha = 10
Log Loss : 1.707245469798681
for alpha = 100
Log Loss : 1.709670565800458
```

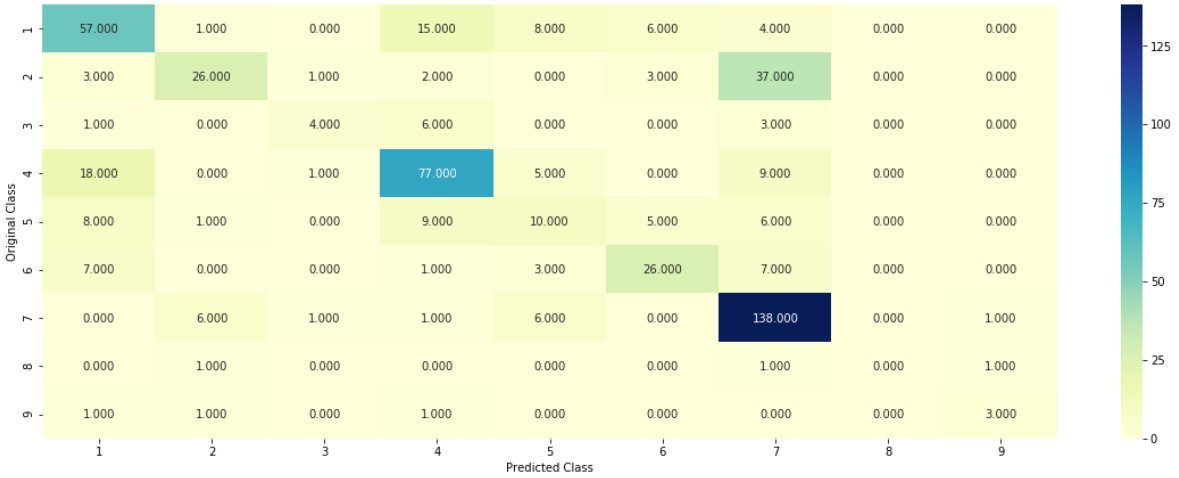



```
For values of best alpha =  0.001 The train log loss is: 0.558696394018
7292
For values of best alpha =  0.001 The cross validation log loss is: 1.1
34652365301984
For values of best alpha =  0.001 The test log loss is: 1.0694800934756
02
```

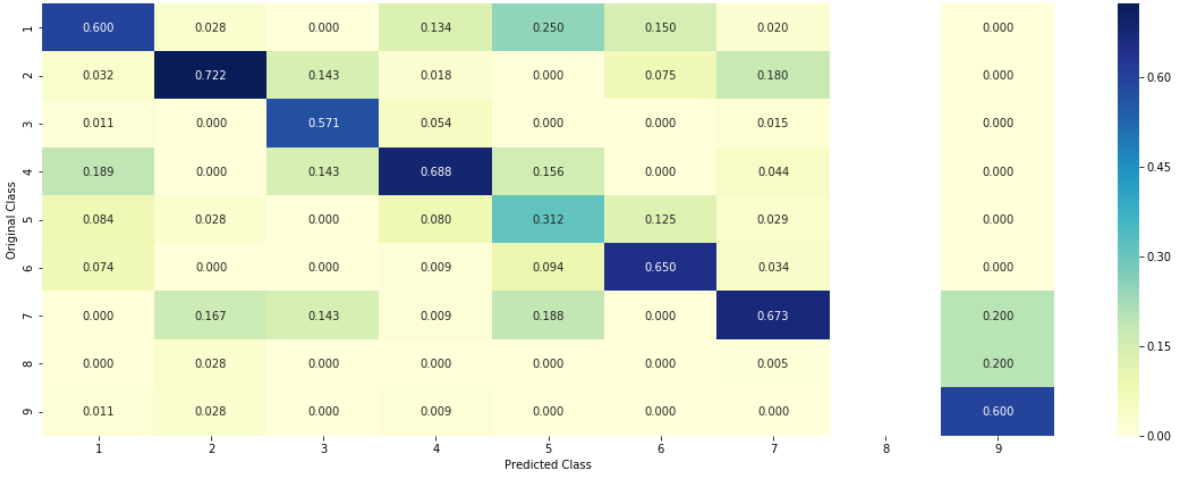
4.3.1.2. Testing the model with best hyper paramters

In [83]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

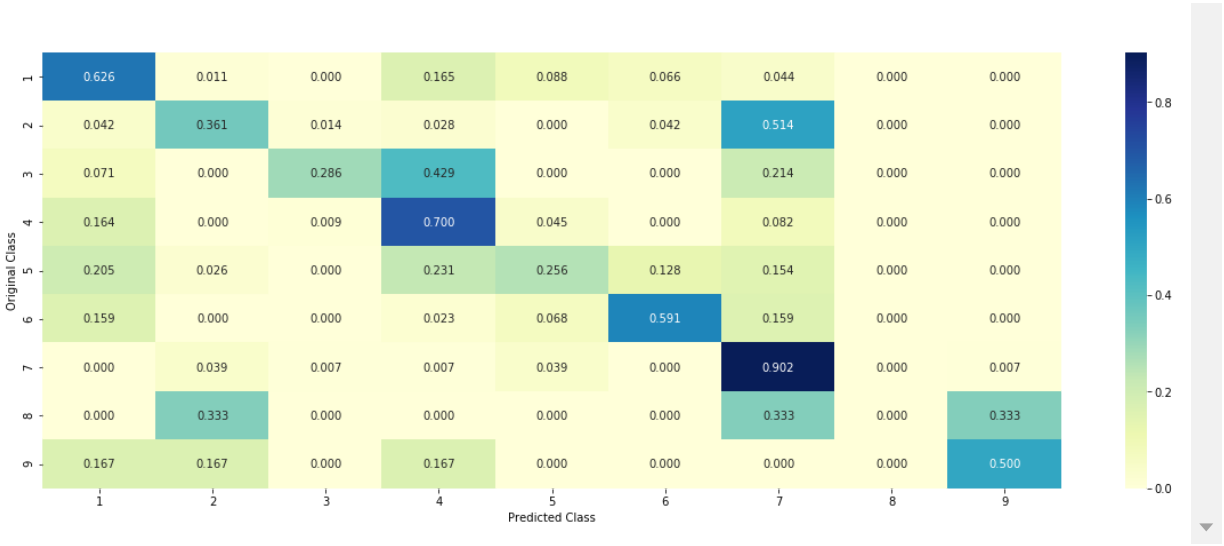
```
Log loss : 1.134652365301984
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [84]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
```

```
ry point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

4.3.1.3.1. Correctly Classified point

In [85]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0401 0.0524 0.0123 0.0386 0.1825 0.6
216 0.0425 0.0052 0.0048]]
Actual Class : 6
--------------------------------------------------
89 Text feature [v1804d] present in test data point [True]
245 Text feature [exosap] present in test data point [True]
344 Text feature [usb] present in test data point [True]
384 Text feature [women] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [86]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[8.204e-01 4.440e-02 3.000e-03 7.030e-0
2 2.600e-03 1.100e-03 5.250e-02
  4.900e-03 7.000e-04]]
Actual Class : 1
--------------------------------------------------
276 Text feature [derepression] present in test data point [True]
284 Text feature [89] present in test data point [True]
298 Text feature [prmt5] present in test data point [True]
396 Text feature [nonsense] present in test data point [True]
438 Text feature [boundary] present in test data point [True]
470 Text feature [intragenic] present in test data point [True]
475 Text feature [truncating] present in test data point [True]
486 Text feature [deletion] present in test data point [True]
487 Text feature [frameshift] present in test data point [True]
495 Text feature [decay] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

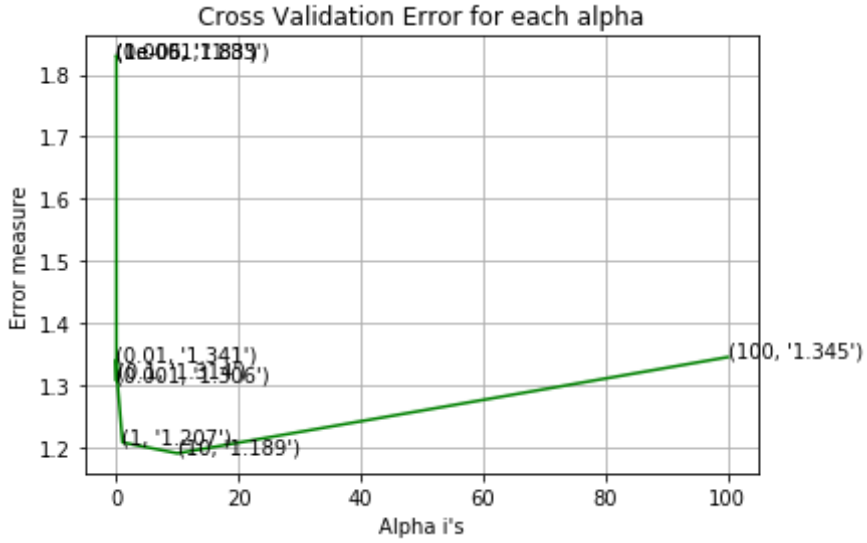
Using ngram with BOW

In [87]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
```

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

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.8304997567764278
for alpha = 1e-05
Log Loss : 1.8304997567764278
for alpha = 0.0001
Log Loss : 1.8304997567764278
for alpha = 0.001
Log Loss : 1.3061860862876324
for alpha = 0.01
Log Loss : 1.3407261918460418
for alpha = 0.1
Log Loss : 1.3136714376325198
for alpha = 1
Log Loss : 1.207019976731992
for alpha = 10
Log Loss : 1.1894541232745797
for alpha = 100
Log Loss : 1.34461620399015
```

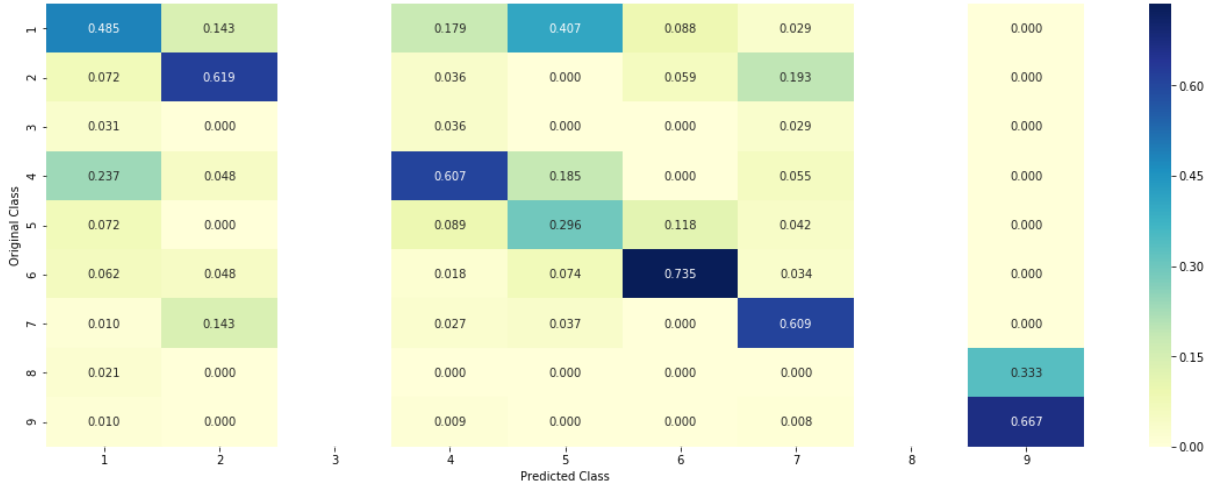


```
For values of best alpha =  10 The train log loss is: 0.959908730929978
2
For values of best alpha =  10 The cross validation log loss is: 1.1894
541232745797
For values of best alpha =  10 The test log loss is: 1.1894533601728738
```

