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

Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

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
```

```
Using TensorFlow backend.
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:516:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:517:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:518:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
```

```
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:520:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/ubuntu/anaconda3/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:525:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
WARNING: Logging before flag parsing goes to stderr.
W0825 14:16:31.745189 139996139153216 __init__.py:308] Limited tf.compat.v2.summary API due to mis
sing TensorBoard installation.
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv('training/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/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skip
rows=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
stop_words = set(stopwords.words('english'))


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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 208.495928 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...

	ID	Gene	Variation	Class	TEXT
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 of output varaible 'y_true'
[stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2
)
# split the train data into train and cross validation by maintaining same distribution of output
varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=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])
```

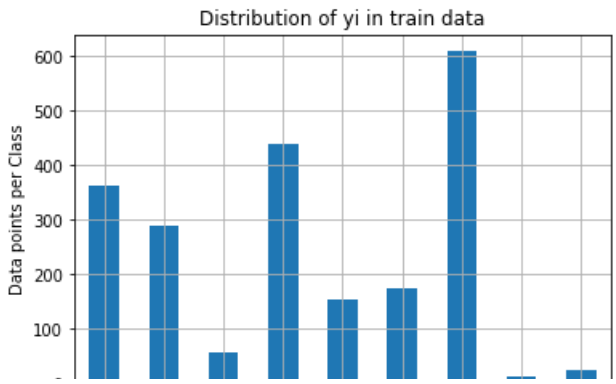
```
Number of data points in train data: 2124
```

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

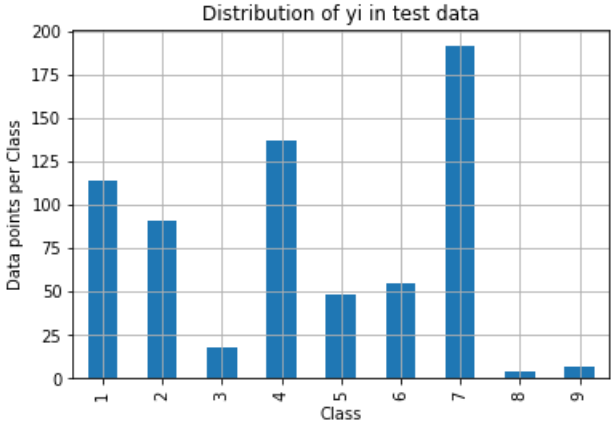
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:

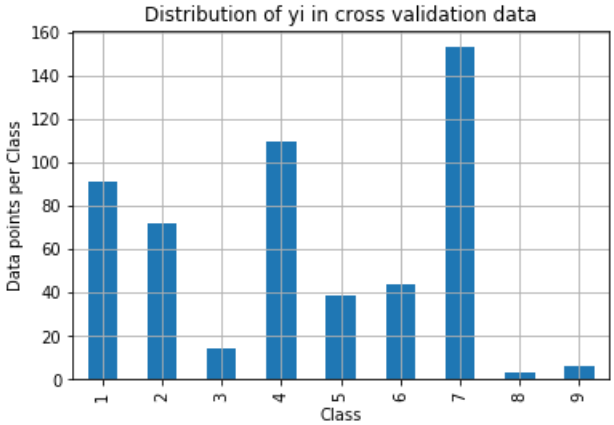
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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


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

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

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

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




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




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




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

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]:

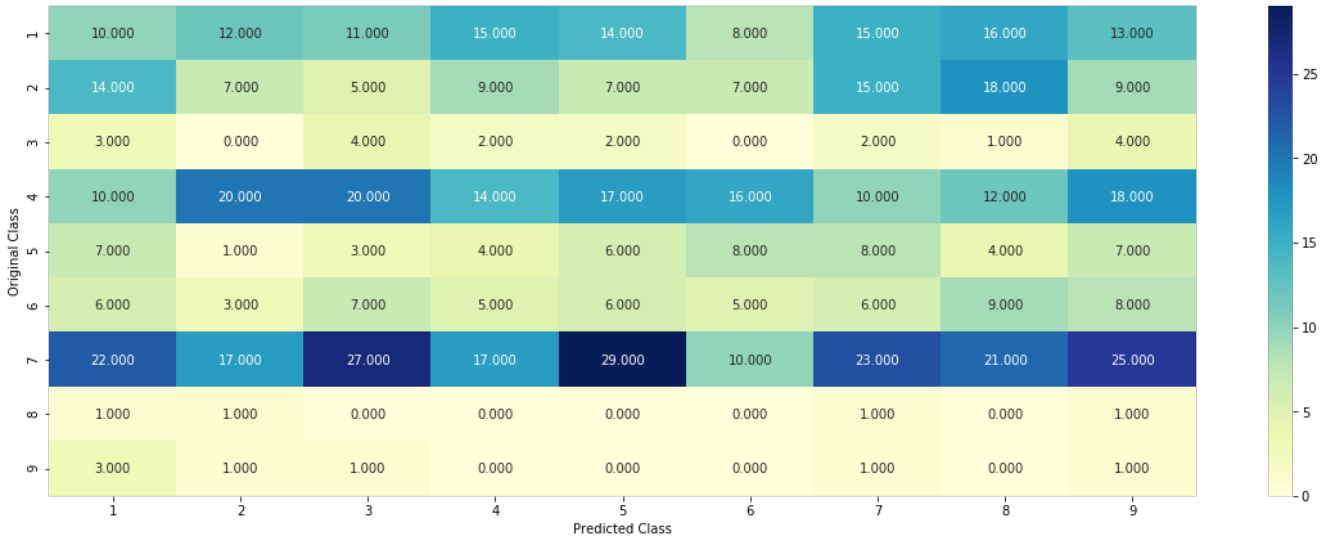
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements 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 corresponds 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 elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in 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=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

In [14]:

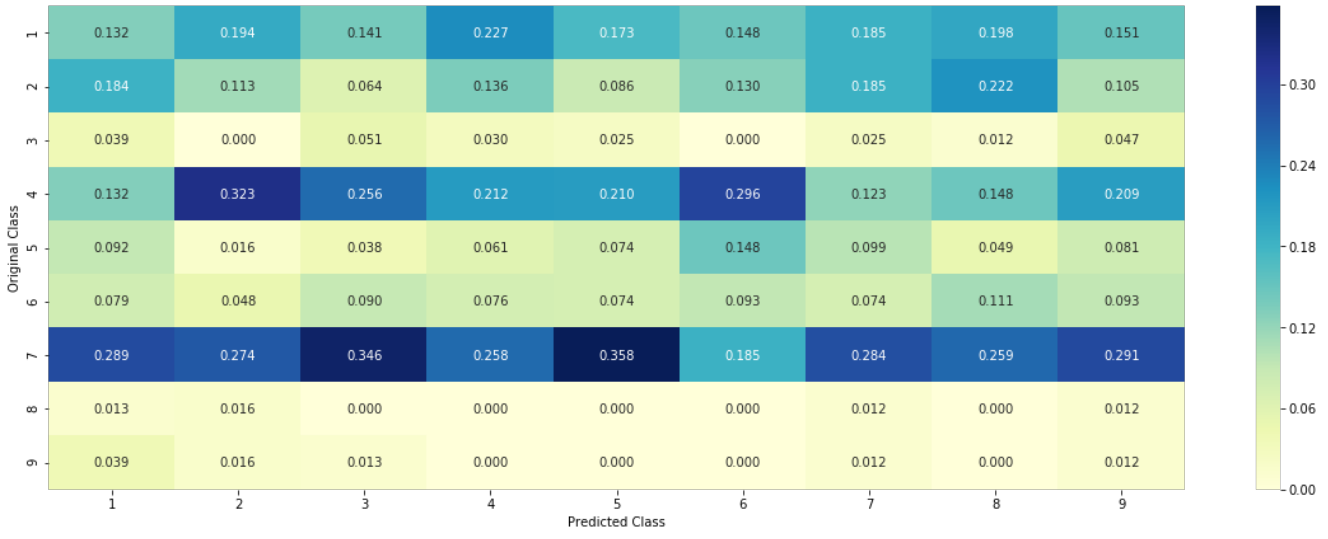
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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_predicted_y, eps=1e-15))

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

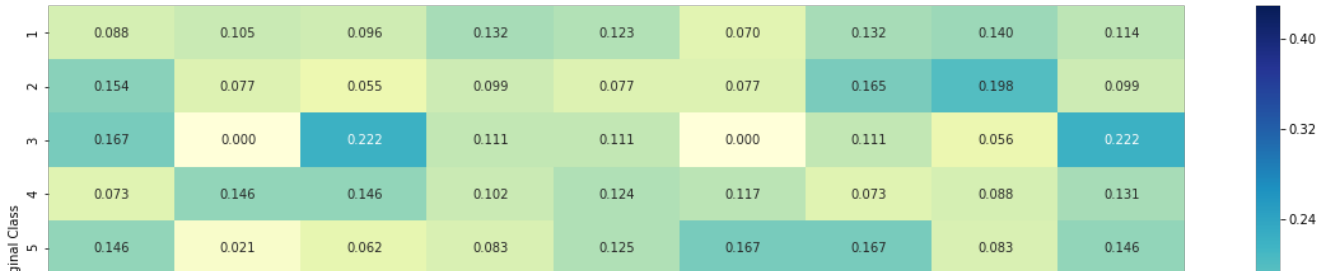
```
Log loss on Cross Validation Data using Random Model 2.4120679759903165
Log loss on Test Data using Random Model 2.5384308680465124
-------------------- Confusion matrix --------------------
```

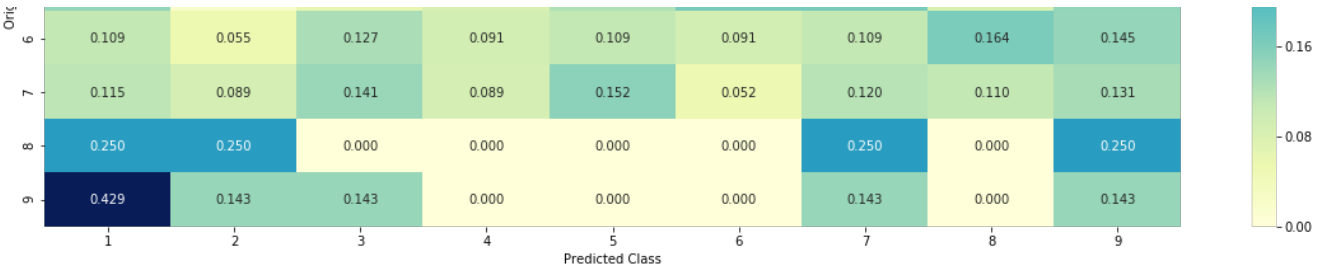



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- 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 feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / nu
mber of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation 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_fea'
# 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 array for each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs 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['Gene']=='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[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured 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 as 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, 0.0378787878787878, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787878787878, 0.0378787878787878,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    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 each feature value in the da
ta
    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_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             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 : 227
BRCA1     171
TP53      104
EGFR       95
PTEN       89
BRCA2      76
KIT        65
BRAF       59
ALK        46
ERBB2      45
PIK3CA     42
Name: Gene, dtype: int64
```