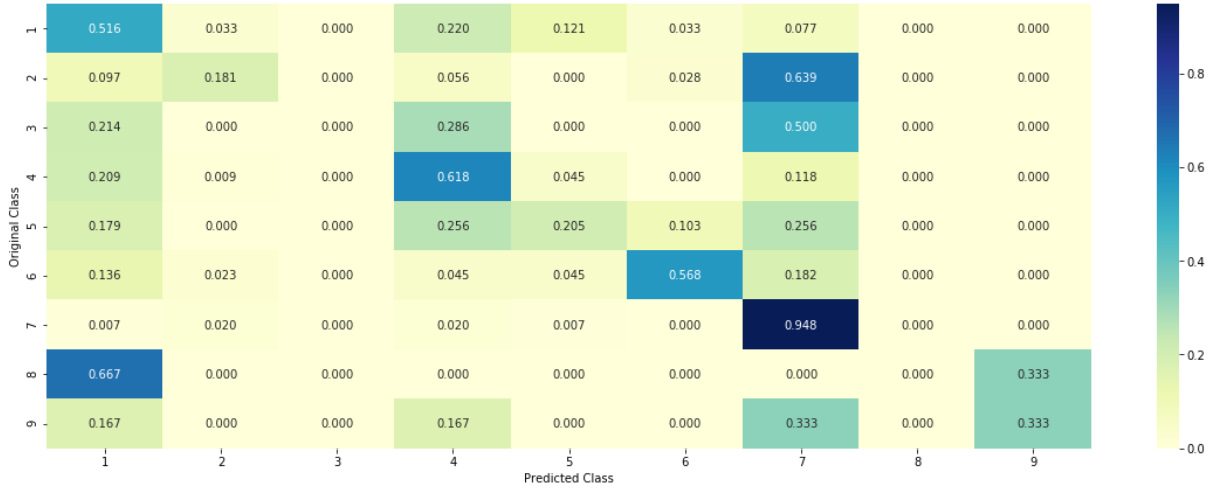
In [88]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_ngram, train_y, cv_x_ngram, c
v_y, clf)
```

```
Log loss : 1.1894541232745797
Number of mis-classified points : 0.42105263157894735
-------------------- Confusion matrix --------------------
```


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



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



In [89]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
ry point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

In [90]:
```
'''clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alph
a], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_ngram,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngram[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
(test_x_ngram[test_point_index]),4))
```

```
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names( test_df['TEXT'].iloc[test_point_index],indices
[0],no_feature)'''
```

Out[90]:

```
'clf = SGDClassifier(class_weight=\'balanced\', alpha=alpha[best_alph
a], penalty=\'l2\', loss=\'log\', random_state=42)\nclf.fit(train_x_ngr
am,train_y)\ntest_point_index = 1\nno_feature = 500\npredicted_cls = si
g_clf.predict(test_x_ngram[test_point_index])\nprint("Predicted Class
:", predicted_cls[0])\nprint("Predicted Class Probabilities:", np.round
(sig_clf.predict_proba(test_x_ngram[test_point_index]),4))\nprint("Actu
al Class :", test_y[test_point_index])\nindices = np.argsort(-clf.coef
_)[predicted_cls-1][:,:no_feature]\nprint("-"*50)\nget_imp_feature_name
s( test_df[\'TEXT\'].iloc[test_point_index],indices[0],no_feature)'
```

In [91]:

```
'''test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngram[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
(test_x_ngram[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_inde
x],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[tes
t_point_index], no_feature)'''
```

Out[91]:

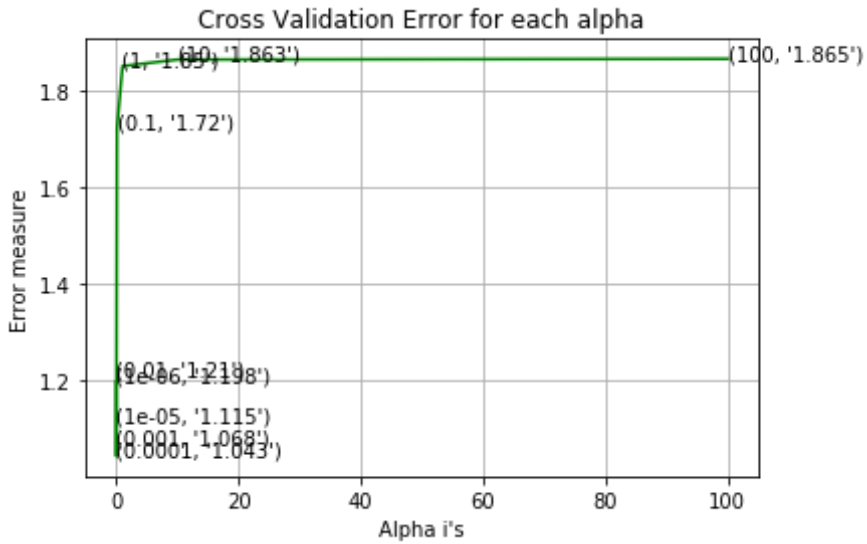
```
'test_point_index = 100\nno_feature = 500\npredicted_cls = sig_clf.pred
ict(test_x_ngram[test_point_index])\nprint("Predicted Class :", predict
ed_cls[0])\nprint("Predicted Class Probabilities:", np.round(sig_clf.pr
edict_proba(test_x_ngram[test_point_index]),4))\nprint("Actual Class
:", test_y[test_point_index])\nindices = np.argsort(-clf.coef_)[predict
ed_cls-1][:,:no_feature]\nprint("-"*50)\nget_impfeature_names(indices
[0], test_df[\'TEXT\'].iloc[test_point_index],test_df[\'Gene\'].iloc[te
st_point_index],test_df[\'Variation\'].iloc[test_point_index], no_featu
re)'
```

Using TFIDF vectorizer

In [92]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
 loss='log', random_state=42)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

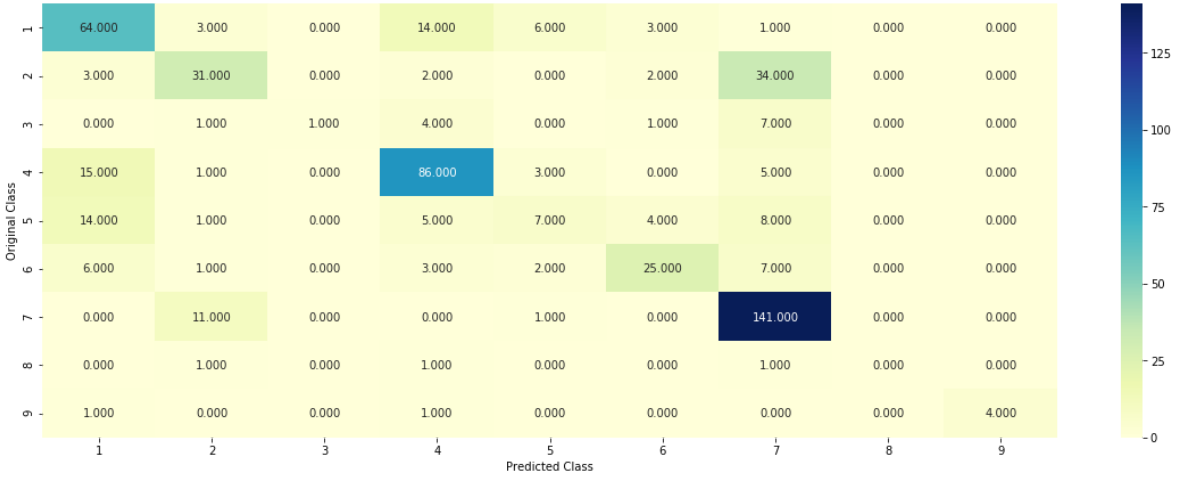
```
for alpha = 1e-06
Log Loss : 1.1975142066252216
for alpha = 1e-05
Log Loss : 1.114865088023385
for alpha = 0.0001
Log Loss : 1.0433440991807044
for alpha = 0.001
Log Loss : 1.0681495388612647
for alpha = 0.01
Log Loss : 1.210410302069057
for alpha = 0.1
Log Loss : 1.7204667089419212
for alpha = 1
Log Loss : 1.8495011602885278
for alpha = 10
Log Loss : 1.8630712009146557
for alpha = 100
Log Loss : 1.8646091410876398
```



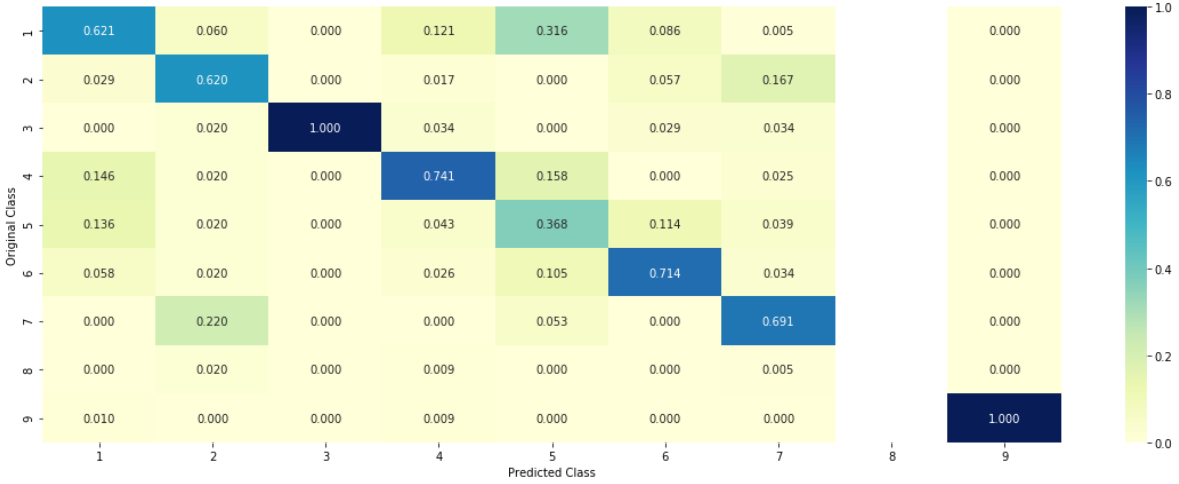

```
For values of best alpha =  0.0001 The train log loss is: 0.43142850820
363027
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0433440991807044
For values of best alpha =  0.0001 The test log loss is: 1.001888093523
85
```

In [93]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, c
v_y, clf)
```

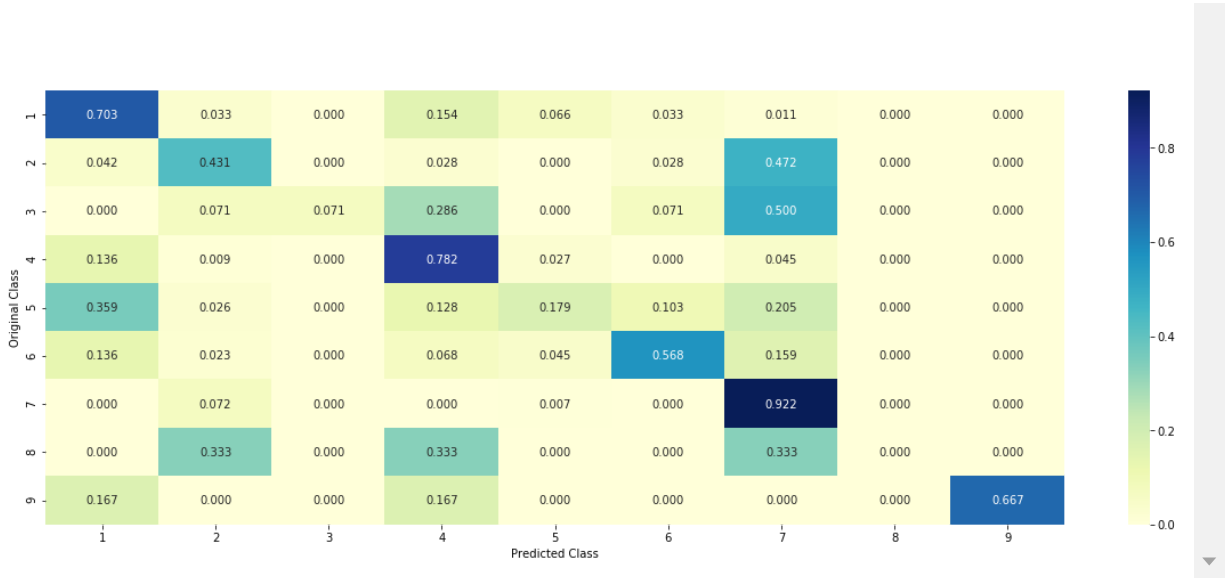
```
Log loss : 1.0433440991807044
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```



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



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



In [94]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[2.520e-02 1.050e-02 3.700e-03 9.900e-0
3 7.290e-02 8.722e-01 1.300e-03
  3.600e-03 8.000e-04]]
Actual Class : 6
--------------------------------------------------
```

```
47 Text feature [12delta] present in test data point [True]
278 Text feature [000] present in test data point [True]
488 Text feature [14] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

In [95]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.326e-01 1.270e-02 4.000e-04 3.320e-0
2 2.400e-03 1.000e-03 6.600e-03
  1.110e-02 1.000e-04]]
Actual Class : 1
--------------------------------------------------
199 Text feature [147] present in test data point [True]
372 Text feature [100] present in test data point [True]
401 Text feature [1640] present in test data point [True]
426 Text feature [120] present in test data point [True]
445 Text feature [18] present in test data point [True]
453 Text feature [10] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.3.2. Without Class balancing

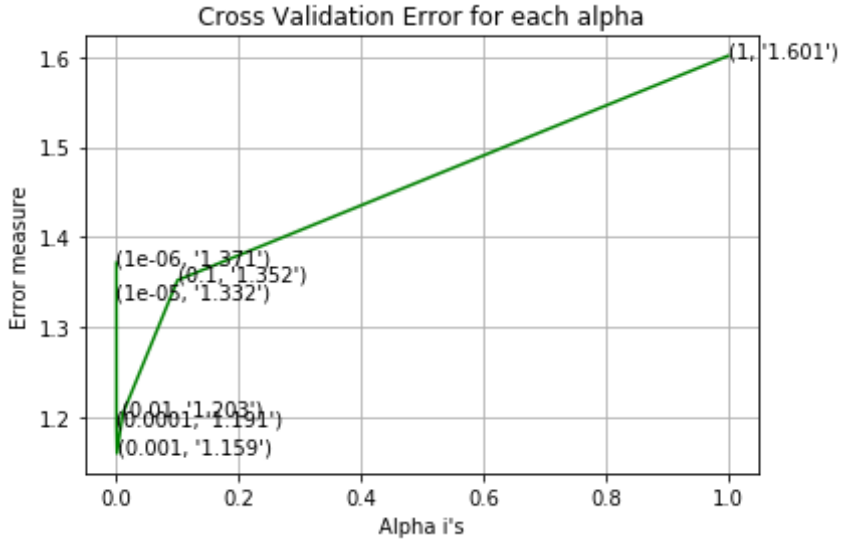
4.3.2.1. Hyper paramter tuning

In [96]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ------------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#------------------------------------
# video link:
#------------------------------------
```

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.3712196239006715
for alpha = 1e-05
Log Loss : 1.3318967900609533
for alpha = 0.0001
Log Loss : 1.1909663793183727
for alpha = 0.001
Log Loss : 1.1594320587164226
for alpha = 0.01
Log Loss : 1.202684128879051
for alpha = 0.1
Log Loss : 1.35191049619177
for alpha = 1
Log Loss : 1.6013550143237567
```



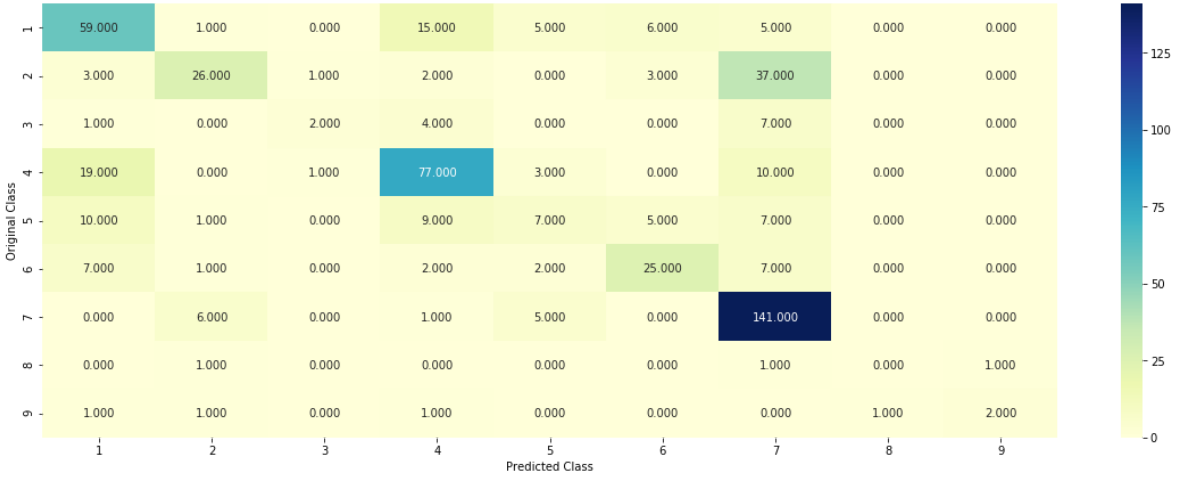

```
For values of best alpha =  0.001 The train log loss is: 0.554182844719
9388
For values of best alpha =  0.001 The cross validation log loss is: 1.1
594320587164226
```

```
5945205871642260
For values of best alpha =  0.001 The test log loss is: 1.0863891194119
446
```

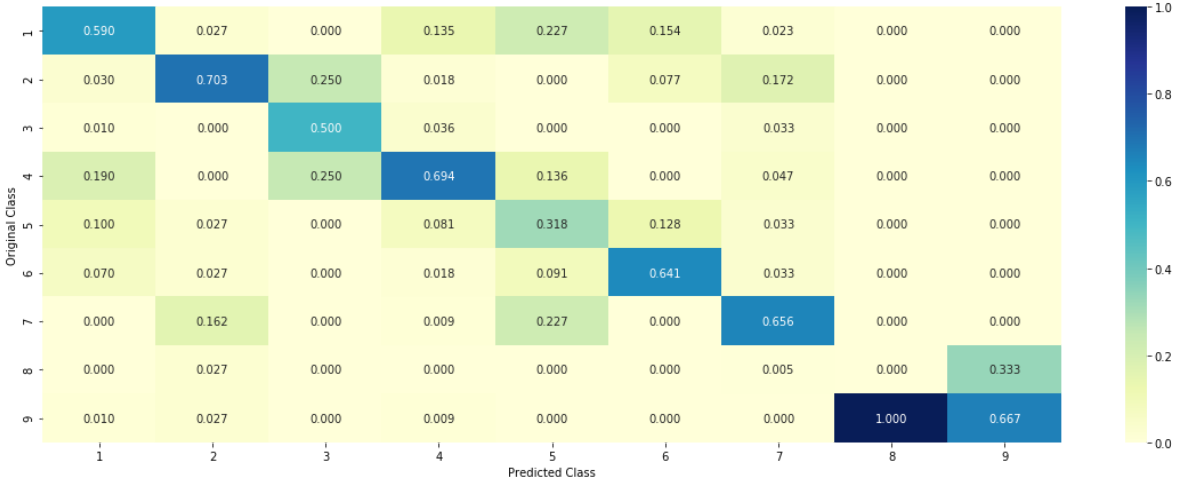
4.3.2.2. Testing model with best hyper parameters

In [97]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

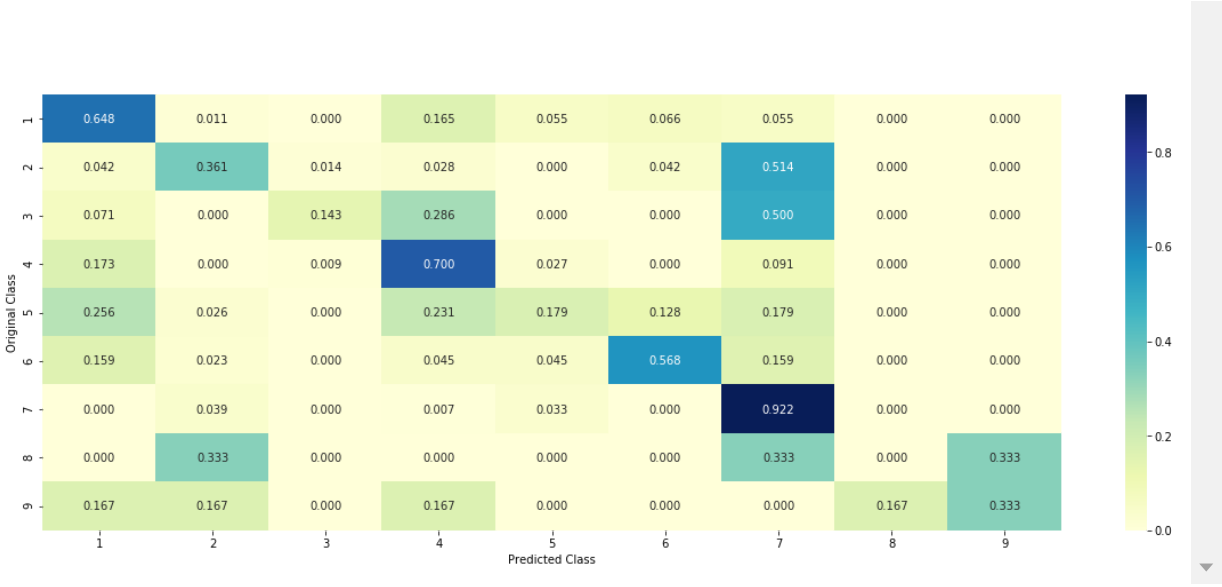
```
Log loss : 1.1594320587164226
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [98]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0396 0.0544 0.0127 0.0386 0.1715 0.6
295 0.045  0.0048 0.0038]]
```

```
Actual Class : 6

--------------------------------------------------
96 Text feature [v1804d] present in test data point [True]
283 Text feature [exosap] present in test data point [True]
378 Text feature [usb] present in test data point [True]
420 Text feature [women] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [99]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[8.111e-01 4.230e-02 1.400e-03 7.930e-0
2 2.200e-03 9.000e-04 6.100e-02
  1.600e-03 3.000e-04]]
Actual Class : 1
--------------------------------------------------
294 Text feature [89] present in test data point [True]
307 Text feature [derepression] present in test data point [True]
318 Text feature [prmt5] present in test data point [True]
419 Text feature [nonsense] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

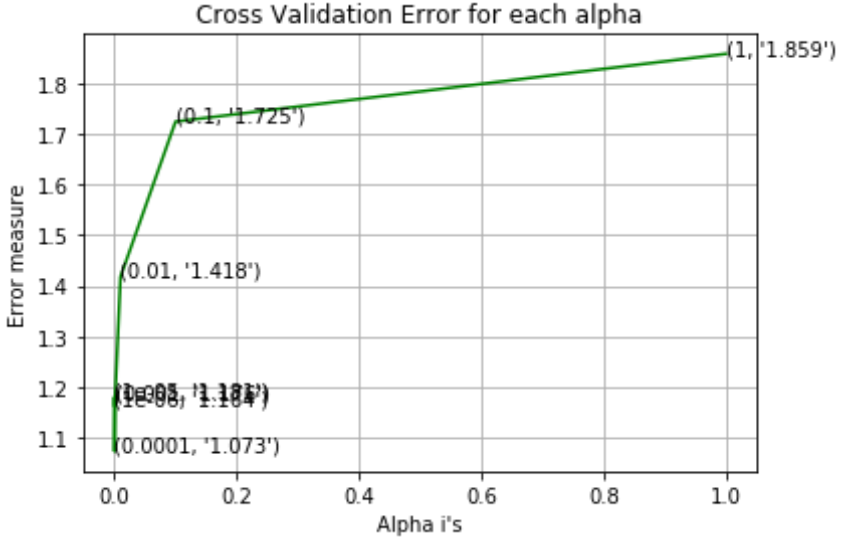
Using TFIDF Vectorization

In [100]:

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.1640220464695044
for alpha = 1e-05
Log Loss : 1.1805896690412814
for alpha = 0.0001
Log Loss : 1.0730416410363148
for alpha = 0.001
Log Loss : 1.1761096499678982
for alpha = 0.01
Log Loss : 1.4180098985872183
for alpha = 0.1
Log Loss : 1.724663472326818
for alpha = 1
Log Loss : 1.8586731719508154
```

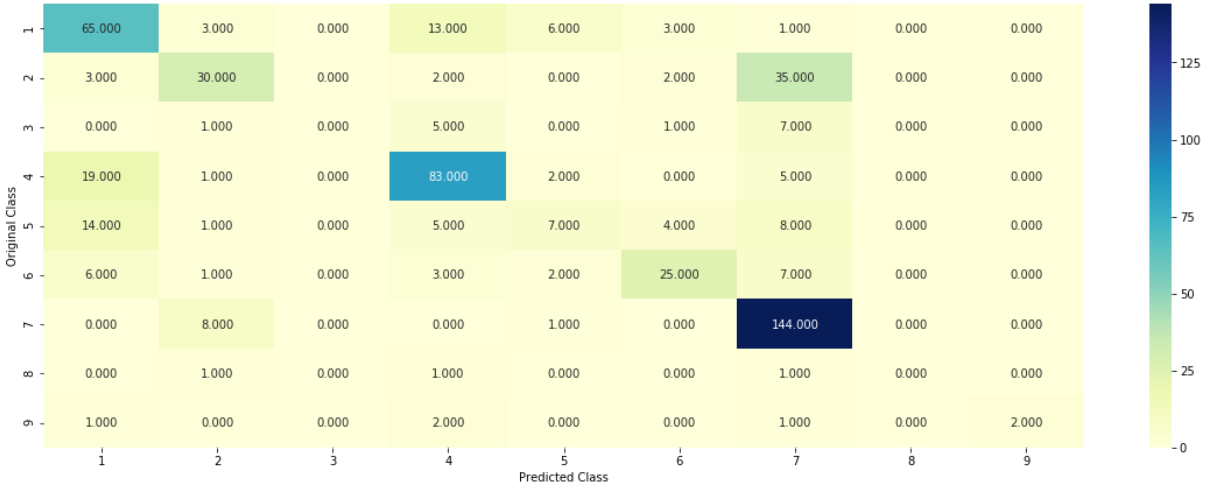