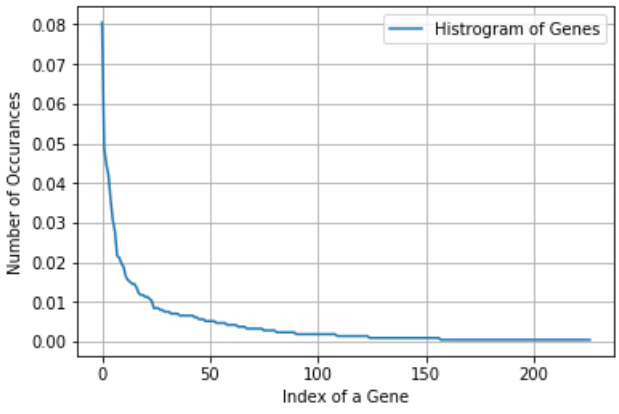
In [17]:

```
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 227 different categories of genes in the train data, and they are distibuted as follows
```

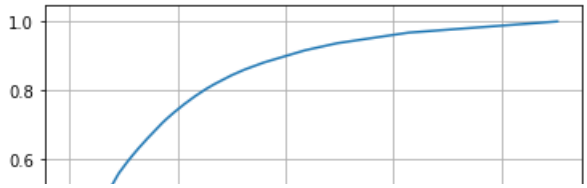
In [18]:

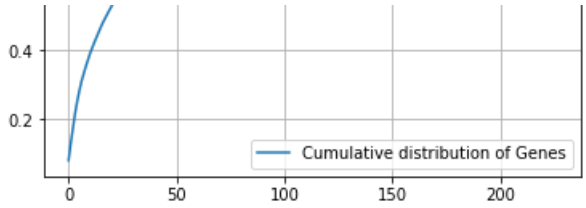
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", 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 respone coding method. The sha
pe of gene feature:", train_gene_feature_responseCoding.shape)
```

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

TFIDF vectorizer on 'gene' feature with unigrams and max_features = 1000

In [23]:

```
#TFIDF vectorizer to featurize 'gene'
gene_vectorizer = TfidfVectorizer(ngram_range=(1,1),max_features = 1000)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [24]:

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

Out[24]:

```
2483     BRCA1
660     CDKN2A
404       TP53
2399       NF1
3232     NTRK2
Name: Gene, dtype: object
```

In [25]:

```
gene_vectorizer.get_feature_names()
```

Out[25]:

```
['abl1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dusp4',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
```

```
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'gata3',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'il7r',
'jak1',
'jak2',
'kdm5a',
'kdm5c',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm4',
'med12',
'mef2b',
'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',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'setd2',
'sf3b1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stag2',
'stat3',
'stk11',
'tert',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'whsc1',
'xpo1',
'xrcc2',
'yap1']
```

In [26]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 227)
```

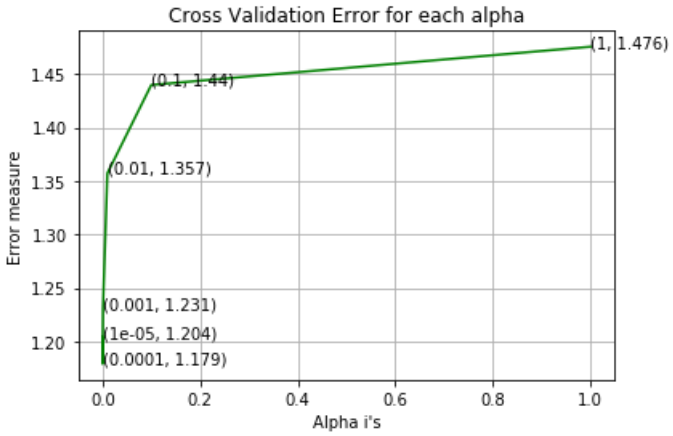
Q4. How good is this gene feature in predicting y_i?

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

In [27]:

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2035269830410114
For values of alpha =  0.0001 The log loss is: 1.1789154940181348
For values of alpha =  0.001 The log loss is: 1.2305830044644832
For values of alpha =  0.01 The log loss is: 1.3572022730697737
For values of alpha =  0.1 The log loss is: 1.4398975967861238
For values of alpha =  1 The log loss is: 1.4756594831256105
```




```
For values of best alpha =  0.0001 The train log loss is: 1.000078152376021
For values of best alpha =  0.0001 The cross validation log loss is: 1.1789154940181348
For values of best alpha =  0.0001 The test log loss is: 1.2136268823363505
```

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

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

In [28]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  227  genes in train dataset?
Ans
1. In test data 635 out of 665 : 95.48872180451127
2. In cross validation data 515 out of  532 : 96.80451127819549
```

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 [29]:

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

```
Number of Unique Variations : 1933
Truncating_Mutations     54
Deletion                 47
```

```
Amplification           45
Fusions                 22
Q61L                     3
G12V                     3
S308A                    2
A146V                    2
Q61H                     2
Q209L                    2
Name: Variation, dtype: int64
```

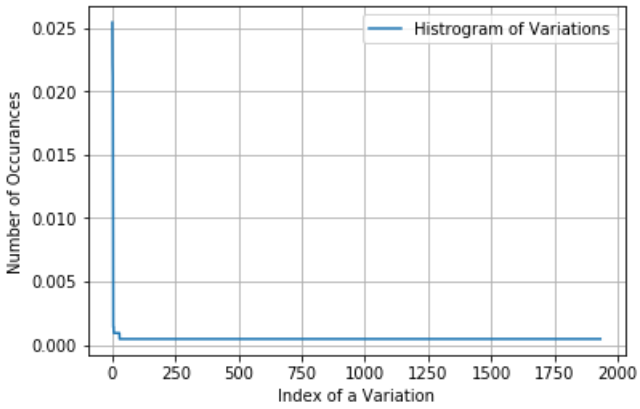
In [30]:

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