```
For values of best alpha =  0.0001 The train log loss is: 0.42393838661
747824
For values of best alpha =  0.0001 The cross validation log loss is: 1
```

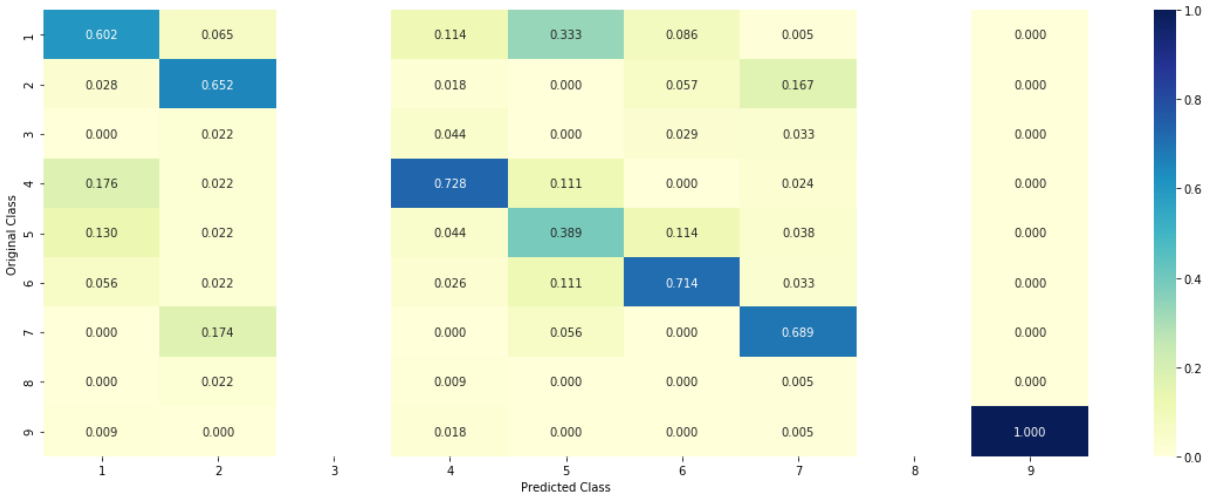
```
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0730416410363148
For values of best alpha =  0.0001 The test log loss is: 1.026168479369
4514
```

In [101]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, c
v_y, clf)
```

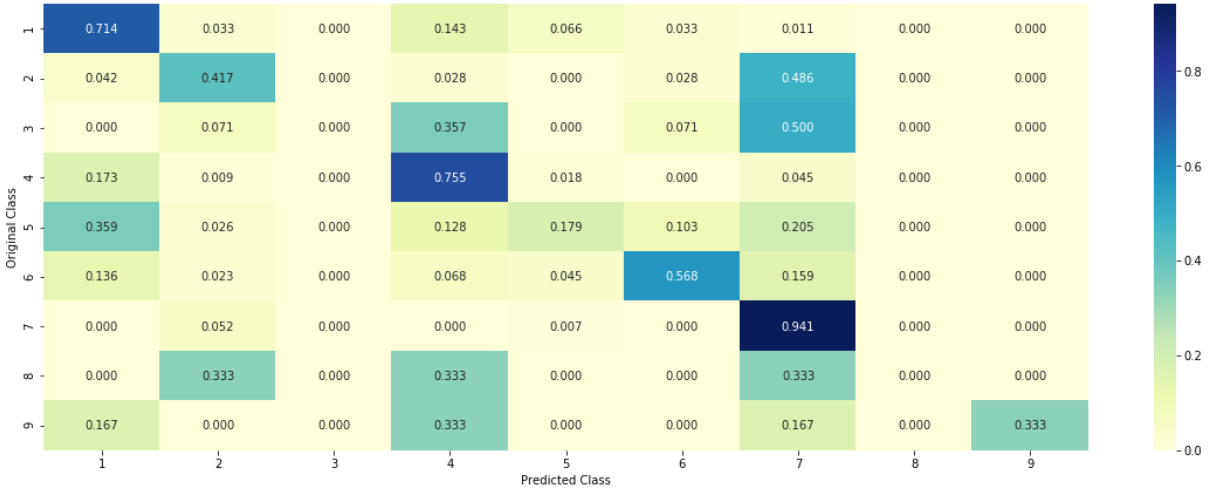
```
Log loss : 1.0730416410363148
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```




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



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



In [102]:
```
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_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[2.540e-02 1.100e-02 2.600e-03 1.170e-0
2 8.140e-02 8.619e-01 1.600e-03
  3.800e-03 5.000e-04]]
Actual Class : 6
--------------------------------------------------
82 Text feature [v1804d] present in test data point [True]
347 Text feature [women] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

In [103]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
```

```
Predicted Class Probabilities: [[9.126e-01 1.310e-02 3.000e-04 4.930e-0
2 1.900e-03 8.000e-04 8.200e-03
  1.380e-02 1.000e-04]]
Actual Class : 1
--------------------------------------------------
368 Text feature [89] present in test data point [True]
370 Text feature [developmentally] present in test data point [True]
382 Text feature [decay] present in test data point [True]
405 Text feature [intragenic] present in test data point [True]
436 Text feature [prmt5] present in test data point [True]
446 Text feature [derepression] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [104]:

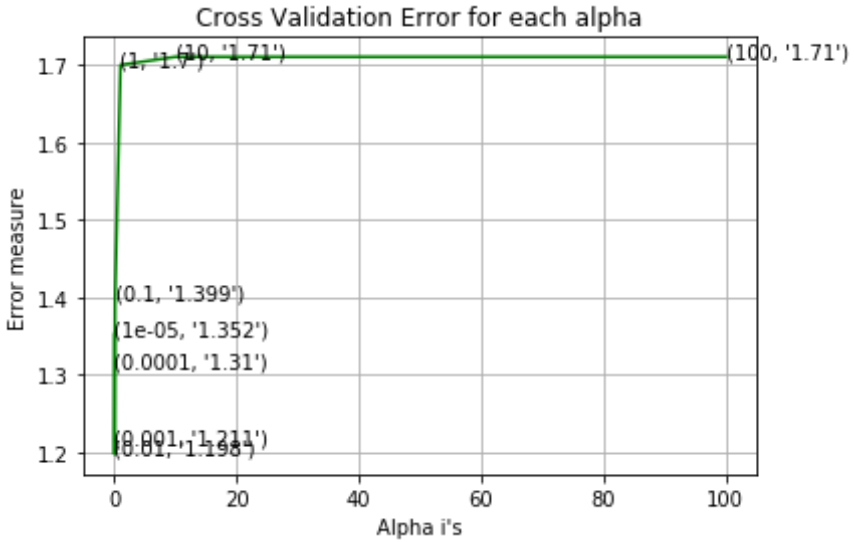
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# ---------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# ---------------------------------
```

```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
```

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for C = 1e-05
Log Loss : 1.3524653326283271
for C = 0.0001
Log Loss : 1.309932479289841
for C = 0.001
Log Loss : 1.210561642559358
for C = 0.01
Log Loss : 1.1976770814502262
for C = 0.1
Log Loss : 1.3985854650123055
for C = 1
Log Loss : 1.6996195024822
```

```
for C = 10
Log Loss : 1.7100970283304024
for C = 100
Log Loss : 1.7100893874839835
```




```
For values of best alpha =  0.01 The train log loss is: 0.7484308987607
594
For values of best alpha =  0.01 The cross validation log loss is: 1.19
76770814502262
For values of best alpha =  0.01 The test log loss is: 1.13138603976880
38
```

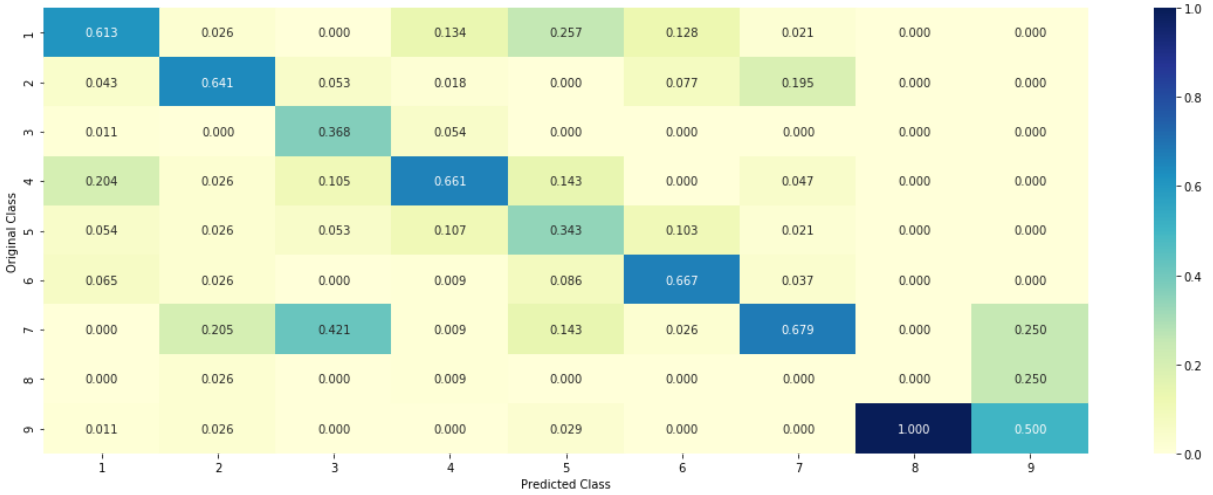
4.4.2. Testing model with best hyper parameters