```
Ans: There are 1933 different categories of variations in the train data, and they are distibuted
as follows
```

In [31]:

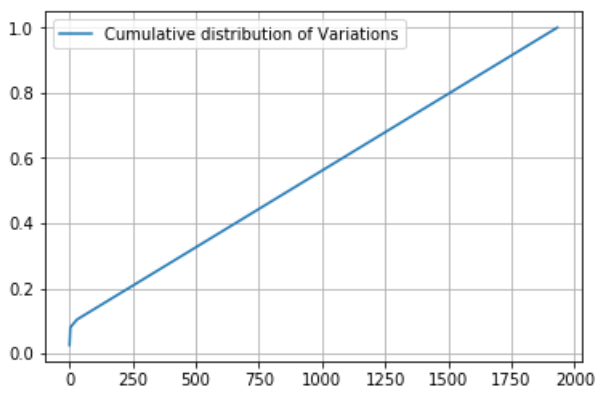
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [32]:

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

```
[0.02542373 0.04755179 0.06873823 ... 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 [33]:

```
# 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, "Variation", cv_df))
```

In [34]:

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

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

TFIDF vectorizer on 'variation' feature with unigrams and max_features = 1000

In [35]:

```
variation_vectorizer = TfidfVectorizer(ngram_range=(1,1),max_features = 1000)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [36]:

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

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

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

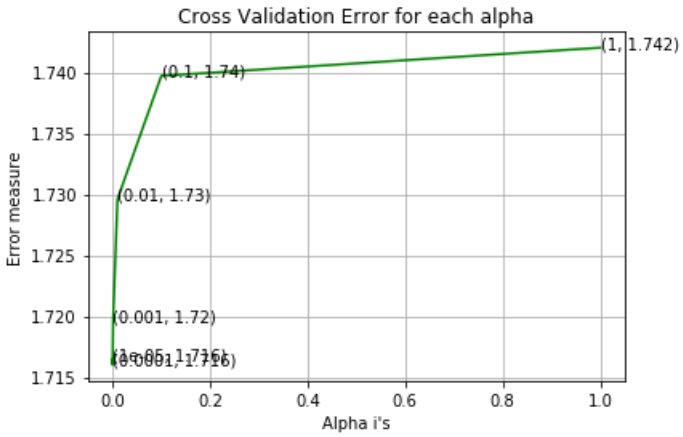
Let's build a model just like the earlier!

In [37]:

```
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_i
ter=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.clas
ses_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7164560528364896
For values of alpha =  0.0001 The log loss is: 1.7160036260654847
For values of alpha =  0.001 The log loss is: 1.7195942623911278
For values of alpha =  0.01 The log loss is: 1.7295914781307313
For values of alpha =  0.1 The log loss is: 1.7397024759914088
For values of alpha =  1 The log loss is: 1.7420105404501762
```




```
For values of best alpha =  0.0001 The train log loss is: 1.2098333112844335
```

```
For values of best alpha =  0.0001 The cross validation log loss is: 1.7160036260654847
For values of best alpha =  0.0001 The test log loss is: 1.6892222367746912
```

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

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

In [38]:

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

```
Q12. How many data points are covered by total  1933  genes in test and cross validation data
sets?
Ans
1. In test data 73 out of 665 : 10.977443609022556
2. In cross validation data 58 out of  532 : 10.902255639097744
```

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 [39]:

```
# 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 [40]:

```
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 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

TFIDF vectorizer on feature 'text' with min_df = 3 and max_features = 1000

In [41]:

```
text_vectorizer = TfidfVectorizer(min_df=3,max_features =1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
print("Total number of unique words in train data :", len(train_text_features))
```

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

In [42]:

```
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 [43]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [44]:

```
# 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_text_feature_responseCoding.
sum(axis=1)).T
```

In [45]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [46]:

```
#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 [47]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({250.84938334481316: 1, 173.58040587685227: 1, 136.131513631916: 1, 127.40736198786946: 1,
125.6227998276431: 1, 115.76926309368311: 1, 115.41067811081385: 1, 113.64915523709207: 1,
106.70389361650162: 1, 106.28213007181618: 1, 105.07031026138013: 1, 90.52959870839136: 1,
88.65383270061382: 1, 86.26121595042265: 1, 85.97550568854363: 1, 83.53712668932876: 1,
80.07449711822305: 1, 77.85076928217704: 1, 77.17428191750331: 1, 74.77533822393333: 1,
73.94732049641246: 1, 72.40888391962064: 1, 68.56246983322865: 1, 68.14851726693247: 1,
65.97202169487954: 1, 65.6630294121224: 1, 65.53065606544048: 1, 63.1164083110664 65: 1,
62.34609875275944: 1, 62.17292268001425: 1, 62.07344177801858: 1, 61.69872742635312: 1,
60.86546856374718: 1, 59.253499255394225: 1, 58.19334963305755: 1, 58.13617848509244: 1,
55.36919445115448: 1, 55.05364080766033: 1, 53.345874709448026: 1, 51.42160796798248: 1,
51.05904408659656: 1, 49.803756993343775: 1, 48.0352842886269: 1, 47.84521620917027: 1,
47.60813184657325: 1, 47.601474304741 72: 1, 46.63956250312826: 1, 45.23674433187654: 1,
44.897505013870855: 1, 44.62508435261888: 1, 43.256132306302625: 1, 42.54849139508399: 1,
42.380739989755455: 1, 42.122104436336734: 1, 42.021491474152846: 1, 41.88477368695597: 1,
41.74155059200702: 1, 41.352896137943: 1, 41.17830423868415: 1, 41.125809194110964: 1,
40.98624572070282: 1, 40.87750560848737: 1, 40.28467187448177: 1, 39.8867082047967: 1,
39.49325249451955: 1, 39.102431412583705: 1, 38.64385726378947: 1, 38.56357920216562: 1,
38.209815521661874: 1, 38.17297871148776: 1, 38.05514728578337: 1, 37.9684547273585: 1,
37.87242945342346: 1, 36.93278388399008: 1, 36.80043069763593: 1, 36.31232875149059: 1,
36.21859925900251: 1, 36.18181317645938: 1, 35.32487182092265: 1, 35.00429982768249: 1,
34.83435995859653: 1, 34.825294135678284: 1, 34.777156996624704: 1, 34.76228084777424: 1,
34.639196550820586: 1, 34.25905808953259: 1, 34.22677226596495: 1, 33.7806262126190 26: 1,
33.14863396704606: 1, 32.92823916998942: 1, 32.85448109170402: 1, 32.815545106876016: 1,
32.655913855647974: 1, 32.51380035589174: 1, 31.906903809697802: 1, 31.7418331237653: 1,
31.731487237109228: 1, 31.703223809150717: 1, 31.48410957548026: 1, 31.1738499434235: 1,
31.103655923751987: 1, 31.031885417778433: 1, 30.885758997939828: 1, 30.8650375008542 46: 1,
30.73365363648758: 1, 30.713110648021345: 1, 30.69736905089719: 1, 30.4538279969494 14: 1,
30.417981672067306: 1, 30.190488407298503: 1, 30.055101962661123: 1, 30.0134641123100 54: 1,
29.862743937386664: 1, 29.757586308836: 1, 29.744095525049314: 1, 29.69051921102227: 1,
29.570871807955815: 1, 29.52433675280889: 1, 29.455781102682593: 1, 29.34177981348722: 1,
29.21458171114347: 1, 29.07761576743431: 1, 29.0043661233152 43: 1, 28.89647578340294: 1,
28.75613135924859: 1, 28.575492916706324: 1, 28.425811068615907: 1, 28.12536077942352: 1,
28.04995480550561: 1, 27.830886706581897: 1, 27.777648604838333: 1, 27.6782557225077 64: 1,
27.55802849489344: 1, 27.352200129428965: 1, 27.19036486802865: 1, 27.1345878635766 48: 1,
26.98815535550537: 1, 26.922994691299102: 1, 26.816057702218743: 1, 26.8119858565612 75: 1,
26.75501740464815: 1, 26.718209928846925: 1, 26.71090714509612: 1, 26.6084023675234 14: 1,
26.40187503579523: 1, 26.329345436213146: 1, 26.12932040993099: 1, 26.00598063322332: 1,
25.95521002966407: 1, 25.817996770785484: 1, 25.733490179961905: 1, 25.542017988836 92: 1,
25.44565895985404: 1, 25.438881992529506: 1, 24.95840471878667: 1, 24.8542938959930 03: 1,
24.706172586209615: 1, 24.66965407199232: 1, 24.60990997299197: 1, 24.5932612527840 46: 1,
24.57668885117809: 1, 24.565637554926994: 1, 24.362009708525093: 1, 24.1626019947023 96: 1,
23.920780144187162: 1, 23.90161858704875: 1, 23.86200086601482: 1, 23.8237450221005 03: 1,
23.73658815859873: 1, 23.720961139115474: 1, 23.69494754519935: 1, 23.6812480877039 14: 1,
23.64981076061296: 1, 23.585258245615233: 1, 23.511756698483875: 1, 23.4489021948578 12: 1,
23.421156542574217: 1, 23.390327357676462: 1, 23.355690555836137: 1, 23.2014811227945 1: 1,
23.19864744374203: 1, 23.16576472375741: 1, 23.081918200635812: 1, 22.8749538394984 06: 1,
22.873467102154724: 1, 22.82524137459261: 1, 22.745274444038188: 1, 22.613251023898 89: 1,
22.47438750436784: 1, 22.423736184826353: 1, 22.320017620419243: 1, 22.2678734647211 84: 1,
22.265020444095445: 1, 22.224600459621872: 1, 22.20921881074432: 1, 22.0792632171357 65: 1,
21.9138128846308: 1, 21.813679344669413: 1, 21.810433367399565: 1, 21.69245537307: 1,
21.652250975244783: 1, 21.639634054314076: 1, 21.599266145662394: 1, 21.5852161116266 43: 1,
21.54343598236743: 1, 21.50144817660116: 1, 21.461367000112993: 1, 21.4348423909893 7: 1,
21.43467147182242: 1, 21.415135403769302: 1, 21.36242425725595: 1, 21.2723441698721 03: 1,
21.264192193627334: 1, 21.245420808709103: 1, 21.219249372099352: 1, 21.2169728629466 8: 1,
21.144075834876325: 1, 21.095250812363965: 1, 21.082782504852883: 1, 21.0470242944691 64: 1,
21.0054480840841: 1, 21.003730437974728: 1, 20.974558245090442: 1, 20.919725975030 09: 1,
20.84993459594471: 1, 20.838706748949292: 1, 20.813881114793347: 1, 20.7966373912451 56: 1,
20.60639887412778: 1, 20.564561594556558: 1, 20.555368622191402: 1, 20.5059034445215 96: 1,
20.472245077814513: 1, 20.4616448002987: 1, 20.344074472428215: 1, 20.2938922161448 47: 1,
20.209127504476147: 1, 20.199211531478827: 1, 20.117875956160344: 1, 20.0765832447720 17: 1,
19.970285530708452: 1, 19.95221216714082: 1, 19.942594582479003: 1, 19.9350806658382 25: 1,
19.87665767090135: 1, 19.864099236165796: 1, 19.830938322928343: 1, 19.7254277304826 18: 1,
19.696603666644545: 1, 19.671301940123264: 1, 19.67085707402411: 1, 19.5897784746848 9: 1,
19.545380856812578: 1, 19.527223437547104: 1, 19.482832881205237: 1, 19.4630503048010 5: 1,
19.43266520916424: 1, 19.38903160640166: 1, 19.34329935456212: 1, 19.331785814992 89: 1,
19.29969059406348: 1, 19.22209296468521: 1, 19.20977305548122: 1, 19.176398845065435: 1,
18.937091659016563: 1, 18.90425724222098: 1, 18.887875905185222: 1, 18.8620038942702 7: 1,
18.851614094682: 1, 18.771201890202015: 1, 18.75316124728579: 1, 18.6997910476799: 1,
```