In [105]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
```

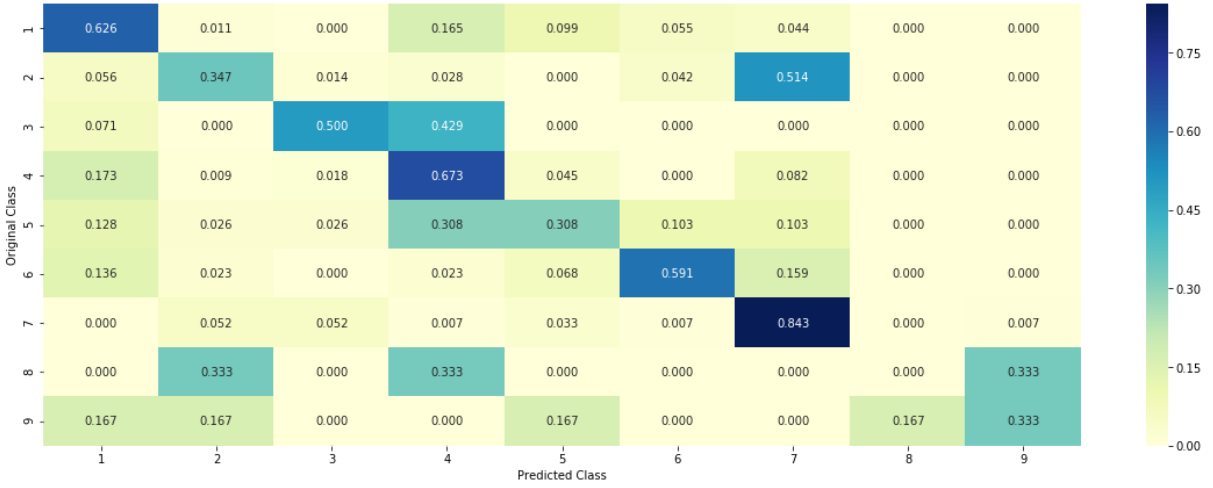
```
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class
_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

```
Log loss : 1.1976770814502262
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [106]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
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_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0521 0.0554 0.0103 0.0554 0.0388 0.7
046 0.0764 0.0035 0.0035]]
Actual Class : 6
--------------------------------------------------
4 Text feature [n1878k] present in test data point [True]
5 Text feature [h1966y] present in test data point [True]
6 Text feature [alphaimager] present in test data point [True]
7 Text feature [ivs23] present in test data point [True]
8 Text feature [s127n] present in test data point [True]
```

```
9 Text feature [12delta] present in test data point [True]
10 Text feature [leandro] present in test data point [True]
11 Text feature [avgd] present in test data point [True]
12 Text feature [a1170v] present in test data point [True]
13 Text feature [innotech] present in test data point [True]
14 Text feature [undercalling] present in test data point [True]
16 Text feature [and6] present in test data point [True]
17 Text feature [cored] present in test data point [True]
18 Text feature [2401] present in test data point [True]
19 Text feature [m784v] present in test data point [True]
20 Text feature [dnastar] present in test data point [True]
21 Text feature [a4] present in test data point [True]
22 Text feature [splicesitefinder] present in test data point [True]
24 Text feature [exosap] present in test data point [True]
25 Text feature [incorrectly] present in test data point [True]
26 Text feature [adaptable] present in test data point [True]
27 Text feature [weighting] present in test data point [True]
78 Text feature [esefinder] present in test data point [True]
85 Text feature [a3] present in test data point [True]
87 Text feature [usb] present in test data point [True]
90 Text feature [assigns] present in test data point [True]
95 Text feature [theoretical] present in test data point [True]
100 Text feature [visually] present in test data point [True]
102 Text feature [i2285v] present in test data point [True]
104 Text feature [and4] present in test data point [True]
108 Text feature [cytokeratins] present in test data point [True]
114 Text feature [ivs2] present in test data point [True]
116 Text feature [3200] present in test data point [True]
117 Text feature [pursuing] present in test data point [True]
187 Text feature [kconfab] present in test data point [True]
195 Text feature [histopathology] present in test data point [True]
198 Text feature [brca] present in test data point [True]
211 Text feature [causation] present in test data point [True]
214 Text feature [e2856a] present in test data point [True]
220 Text feature [a5] present in test data point [True]
221 Text feature [and3] present in test data point [True]
223 Text feature [likelihoods] present in test data point [True]
224 Text feature [salt] present in test data point [True]
232 Text feature [ivs13] present in test data point [True]
420 Text feature [histopathologic] present in test data point [True]
```

```
475 Text feature [unclassified] present in test data point [True]
Out of the top  500  features  46 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [107]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

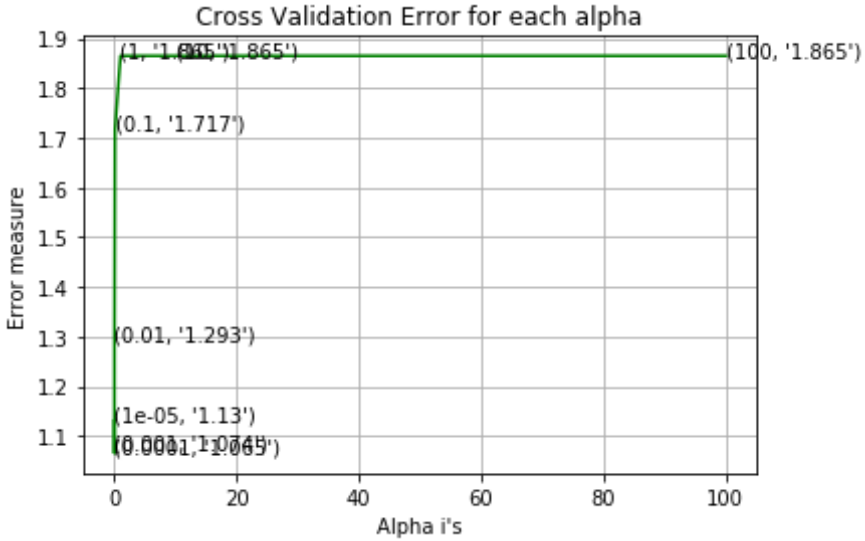
```
Predicted Class : 1
Predicted Class Probabilities: [[0.699  0.0625 0.0042 0.1233 0.0109 0.0
069 0.0904 0.0015 0.0012]]
Actual Class : 1
--------------------------------------------------
71 Text feature [derepression] present in test data point [True]
87 Text feature [prmt5] present in test data point [True]
125 Text feature [89] present in test data point [True]
179 Text feature [binomial] present in test data point [True]
235 Text feature [axin] present in test data point [True]
313 Text feature [truncating] present in test data point [True]
330 Text feature [speech] present in test data point [True]
372 Text feature [repress] present in test data point [True]
380 Text feature [axin2] present in test data point [True]
382 Text feature [deficient] present in test data point [True]
457 Text feature [duolink] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

Using TFIDF Vectorization

In [108]:
```
alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='bal
anced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2'
, loss='hinge', random_state=42)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
```

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

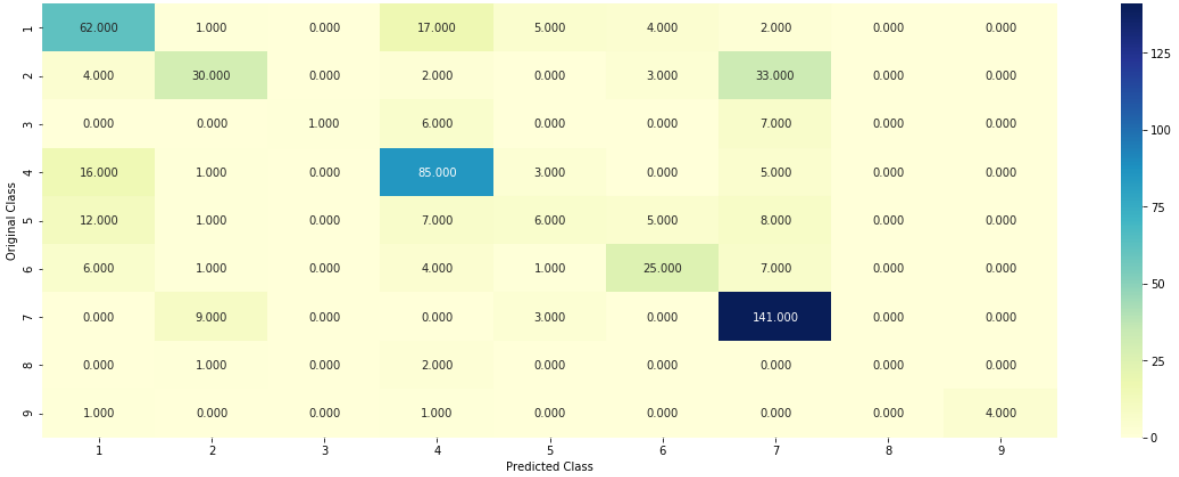
```
for C = 1e-05
Log Loss : 1.130239775252215
for C = 0.0001
Log Loss : 1.0649371520100428
for C = 0.001
Log Loss : 1.0741679253642669
for C = 0.01
Log Loss : 1.2929371691645177
for C = 0.1
Log Loss : 1.716997105707117
for C = 1
Log Loss : 1.8650496643345962
for C = 10
Log Loss : 1.8650443645748
for C = 100
Log Loss : 1.8650262122555723
```



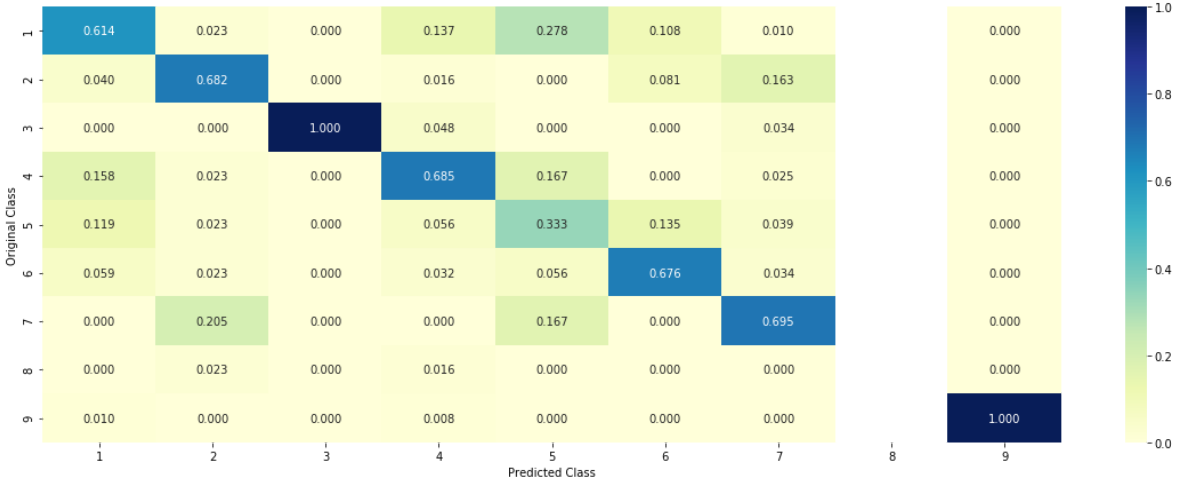

```
For values of best alpha =  0.0001 The train log loss is: 0.392355434
1837163
For values of best alpha =  0.0001 The cross validation log loss is:
1.0649371520100428
For values of best alpha =  0.0001 The test log loss is: 1.0215302342
055144
```

In [109]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_
y, clf)
```

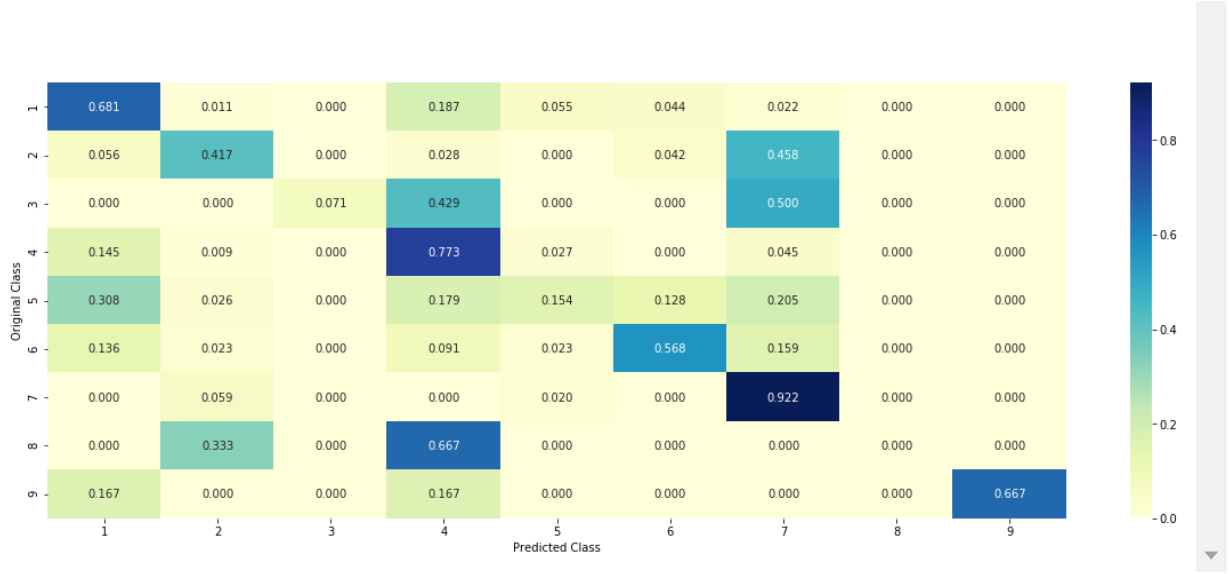
```
Log loss : 1.0649371520100428
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```



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



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



In [110]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[4.850e-02 3.910e-02 9.800e-03 4.030e-0
2 3.310e-02 8.208e-01 4.600e-03
  3.100e-03 8.000e-04]]
Actual Class : 6
```

```
----------------------------------------------------
84 Text feature [12delta] present in test data point [True]
272 Text feature [000] present in test data point [True]
438 Text feature [15] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

In [111]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.267e-01 2.910e-02 2.600e-03 1.800e-0
2 4.300e-03 2.500e-03 8.900e-03
  7.700e-03 3.000e-04]]
Actual Class : 1
----------------------------------------------------
249 Text feature [1640] present in test data point [True]
319 Text feature [147] present in test data point [True]
323 Text feature [100] present in test data point [True]
362 Text feature [18] present in test data point [True]
446 Text feature [120] present in test data point [True]
450 Text feature [10] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [112]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# ------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
```

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

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

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

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
```

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2514601128449547
for n_estimators = 100 and max depth =  10
Log Loss : 1.1958214596110226
for n_estimators = 200 and max depth =  5
Log Loss : 1.2355454274999174
for n_estimators = 200 and max depth =  10
Log Loss : 1.1899014074698653
for n_estimators = 500 and max depth =  5
Log Loss : 1.2262114372600346
for n_estimators = 500 and max depth =  10
Log Loss : 1.187222026148504
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2275393772781482
for n_estimators = 1000 and max depth =  10
Log Loss : 1.184053917787347
for n_estimators = 2000 and max depth =  5
Log Loss : 1.22707114545245
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1850439989840387
For values of best estimator =  1000 The train log loss is: 0.711978609
6183438
For values of best estimator =  1000 The cross validation log loss is:
1.1840539177873473
```

```
For values of best estimator =  1000 The test log loss is: 1.1348978450
979081
```

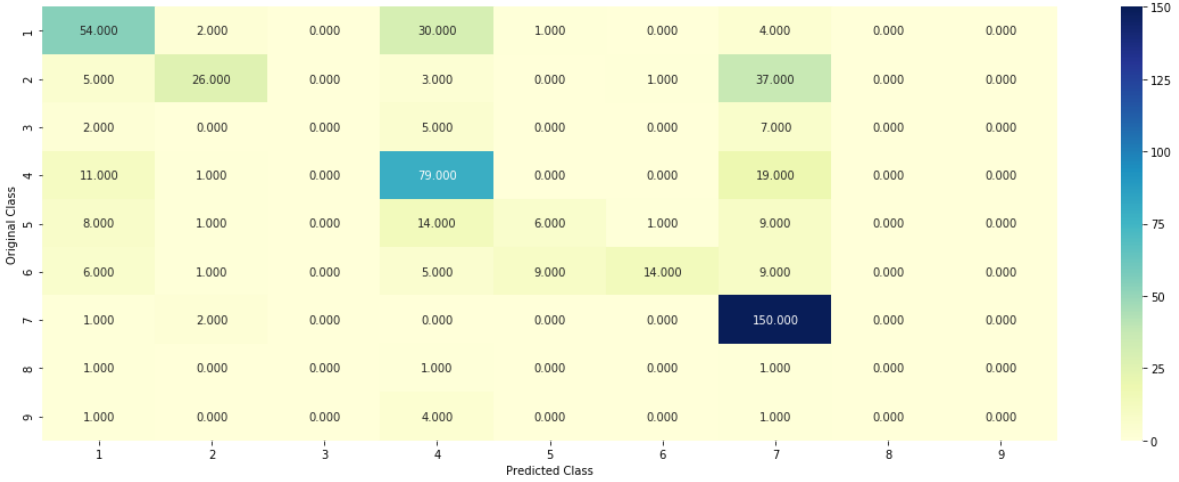
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [113]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# ------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

```
Log loss : 1.1840539177873473
Number of mis-classified points : 0.3815789473684211
```

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




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


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

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [114]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
```

```
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0464 0.0193 0.0129 0.0368 0.0872 0.7
741 0.0158 0.0029 0.0046]]
Actual Class : 6
--------------------------------------------------
7 Text feature [function] present in test data point [True]
8 Text feature [missense] present in test data point [True]
11 Text feature [receptor] present in test data point [True]
12 Text feature [cells] present in test data point [True]
15 Text feature [loss] present in test data point [True]
17 Text feature [growth] present in test data point [True]
22 Text feature [treatment] present in test data point [True]
24 Text feature [brca1] present in test data point [True]
28 Text feature [deleterious] present in test data point [True]
29 Text feature [brca2] present in test data point [True]
42 Text feature [treated] present in test data point [True]
43 Text feature [variants] present in test data point [True]
48 Text feature [functional] present in test data point [True]
50 Text feature [truncating] present in test data point [True]
52 Text feature [patients] present in test data point [True]
53 Text feature [pathogenic] present in test data point [True]
55 Text feature [therapeutic] present in test data point [True]
61 Text feature [cell] present in test data point [True]
64 Text feature [clinical] present in test data point [True]
71 Text feature [lines] present in test data point [True]
73 Text feature [carriers] present in test data point [True]
77 Text feature [protein] present in test data point [True]
79 Text feature [variant] present in test data point [True]
83 Text feature [factor] present in test data point [True]
95 Text feature [favor] present in test data point [True]
Out of the top  100  features  25 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [115]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3775 0.1338 0.0191 0.1881 0.049  0.0
418 0.1722 0.0086 0.0099]]
Actuall Class : 1
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [activation] present in test data point [True]
7 Text feature [function] present in test data point [True]
8 Text feature [missense] present in test data point [True]
10 Text feature [suppressor] present in test data point [True]
11 Text feature [receptor] present in test data point [True]
12 Text feature [cells] present in test data point [True]
13 Text feature [nonsense] present in test data point [True]
14 Text feature [downstream] present in test data point [True]
15 Text feature [loss] present in test data point [True]
17 Text feature [growth] present in test data point [True]
18 Text feature [oncogenic] present in test data point [True]
22 Text feature [treatment] present in test data point [True]
25 Text feature [constitutive] present in test data point [True]
26 Text feature [months] present in test data point [True]
30 Text feature [therapy] present in test data point [True]
35 Text feature [defective] present in test data point [True]
```

```
37 Text feature [signaling] present in test data point [True]
38 Text feature [frameshift] present in test data point [True]
40 Text feature [serum] present in test data point [True]
42 Text feature [treated] present in test data point [True]
43 Text feature [variants] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
48 Text feature [functional] present in test data point [True]
50 Text feature [truncating] present in test data point [True]
51 Text feature [survival] present in test data point [True]
52 Text feature [patients] present in test data point [True]
53 Text feature [pathogenic] present in test data point [True]
55 Text feature [therapeutic] present in test data point [True]
58 Text feature [proliferation] present in test data point [True]
60 Text feature [inhibited] present in test data point [True]
61 Text feature [cell] present in test data point [True]
64 Text feature [clinical] present in test data point [True]
65 Text feature [expression] present in test data point [True]
71 Text feature [lines] present in test data point [True]
72 Text feature [activate] present in test data point [True]
74 Text feature [sequencing] present in test data point [True]
77 Text feature [protein] present in test data point [True]
80 Text feature [proteins] present in test data point [True]
83 Text feature [factor] present in test data point [True]
85 Text feature [mammalian] present in test data point [True]
87 Text feature [antibodies] present in test data point [True]
89 Text feature [starved] present in test data point [True]
92 Text feature [trial] present in test data point [True]
94 Text feature [amplification] present in test data point [True]
95 Text feature [favor] present in test data point [True]
96 Text feature [response] present in test data point [True]
98 Text feature [oncogene] present in test data point [True]
99 Text feature [abolish] present in test data point [True]
Out of the top  100  features  52 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [116]:
```
# ------------------------------
```

```
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
```

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

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

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

```
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tra
in log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cro
ss validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.129424223738912
for n_estimators = 10 and max depth =  3
Log Loss : 1.6476602395905584
for n_estimators = 10 and max depth =  5
Log Loss : 1.4281894850304124
for n_estimators = 10 and max depth =  10
Log Loss : 1.747471290020896
for n_estimators = 50 and max depth =  2
Log Loss : 1.55991354977933
for n_estimators = 50 and max depth =  3
Log Loss : 1.386483352308967
for n_estimators = 50 and max depth =  5
Log Loss : 1.3549118903028194
for n_estimators = 50 and max depth =  10
Log Loss : 1.5722997075687621
for n_estimators = 100 and max depth =  2
Log Loss : 1.504586961534706
for n_estimators = 100 and max depth =  3
Log Loss : 1.4322918761376464
for n_estimators = 100 and max depth =  5
Log Loss : 1.3199256638835044
for n_estimators = 100 and max depth =  10
Log Loss : 1.6091149836453043
for n_estimators = 200 and max depth =  2
```

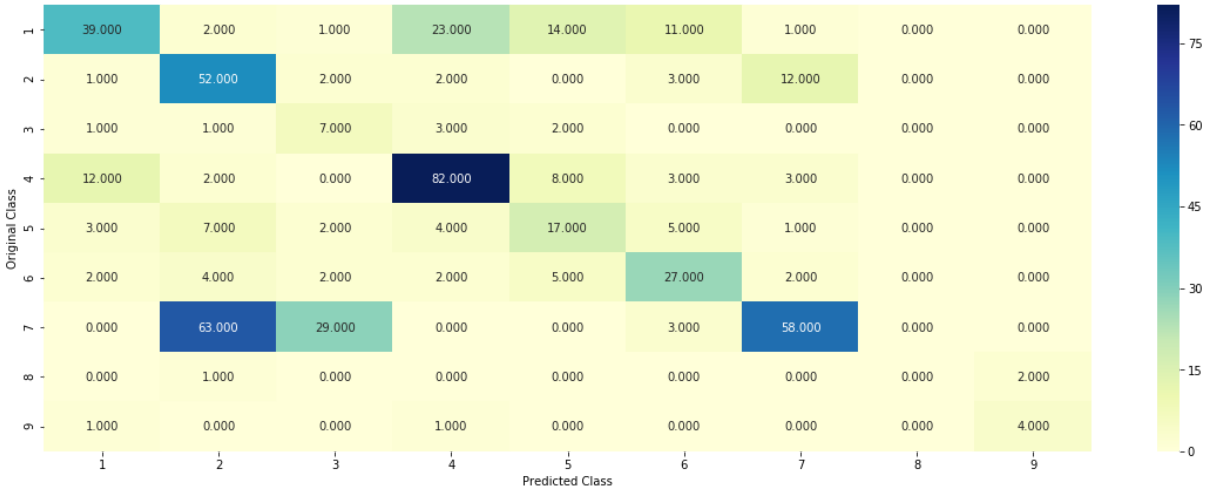
```
Log Loss : 1.5332534138123461
for n_estimators = 200 and max depth =  3
Log Loss : 1.4243627650206774
for n_estimators = 200 and max depth =  5
Log Loss : 1.3598662453572712
for n_estimators = 200 and max depth =  10
Log Loss : 1.597285793071707
for n_estimators = 500 and max depth =  2
Log Loss : 1.5882951801299792
for n_estimators = 500 and max depth =  3
Log Loss : 1.5006054248099145
for n_estimators = 500 and max depth =  5
Log Loss : 1.3844847911904787
for n_estimators = 500 and max depth =  10
Log Loss : 1.5938894951729325
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5813369358004472
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5146389296516813
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3688421391370122
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6195175471329712
For values of best alpha =  100 The train log loss is: 0.05698947423814
736
For values of best alpha =  100 The cross validation log loss is: 1.319
9256638835044
For values of best alpha =  100 The test log loss is: 1.285115789869111
5
```

4.5.4. Testing model with best hyper parameters (Response Coding)

In [117]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
```

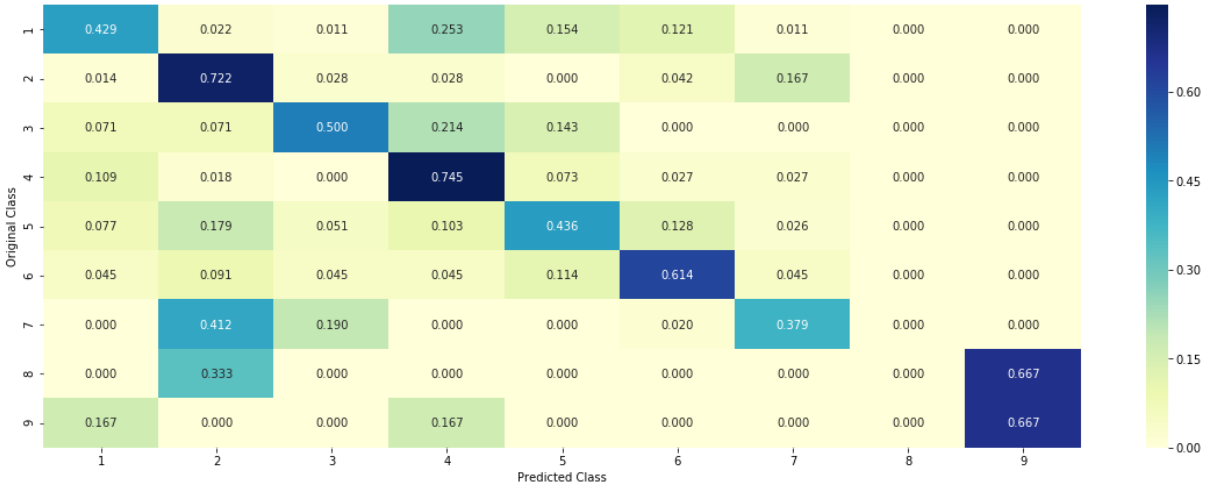
```
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# --------------------------------

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