18.623145315023 73: 1, 18.618694584504 82: 1, 18.603750214859264: 1, 18.600259515405416: 1,
18.575527292753137: 1, 18.501702688252138: 1, 18.43835545037385: 1, 18.417829489928376: 1,
18.41231708443015: 1, 18.3080994946346 86: 1, 18.299219857016016: 1, 18.288198941831002: 1,
18.24415991254655: 1, 18.216283055313 02: 1, 18.19824405115925: 1, 18.19768293801141: 1,
18.123430817391245: 1, 18.07196654026417: 1, 18.069273969630046: 1, 18.05594356989582: 1,
18.028057900800874: 1, 17.999929940132215: 1, 17.97610977437094: 1, 17.96483968494: 1,
17.95795709605899: 1, 17.888327630778274: 1, 17.84732942086 16: 1, 17.819610722010502: 1,
17.815359361088113: 1, 17.776935480760315: 1, 17.685071520403387: 1, 17.63809544048854: 1,
17.626630507401593: 1, 17.614253703293862: 1, 17.560313981870994: 1, 17.525965602765005: 1,
17.413224017656635: 1, 17.391374862240774: 1, 17.362396201953647: 1, 17.358589374707133: 1,
17.3463673648563: 1, 17.337058777191928: 1, 17.331207049211 38: 1, 17.295094172846706: 1,
17.264816717748126: 1, 17.249446409623875: 1, 17.234470257170447: 1, 17.198962717717414: 1,
17.13095304447764: 1, 17.069223254646584: 1, 17.02491926050582: 1, 17.022137430834693: 1,
16.9531971722634: 1, 16.952512624944436: 1, 16.944922988599632: 1, 16.8942232253848: 1,
16.88128806734687: 1, 16.845169424120076: 1, 16.830210994368233: 1, 16.807700727586386: 1,
16.804215550910598: 1, 16.77304764883882: 1, 16.734294603168436: 1, 16.721473029743024: 1,
16.67161724322511: 1, 16.655449566302625: 1, 16.617744253864416: 1, 16.612217430313724: 1,
16.596546531270377: 1, 16.568598560565395: 1, 16.54590131809162: 1, 16.521870265279773: 1,
16.514778912259935: 1, 16.477393648295774: 1, 16.447861601712546: 1, 16.413150176740277: 1,
16.376086380065594: 1, 16.3457790323645: 1, 16.2726283296015 45: 1, 16.2613526579027 48: 1,
16.23240435180572: 1, 16.202259682758484: 1, 16.201234838784593: 1, 16.19135326866363: 1,
16.16941183106635: 1, 16.168221176864684: 1, 16.16216183912841: 1, 16.1573891086037: 1,
16.132527259990677: 1, 16.10017375499155: 1, 16.091048771882996: 1, 15.980753560482356: 1,
15.954863211052558: 1, 15.935924176781526: 1, 15.92138693822821: 1, 15.8907208825705: 1,
15.886962304876615: 1, 15.864300044371529: 1, 15.860252020803184: 1, 15.853319909544624: 1,
15.824510681647714: 1, 15.790243348706424: 1, 15.744739893544832: 1, 15.74035736134 08: 1,
15.72801421770738: 1, 15.678919080410013: 1, 15.663703173935284: 1, 15.656003449850395: 1,
15.64354279098606: 1, 15.570304236358599: 1, 15.454873920645795: 1, 15.448625198912627: 1,
15.420539011780043: 1, 15.409660385710062: 1, 15.408265753512852: 1, 15.393788919508811: 1,
15.367224000316371: 1, 15.358402268802 85: 1, 15.316303910543 63: 1, 15.306181746373708: 1,
15.148213532552134: 1, 15.145264300739 81: 1, 15.133472042951535: 1, 15.130750762535309: 1,
15.081065833322425: 1, 15.072814057675581: 1, 15.070519717105682: 1, 15.006894707586506: 1,
15.000890870056084: 1, 14.980486856809122: 1, 14.963928090215745: 1, 14.955797069248268: 1,
14.91735362585259: 1, 14.896348669797 92: 1, 14.895868860574407: 1, 14.89065363207244: 1,
14.889772673704137: 1, 14.870073705469297: 1, 14.852765712775707: 1, 14.838901048862944: 1,
14.771573932277338: 1, 14.760816224779285: 1, 14.747331731868687: 1, 14.728631978451919: 1,
14.722585397821979: 1, 14.717494027460178: 1, 14.701136687743048: 1, 14.680730483218104: 1,
14.659263482086935: 1, 14.649842715944008: 1, 14.621520804856608: 1, 14.593714603769326: 1,
14.589428224848977: 1, 14.556698032661542: 1, 14.549428661376 06: 1, 14.538314333791476: 1,
14.534799566274254: 1, 14.494415650179 67: 1, 14.491194677178957: 1, 14.485114621165: 1,
14.466917886983458: 1, 14.457269170289454: 1, 14.439453788931 92: 1, 14.407370579202224: 1,
14.383533941485071: 1, 14.364629674140 38: 1, 14.362683479217196: 1, 14.359632113750248: 1,
14.354591675740526: 1, 14.299495303258924: 1, 14.293636028663848: 1, 14.290515530079 54: 1,
14.282572615382925: 1, 14.281499008231712: 1, 14.256811062735 41: 1, 14.241698145221829: 1,
14.214777206779633: 1, 14.196584718992755: 1, 14.156874527575747: 1, 14.144127761635 18: 1,
14.143380980514443: 1, 14.098888618688 19: 1, 14.084144315272216: 1, 14.081686368948473: 1,
14.076067129284699: 1, 14.057131381363 95: 1, 14.017170084450136: 1, 14.009035121975593: 1,
13.982607121021658: 1, 13.976250342297808: 1, 13.931402091775778: 1, 13.925055252070054: 1,
13.917400668577478: 1, 13.863702029027403: 1, 13.862627612279061: 1, 13.862472842630874: 1,
13.837459627775539: 1, 13.786721763678909: 1, 13.767291999339184: 1, 13.735599128815776: 1,
13.733628626627588: 1, 13.714944643767938: 1, 13.705390502886242: 1, 13.702893036194766: 1,
13.676859048538505: 1, 13.621963861240 25: 1, 13.559959410369 83: 1, 13.559122932425076: 1,
13.489509091807568: 1, 13.419691594817746: 1, 13.406726467766019: 1, 13.402655204211477: 1,
13.385551826685571: 1, 13.369954318276735: 1, 13.345898483151622: 1, 13.323899434442222: 1,
13.321489812529537: 1, 13.274828846166306: 1, 13.228141266715 11: 1, 13.227127729694645: 1,
13.17437551196785: 1, 13.150174362186801: 1, 13.137402861270756: 1, 13.128686031214592: 1,
13.103520303470185: 1, 13.014480965539422: 1, 12.982320201698748: 1, 12.952931860042497: 1,
12.94658460021788: 1, 12.942647800014777: 1, 12.933058654191669: 1, 12.921660467041214: 1,
12.900868636880142: 1, 12.898776277030093: 1, 12.889289407047654: 1, 12.873670204514879: 1,
12.8435817506172: 1, 12.833348095707397: 1, 12.822345767832608: 1, 12.803595842137 49: 1,
12.797916514692925: 1, 12.764821322118992: 1, 12.749969963162792: 1, 12.739928540891475: 1,
12.723383875122996: 1, 12.709450235213 59: 1, 12.691891545800779: 1, 12.651672680303921: 1,
12.597704713366117: 1, 12.569061365058806: 1, 12.556354313650482: 1, 12.536538018997994: 1,
12.526448846718909: 1, 12.488586407180 2: 1, 12.484868796917638: 1, 12.450573798571 84: 1,
12.431927082219504: 1, 12.412769214126062: 1, 12.411111860256556: 1, 12.384179186576116: 1,
12.374811453048407: 1, 12.349211998057479: 1, 12.346192419603558: 1, 12.334993054613912: 1,
12.275748315454303: 1, 12.266895185885751: 1, 12.231697676450091: 1, 12.229541285157735: 1,
12.225856303364635: 1, 12.224257563700407: 1, 12.218189984629525: 1, 12.173115683126642: 1,
12.140923433600106: 1, 12.112234172700 86: 1, 12.105286711864682: 1, 12.094541136579162: 1,
12.093257495790516: 1, 12.092256470938226: 1, 12.082021499977083: 1, 12.072803073896925: 1,
12.055933869233556: 1, 12.045565880550637: 1, 12.039429678191466: 1, 12.034089835555047: 1,
12.031795458652631: 1, 11.999380480711906: 1, 11.948085369453988: 1, 11.941978853530 97: 1,
11.932150967690298: 1, 11.887775552351433: 1, 11.865117608546242: 1, 11.845659899894146: 1,
11.845214430753204: 1, 11.827768324608128: 1, 11.795575115649859: 1, 11.758979159593162: 1,
11.755272029895284: 1, 11.751742075763058: 1, 11.743311077652724: 1, 11.733954303781788: 1,
11.731430116591776: 1, 11.690978141534794: 1, 11.683387550302657: 1, 11.680734242291 65: 1,
11.657193856175732: 1, 11.637269029519 07: 1, 11.621686836672431: 1, 11.614001824452414: 1,