```
Log loss : 1.3199256638835044
Number of mis-classified points : 0.462406015037594
-------------------- Confusion matrix --------------------
```



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

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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [118]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
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 : 6
Predicted Class Probabilities: [[0.0199 0.004  0.0187 0.0132 0.2294 0.7
011 0.0035 0.0041 0.0062]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
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
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
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
```

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

4.5.5.2. Incorrectly Classified point

In [119]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7682 0.0036 0.0018 0.2047 0.0014 0.0
099 0.0027 0.003  0.0047]]
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
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

Using TFIDF Vectorizer

In [120]:

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

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

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

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

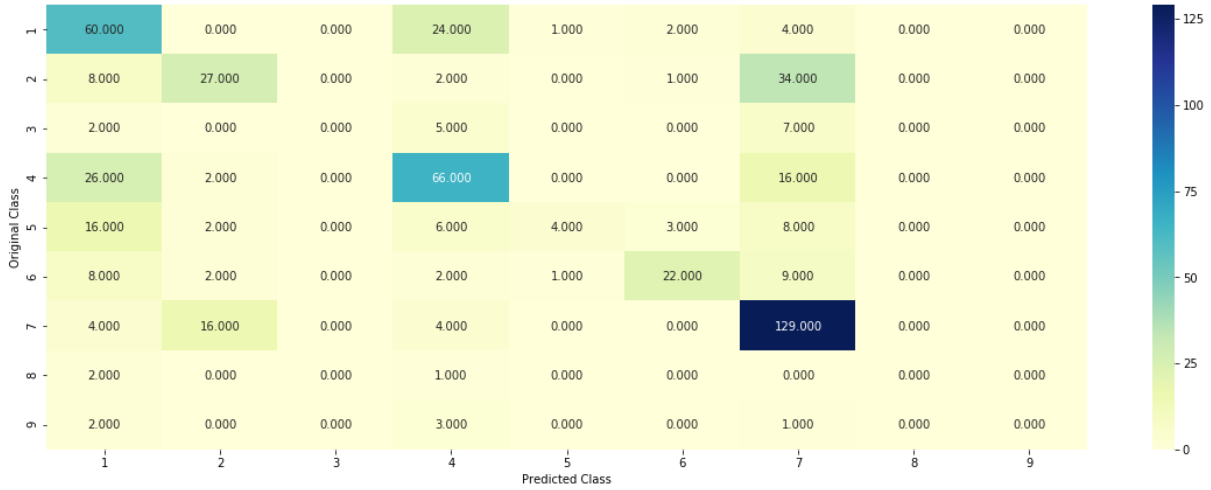
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2026311558309404
for n_estimators = 100 and max depth =  10
Log Loss : 1.226155628407084
for n_estimators = 200 and max depth =  5
```

```
Log Loss : 1.1940574563856394
for n_estimators = 200 and max depth =  10
Log Loss : 1.2204593904836099
for n_estimators = 500 and max depth =  5
Log Loss : 1.1898083994161508
for n_estimators = 500 and max depth =  10
Log Loss : 1.2100597482610742
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1857336474627833
for n_estimators = 1000 and max depth =  10
Log Loss : 1.206745102176953
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1838005585546907
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2037384832512172
For values of best estimator =  2000 The train log loss is: 0.870929081
2683479
For values of best estimator =  2000 The cross validation log loss is:
1.1838005585546905
For values of best estimator =  2000 The test log loss is: 1.1512724618
32149
```

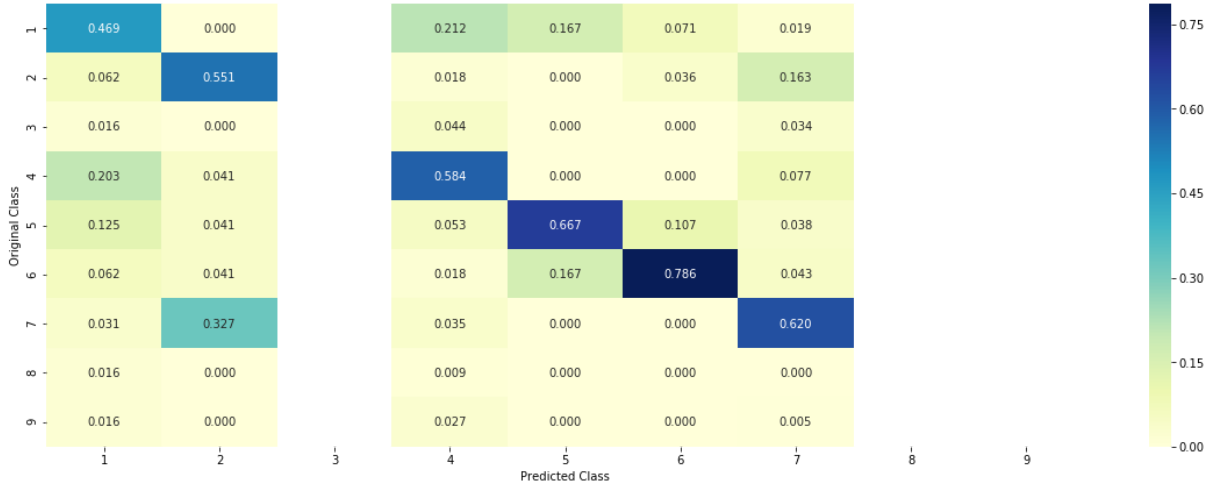
In [121]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_
y, clf)
```

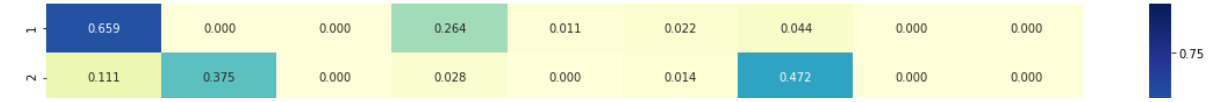
```
Log loss : 1.1838005585546905
Number of mis-classified points : 0.42105263157894735
-------------------- Confusion matrix --------------------
```

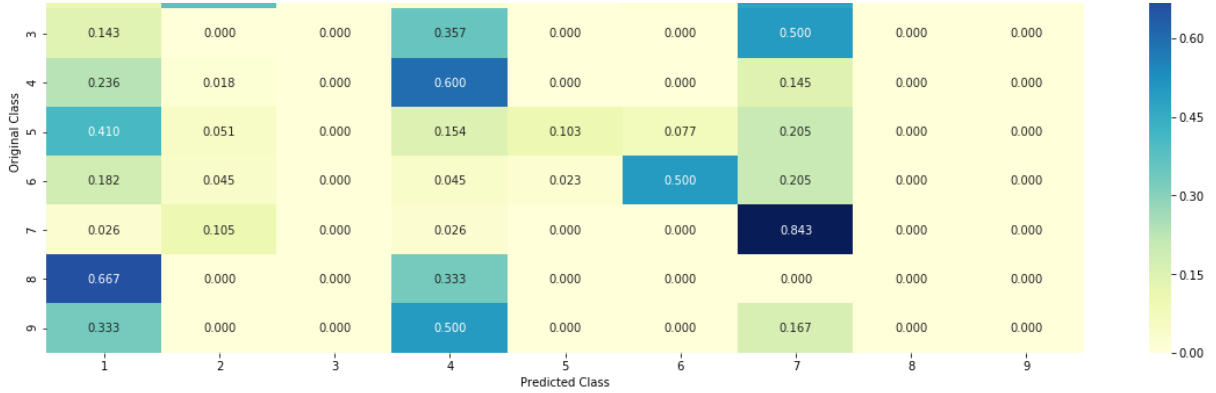


---------- Precision matrix (Columm Sum=1) ----------



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





In [122]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
```

```
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0442 0.011  0.0074 0.0263 0.1404 0.7
534 0.0151 0.0011 0.001 ]]
Actual Class : 6
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

In [123]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_tfidf[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4349 0.1641 0.0159 0.1071 0.0506 0.0
481 0.1622 0.0093 0.0078]]
Actuall Class : 1
--------------------------------------------------
2 Text feature [1c] present in test data point [True]
91 Text feature [169] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [124]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/geometric-intuition-1/
#------------------------------


# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# ------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
```

```
# --------------------------------


# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFo
restClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# --------------------------------


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight=
'balanced', random_state=0)
```

```
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

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

```
Logistic Regression :  Log Loss: 1.14
Support vector machines : Log Loss: 1.70
Naive Bayes : Log Loss: 1.30
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.039
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.530
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.148
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.271
```

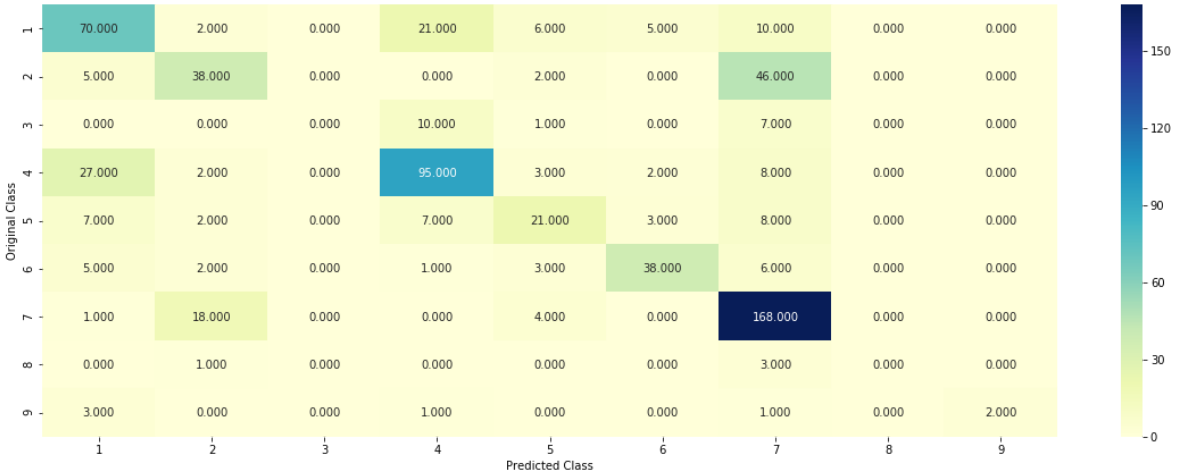
```
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.608
```

4.7.2 testing the model with the best hyper parameters

In [125]:

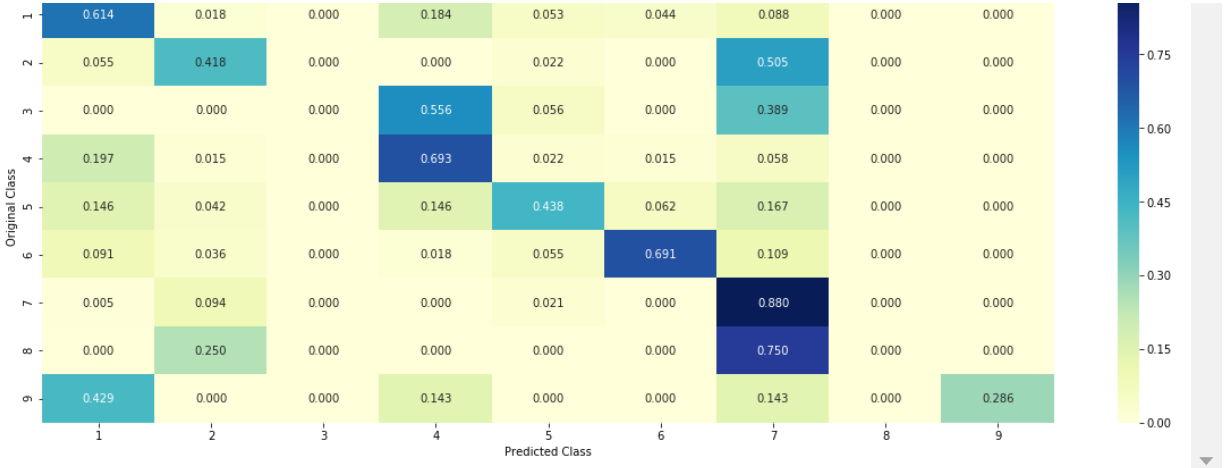
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.648898168223025
Log loss (CV) on the stacking classifier : 1.1481560289727637
Log loss (test) on the stacking classifier : 1.1025907468936813
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```



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

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



Using TFIDF Classifier

In [126]:

```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(train_x_tfidf, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

sig_clf1.fit(train_x_tfidf, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_cl
f1.predict_proba(cv_x_tfidf))))
sig_clf2.fit(train_x_tfidf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig
```

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

```
Logistic Regression :  Log Loss: 1.07
Support vector machines : Log Loss: 1.87
Naive Bayes : Log Loss: 1.21
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.488
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.152
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.353
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.778
```

In [127]:

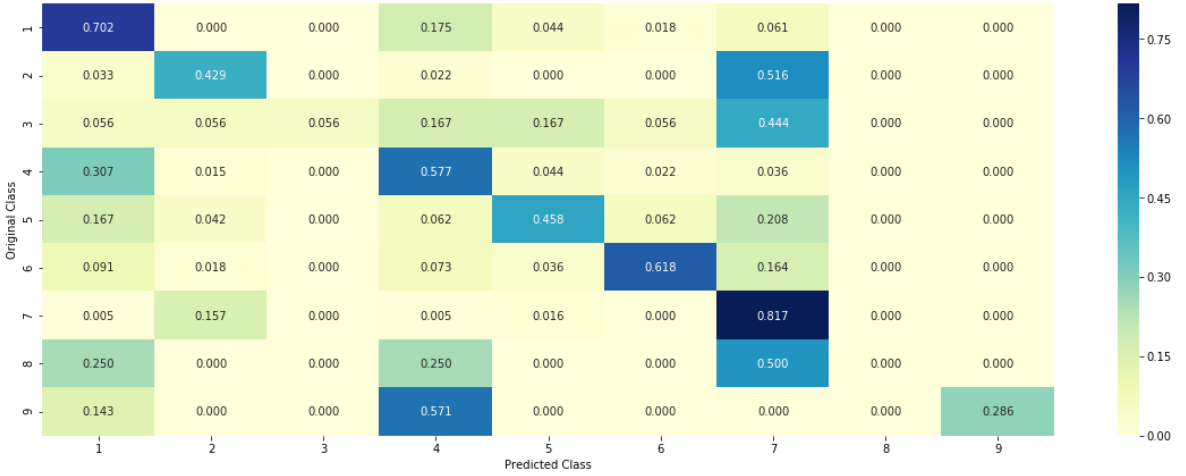
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_tfidf, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5919130617064979
Log loss (CV) on the stacking classifier : 1.151713659300392
Log loss (test) on the stacking classifier : 1.1449905134924137
Number of missclassified point : 0.37894736842105264
-------------------- Confusion matrix --------------------
```


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

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



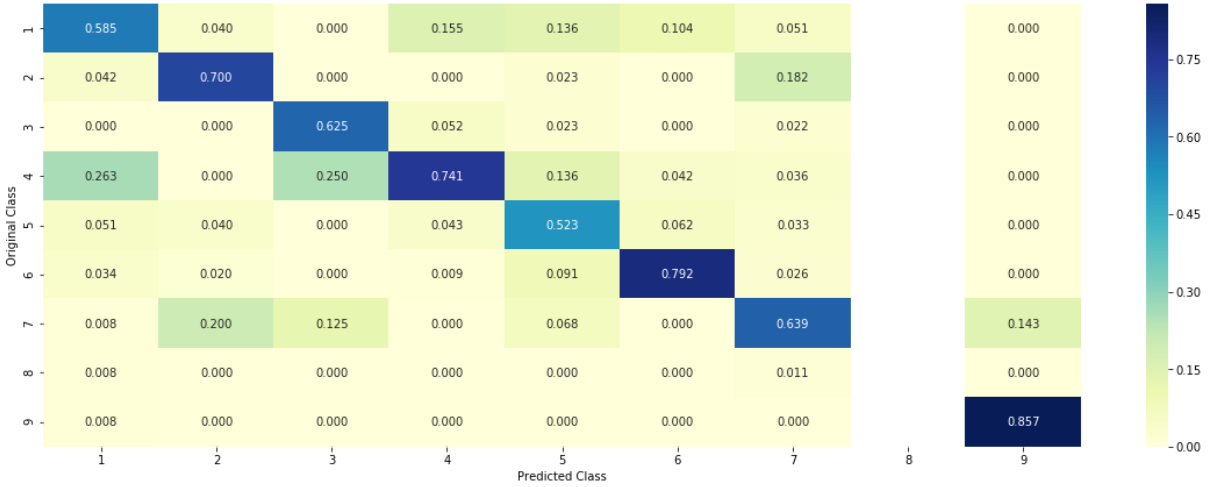
4.7.3 Maximum Voting classifier

In [128]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
```

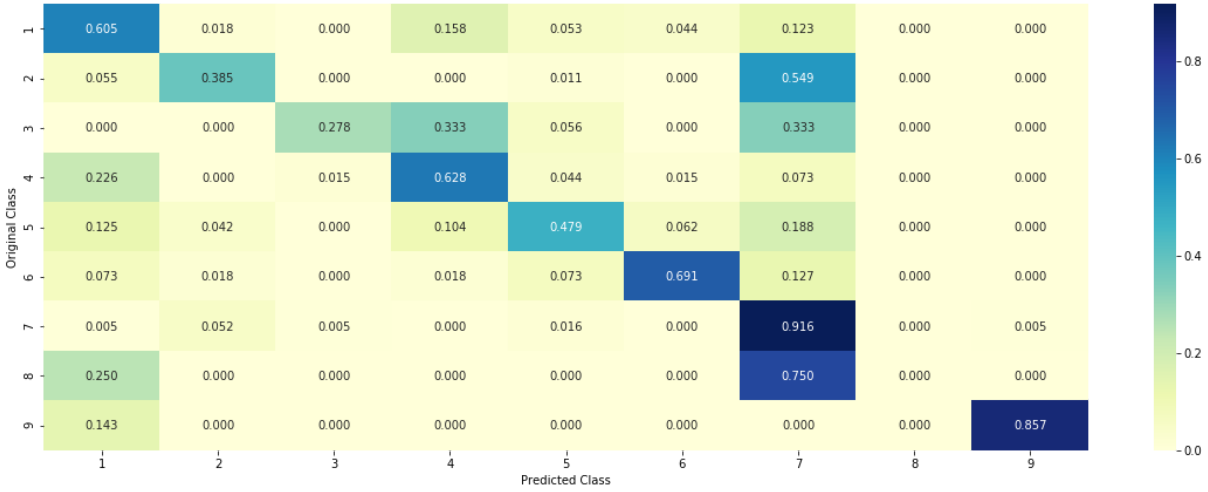
```
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, v
clf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier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

```
Log loss (train) on the VotingClassifier : 0.9032715547282626
Log loss (CV) on the VotingClassifier : 1.227586053098921
Log loss (test) on the VotingClassifier : 1.2115724234187268
Number of missclassified point : 0.34285714285714286
-------------------- Confusion matrix --------------------
```


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



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



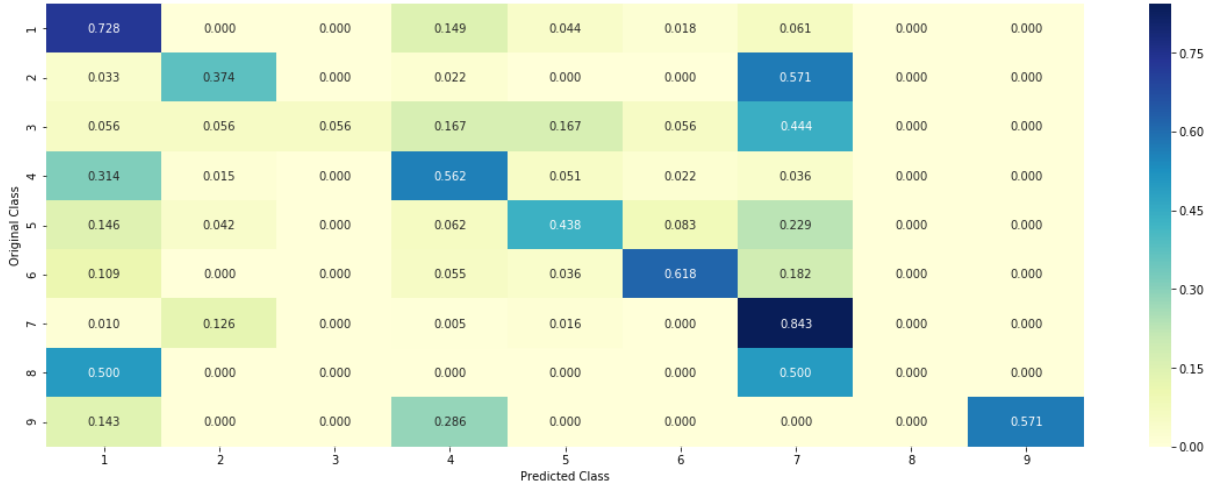
Using TFIDF Vectorizer

In [129]:
```
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_tfidf, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, v
clf.predict_proba(train_x_tfidf)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.pr
edict_proba(cv_x_tfidf)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vcl
f.predict_proba(test_x_tfidf)))
print("Number of missclassified point :", np.count_nonzero((vclf.predic
t(test_x_tfidf)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfid
f))
```

```
Log loss (train) on the VotingClassifier : 0.8509025957559109
Log loss (CV) on the VotingClassifier : 1.1957900163404234
Log loss (test) on the VotingClassifier : 1.1868023715547835
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```


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

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



In [2]:
```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names=(['Vectorizer','Hyperparameter','Model','Test Score'])
x.add_row(['One hot coding',0.1,'Naive Bayes',1.28])
x.add_row(['TFIDF',0.00001,'Naive Bayes',1.20])
```

```
x.add_row(['One hot coding',11,'KNN',1.02])
x.add_row(['TFIDF',5,'KNN',1.099])
x.add_row(['One hot coding',0.001,'Logistic Regression with class balan
cing',1.13])
x.add_row(['TFIDF',0.0001,'Logistic Regression with class balancing',1.
04])
x.add_row(['N-Gram',10,'Logistic Regression with class balancing',1.14
])
x.add_row(['One hot coding',0.01,'SVM',1.19])
x.add_row(['TFIDF',0.0001,'SVM',1.06])
x.add_row(['One hot coding',1000,'Random Forests',1.18])
x.add_row(['TFIDF',2000,'Random Forests',1.18])
x.add_row(['One hot coding',0.1,'Stacking the models',1.14])
x.add_row(['TFIDF',0.1,'Stacking the models',1.144])
x.add_row(['One hot coding',0,'Max Voting Classifier',1.21])
x.add_row(['TFIDF',0,'Max Voting Classifier',1.18])
print(x)
```

```
+---------------+---------------+-----------------------------------
------+------------+
|   Vectorizer   | Hyperparameter |                  Model
      | Test Score |
+---------------+---------------+-----------------------------------
------+------------+
| One hot coding |      0.1       |               Naive Bayes
      |    1.28    |
|     TFIDF      |     1e-05      |               Naive Bayes
      |    1.2     |
| One hot coding |       11       |                   KNN
      |    1.02    |
|     TFIDF      |       5        |                   KNN
      |   1.099    |
| One hot coding |     0.001      | Logistic Regression with class bala
ncing |    1.13    |
|     TFIDF      |     0.0001     | Logistic Regression with class bala
ncing |    1.04    |
|     N-Gram     |       10       | Logistic Regression with class bala
ncing |    1.14    |
| One hot coding |      0.01      |                   SVM
      |    1.19    |
```

```
|     TFIDF      |     0.0001     |                  SVM
      |    1.06    |
| One hot coding |      1000      |              Random Forests
      |    1.18    |
|     TFIDF      |      2000      |              Random Forests
      |    1.18    |
| One hot coding |      0.1       |           Stacking the models
      |    1.14    |
|     TFIDF      |      0.1       |           Stacking the models
      |   1.144    |
| One hot coding |       0        |          Max Voting Classifier
      |    1.21    |
|     TFIDF      |       0        |          Max Voting Classifier
      |    1.18    |
+----------------+----------------+-----------------------------------
------+------------+
```