11.608409785035523: 1, 11.59804064762826: 1, 11.58752405989702: 1, 11.565008013456014: 1,
11.561371766884234: 1, 11.535787213398837: 1, 11.511068613150462: 1, 11.485309035016284: 1,
11.480099575059963: 1, 11.476536768557473: 1, 11.435226911261179: 1, 11.417422222668618: 1,
11.412702923830917: 1, 11.395766763318827: 1, 11.387769819970934: 1, 11.35946216054538: 1,
11.329235526428477: 1, 11.325557343856174: 1, 11.32329016844757: 1, 11.29511246622844: 1,
11.28789447830981: 1, 11.26769728748942: 1, 11.252644409142713: 1, 11.228111116493848: 1,
11.150962433037819: 1, 11.135914898022836: 1, 11.134806677422628: 1, 11.105852319623981: 1,
11.10165046560859: 1, 11.100462360161353: 1, 11.097053569288748: 1, 11.063491010690909: 1,
11.061801538461692: 1, 11.054648706717673: 1, 11.045558793446126: 1, 11.045509230997121: 1,
11.045151808784546: 1, 11.043150577438661: 1, 11.037875375721988: 1, 11.033497111351991: 1,
11.01114451826748: 1, 10.978239398241042: 1, 10.964406138811087: 1, 10.940212564034075: 1,
10.904343926609604: 1, 10.88723478589254: 1, 10.881529057579652: 1, 10.87844432434605: 1,
10.870127518807358: 1, 10.855580784038821: 1, 10.84965067454242: 1, 10.84833063939414: 1,
10.839485758995767: 1, 10.817452695916295: 1, 10.786119219437984: 1, 10.75569327714852: 1,
10.752941537025281: 1, 10.702268140139864: 1, 10.669724137477232: 1, 10.669666084935681: 1,
10.661531578835561: 1, 10.66055145737383: 1, 10.654087084621056: 1, 10.650533195274281: 1,
10.62065653554598: 1, 10.612925265098767: 1, 10.610065512343379: 1, 10.604712284736207: 1,
10.588403011383777: 1, 10.53522490131269: 1, 10.526889705880576: 1, 10.522702734274509: 1,
10.517599012578291: 1, 10.504412492993055: 1, 10.503028320039128: 1, 10.500220986174137: 1,
10.497663380034156: 1, 10.497048099039873: 1, 10.497030123210028: 1, 10.494366245430259: 1,
10.490117599722824: 1, 10.487097024810604: 1, 10.486083724713916: 1, 10.4640387878866l3: 1,
10.458471684351986: 1, 10.447614264131351: 1, 10.424871858267775: 1, 10.414543142771183: 1,
10.399743037734696: 1, 10.3852965851453: 1, 10.383392916481919: 1, 10.374832616743625: 1,
10.368239827180922: 1, 10.364264924566445: 1, 10.324908539021498: 1, 10.324199280519082: 1,
10.311014890379827: 1, 10.278156613998663: 1, 10.27479528925142: 1, 10.259045025364482: 1,
10.256087583085293: 1, 10.251933665172244: 1, 10.24009244988318: 1, 10.225260680218778: 1,
10.223954695985977: 1, 10.22369816097516: 1, 10.21920158982964: 1, 10.20888966659296: 1, 10.198923914780439: 1, 10.185225688988004: 1, 10.173169694365981: 1, 10.169486250349799: 1,
10.163285481241932: 1, 10.162642076223309: 1, 10.155270596947252: 1, 10.150643243984332: 1,
10.112879932665034: 1, 10.103214927860702: 1, 10.102197776253451: 1, 10.095870815890802: 1,
10.095322029528216: 1, 10.074473006588912: 1, 10.073330392015132: 1, 10.038027397555917: 1,
10.025201109776331: 1, 10.017320112303482: 1, 10.013336060047738: 1, 10.002984036010673: 1,
10.001265300991413: 1, 9.997144887873175: 1, 9.98592377312657: 1, 9.981576502616274: 1,
9.97729370070248: 1, 9.95468291186727: 1, 9.950691028696859: 1, 9.944630053328561: 1,
9.935018664539353: 1, 9.904877600974723: 1, 9.90326504109297: 1, 9.900357184155169: 1,
9.885000314022053: 1, 9.874551920836147: 1, 9.838221874680297: 1, 9.833181866295702: 1,
9.828166579132947: 1, 9.817675242081366: 1, 9.813908677866955: 1, 9.787024675459339: 1,
9.736727331656127: 1, 9.735379520314885: 1, 9.71933940852906: 1, 9.714584445021314: 1,
9.699445921631929: 1, 9.659708313223924: 1, 9.644050281304892: 1, 9.628699589672303: 1,
9.627072546313768: 1, 9.617275695533586: 1, 9.612219934599455: 1, 9.6120618978194: 1,
9.602270244047334: 1, 9.595943318926794: 1, 9.591736768259963: 1, 9.58238386913456: 1,
9.574533399666786: 1, 9.569878738546663: 1, 9.534523288051561: 1, 9.52832127864454: 1,
9.519842597314O9: 1, 9.499194990603819: 1, 9.487215499561245: 1, 9.483490562380316: 1,
9.480070609151506: 1, 9.442166267020484: 1, 9.438467736109454: 1, 9.430600343529344: 1,
9.424939602651419: 1, 9.42269307061035: 1, 9.421546663814675: 1, 9.415178526383126: 1,
9.377263306189878: 1, 9.368678922369469: 1, 9.366039368287998: 1, 9.358421482587143: 1,
9.353602230592022: 1, 9.332301034414114: 1, 9.313099607562055: 1, 9.304906123237341: 1,
9.30455605598711: 1, 9.304209091585562: 1, 9.299196955247632: 1, 9.290030487142444: 1,
9.28444989390869: 1, 9.278373830215477: 1, 9.267239339599806: 1, 9.259181167640792: 1,
9.238272411939478: 1, 9.231361991043071: 1, 9.228443797441907: 1, 9.176374927923066: 1,
9.167896241038623: 1, 9.15847340757363: 1, 9.157859084309807: 1, 9.153876357499467: 1,
9.14688297148217: 1, 9.137980374891187: 1, 9.137772578126635: 1, 9.13151386545671: 1,
9.130211971284934: 1, 9.11131620365886: 1, 9.105209405881622: 1, 9.098495801650024: 1,
9.093343492729113: 1, 9.089613643222211: 1, 9.059833991365085: 1, 9.05653609995097: 1,
9.049357123414238: 1, 9.029522729992335: 1, 9.025800164716246: 1, 9.024031619250799: 1,
9.016803496997527: 1, 9.009880117652417: 1, 9.008640292496535: 1, 9.002446109933205: 1,
8.996568694485B9: 1, 8.972779379156671: 1, 8.965577945009267: 1, 8.962148423308633: 1,
8.9530608265758: 1, 8.9500339649937: 1, 8.942730296022111: 1, 8.933757660437694: 1,
8.924210254296273: 1, 8.913440514048506: 1, 8.912613540196523: 1, 8.905224284420603: 1,
8.900145317650416: 1, 8.897078450564342: 1, 8.86971748825695: 1, 8.85258080801555: 1,
8.850098361634512: 1, 8.844835726018346: 1, 8.831748454587599: 1, 8.815063916280412: 1,
8.814625492392157: 1, 8.81097052747637: 1, 8.807834807313974: 1, 8.804264004665905: 1,
8.791492151235738: 1, 8.77831495483206: 1, 8.772128766580977: 1, 8.771177569881502: 1,
8.719606384750549: 1, 8.717848120619625: 1, 8.698208929532758: 1, 8.682989482180005: 1,
8.678533701496649: 1, 8.663318280683592: 1, 8.662763657707908: 1, 8.649633927943913: 1,
8.645529835011528: 1, 8.644397102524707: 1, 8.642092182094053: 1, 8.641054598487898: 1,
8.64103502888645: 1, 8.625616698133397: 1, 8.620269330947322: 1, 8.607649962783098: 1,
8.585710611167197: 1, 8.578336235091925: 1, 8.510348180000234: 1, 8.493342738175937: 1,
8.480545954250818: 1, 8.464264993747138: 1, 8.45790199850654: 1, 8.448386074037158: 1,
8.437446596586112: 1, 8.433598814119767: 1, 8.431753308987506: 1, 8.426403331154267: 1,
8.418018052966325: 1, 8.416762753186196: 1, 8.412794850063833: 1, 8.39986253279029: 1,
8.389187592404332: 1, 8.387425323000109: 1, 8.377349281143552: 1, 8.371779314803668: 1,
8.3616449931462: 1, 8.361181018688413: 1, 8.351113608039743: 1, 8.346734147295539: 1, 8.32108366708
39: 1, 8.291750243592517: 1, 8.283413279744211: 1, 8.240076526134972: 1, 8.237181104497786: 1, 8.21
1044370369837: 1, 8.193280717438768: 1, 8.173233539945205: 1, 8.172919899341567: 1, 8.172704950453C
37: 1, 8.143311475736137: 1, 8.137504012938718: 1, 8.136809006960501: 1, 8.125968086116059: 1, 8.12
253156188227: 1, 8.107115765799806: 1, 8.10503973836053: 1, 8.093521065552297: 1,

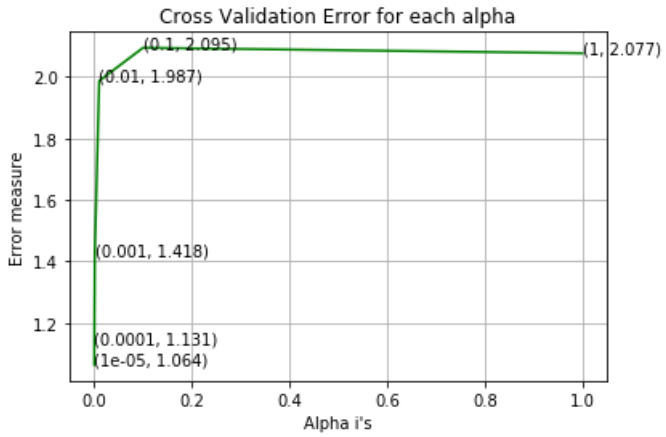
```
8.083895573290508: 1, 8.076770039584892: 1, 8.054766894164558: 1, 8.049694480795697: 1,
8.04324121619698: 1, 7.990257390103659: 1, 7.988003755060118: 1, 7.987060864297764: 1,
7.971592570630994: 1, 7.953764744519241: 1, 7.9503780774662145: 1, 7.930985538400622: 1,
7.920335995828895: 1, 7.915279524238919: 1, 7.911764497764063: 1, 7.909427563720583: 1,
7.898672088496906: 1, 7.89702392598531: 1, 7.879726091724855: 1, 7.877634033440857: 1,
7.870348034271209: 1, 7.870238234278059: 1, 7.867975606083359: 1, 7.862708438997216: 1,
7.844614721340148: 1, 7.844184522451807: 1, 7.817248795173017: 1, 7.814345110156776: 1,
7.802682498913286: 1, 7.7883379570506825: 1, 7.772815755327706: 1, 7.755187662482314: 1,
7.734426556671795: 1, 7.72801718548151: 1, 7.724850190730388: 1, 7.71756508716309: 1,
7.708399241808798: 1, 7.69210279350716: 1, 7.689159523555542: 1, 7.6883132284914595: 1,
7.68470244028102: 1, 7.683780059321191: 1, 7.661745719926168: 1, 7.658054760668431: 1,
7.651811958733248: 1, 7.647806609443449: 1, 7.603127549745608: 1, 7.585378538339278: 1,
7.568330886468624: 1, 7.54812195850637: 1, 7.534843316112778: 1, 7.5325181493818: 1,
7.520387384485567: 1, 7.4874878131381095: 1, 7.461893550585839: 1, 7.461564011972188: 1,
7.456882371372938: 1, 7.447636746384016: 1, 7.4381888166594345: 1, 7.436153136797562: 1,
7.416457201122595: 1, 7.404741613279529: 1, 7.381534932619683: 1, 7.357794129497821: 1,
7.318503903825394: 1, 7.312964161577822: 1, 7.292845853774604: 1, 7.266090372979888: 1,
7.2615218844290395: 1, 7.260160985024689: 1, 7.253755907386711: 1, 7.235799975815874: 1, 7.22648668
00097165: 1, 7.219916733412399: 1, 7.163604091163134: 1, 7.160577662965253: 1, 7.156010126894617:
1, 7.146930809204754: 1, 7.107193339614436: 1, 7.101020225245528: 1, 7.07381864993295: 1,
7.064306725169596: 1, 7.0451666470868926: 1, 7.04302942227507: 1, 7.0230074774155105: 1,
7.0124194630100805: 1, 7.007450402002881: 1, 6.958521402107139: 1, 6.936963263200595: 1, 6.92959809
89354975: 1, 6.891998494615101: 1, 6.885046958784866: 1, 6.8652000598718175: 1, 6.86490432143917:
1, 6.806719522342731: 1, 6.795730699505795: 1, 6.776132282631798: 1, 6.771708542646381: 1,
6.749665250980247: 1, 6.743178414235356: 1, 6.742664488270912: 1, 6.688169245187635: 1,
6.624390909791638: 1, 6.622223215120135: 1, 6.6096624320485935: 1, 6.582792742230055: 1,
6.516624449518357: 1, 6.2393128787720835: 1, 6.214439375389161: 1, 6.153415085581341: 1})
```

In [48]:

```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
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_i
ter=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_text_feature_onehotCoding, y_train)

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

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


best_alpha = np.argmin(cv_log_error_array)
```

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

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

```
For values of alpha =  1e-05 The log loss is: 1.064048914855641
For values of alpha =  0.0001 The log loss is: 1.1311085102977303
For values of alpha =  0.001 The log loss is: 1.418416953930914
For values of alpha =  0.01 The log loss is: 1.9871175540285961
For values of alpha =  0.1 The log loss is: 2.0948493785367517
For values of alpha =  1 The log loss is: 2.0770258585519263
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7262146003807116
For values of best alpha =  1e-05 The cross validation log loss is: 1.064048914855641
For values of best alpha =  1e-05 The test log loss is: 1.1660305919239582
```

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

Ans. Yes, it seems like!

In [49]:

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

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

In [50]:

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

```
3.464 % of word of test data appeared in train data
3.982 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [51]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to 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 [52]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [53]:

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

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

In [54]:

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

    gene_vec = gene_tfidf_vec.fit(train_df['Gene'])
    var_vec  = var_tfidf_vec.fit(train_df['Variation'])
    text_vec = text_tfidf_vec.fit(train_df['TEXT'])

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

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

Stacking the three types of features

In [55]:

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

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

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

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

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


train_gene_var_responseCoding =
np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding =
np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
```

```
cv_gene_var_responseCoding =
np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,
train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding)
)
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
```

In [56]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2227)
(number of data points * number of features) in test data =  (665, 2227)
(number of data points * number of features) in cross validation data = (532, 2227)
```

In [57]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shap
e)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

4.1.1. Naive Bayes

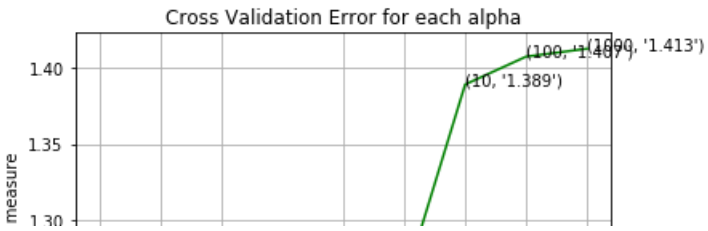
4.1.1.1. Hyper parameter tuning

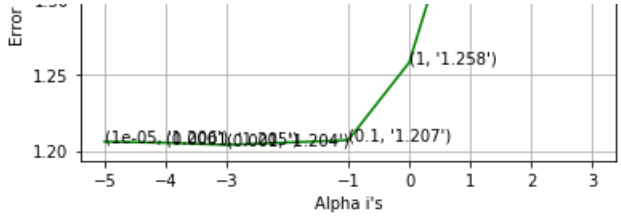
In [58]:

```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=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 vector 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.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.
```

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


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


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

```
for alpha = 1e-05
Log Loss : 1.2060637532581662
for alpha = 0.0001
Log Loss : 1.205310783217136
for alpha = 0.001
Log Loss : 1.2039357117123384
for alpha = 0.1
Log Loss : 1.2070208584998983
for alpha = 1
Log Loss : 1.2581604514292093
for alpha = 10
Log Loss : 1.389105389879291
for alpha = 100
Log Loss : 1.4074360991407115
for alpha = 1000
Log Loss : 1.4126237800779777
```





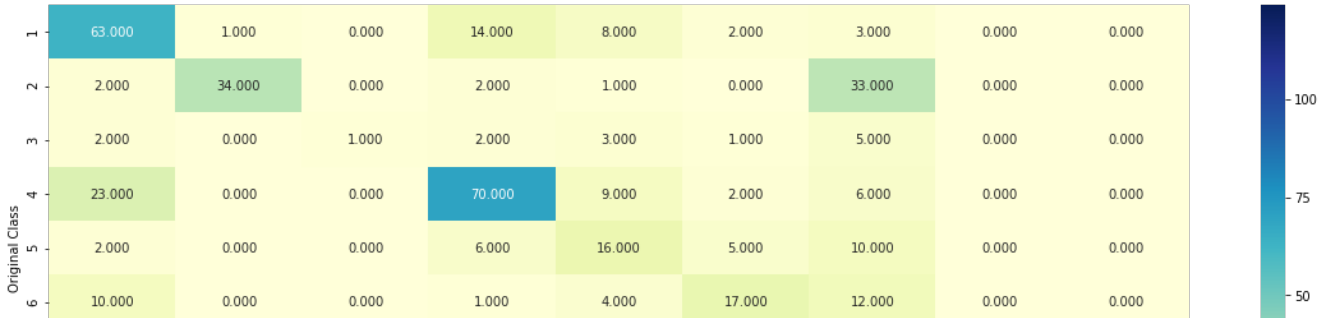

```
For values of best alpha =  0.001 The train log loss is: 0.7442911778617243
For values of best alpha =  0.001 The cross validation log loss is: 1.2039357117123384
For values of best alpha =  0.001 The test log loss is: 1.164097772591265
```

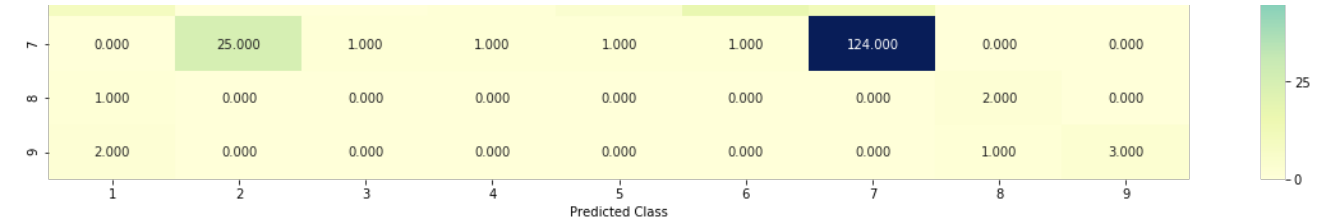
4.1.1.2. Testing the model with best hyper paramters

In [59]:

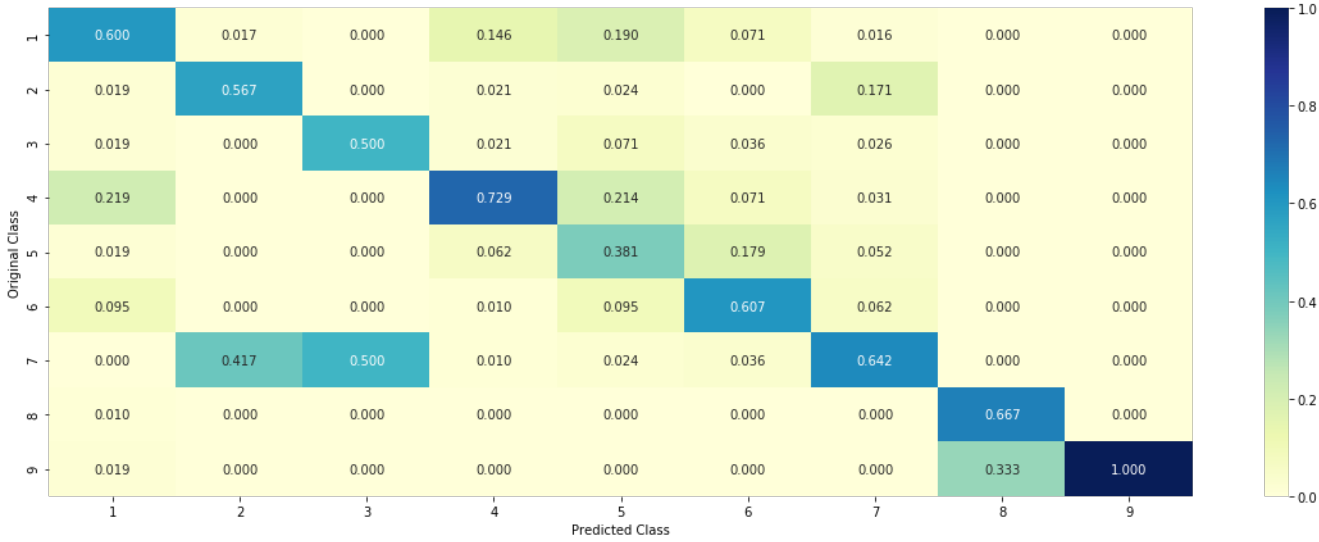
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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 vector 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.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
# ----------------------------

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

```
Log Loss : 1.2039357117123384
Number of missclassified point : 0.37969924812030076
-------------------- Confusion matrix --------------------
```

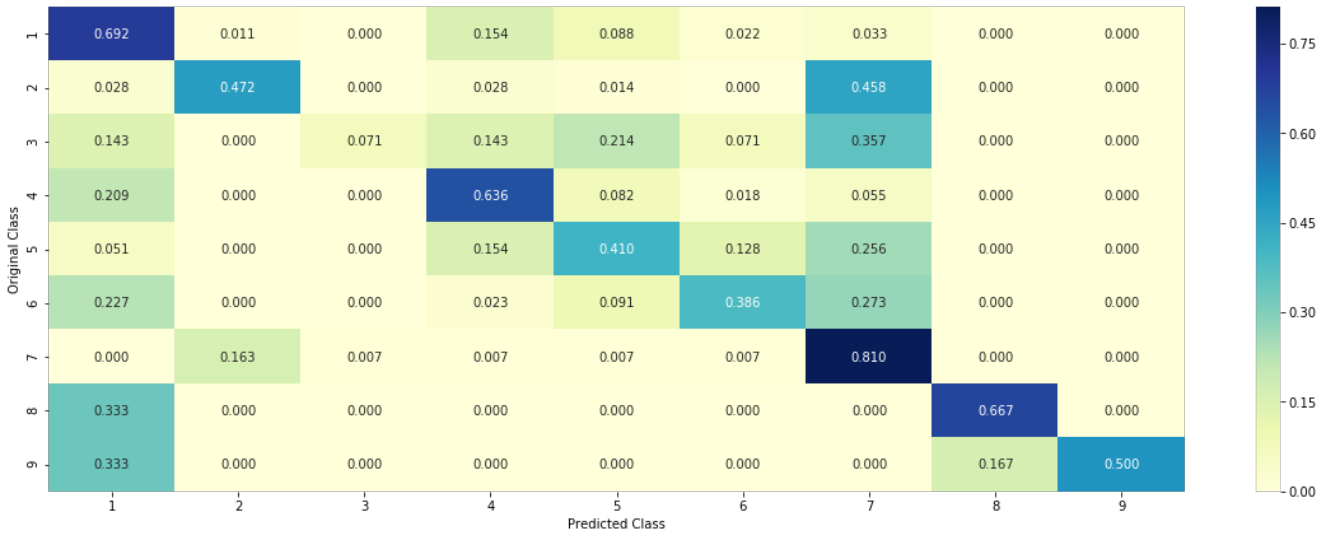




---------- Precision matrix (Columm Sum=1) ----------



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



4.1.1.3. Feature Importance, Correctly classified point

In [60]:

```
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 : 1
Predicted Class Probabilities: [[0.4918 0.0522 0.0135 0.2814 0.0351 0.0365 0.0827 0.0036 0.0033]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

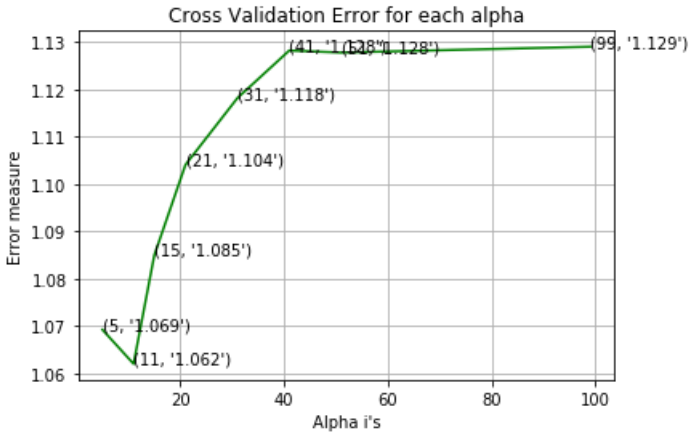
4.2.1. Hyper parameter tuning

In [62]:

```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', 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-ne
ighbors-geometric-intuition-with-a-toy-example-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 = [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 = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
for alpha = 5
Log Loss : 1.0691802054827102
for alpha = 11
Log Loss : 1.0619599051658761
for alpha = 15
Log Loss : 1.0849523812694126
for alpha = 21
Log Loss : 1.1040411018241938
for alpha = 31
Log Loss : 1.1181264579366281
for alpha = 41
Log Loss : 1.1281854594989502
for alpha = 51
Log Loss : 1.1277545508979714
for alpha = 99
Log Loss : 1.1289888469001617
```



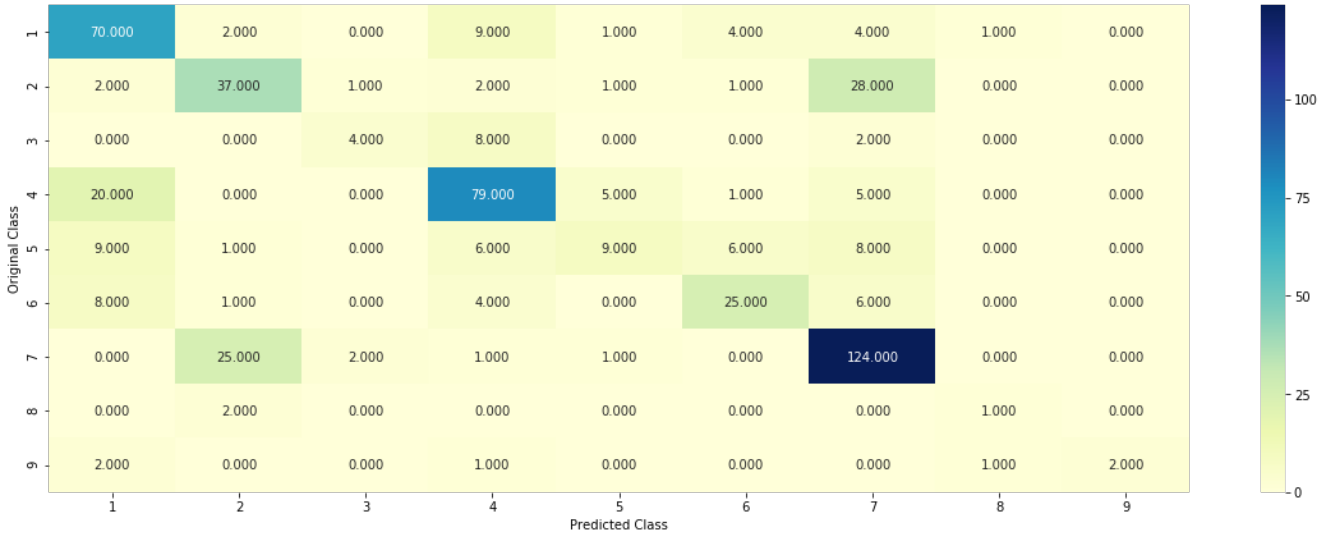

```
For values of best alpha =  11 The train log loss is: 0.6189378371216535
For values of best alpha =  11 The cross validation log loss is: 1.0619599051658761
For values of best alpha =  11 The test log loss is: 1.0963238661163988
```

4.2.2. Testing the model with best hyper paramters

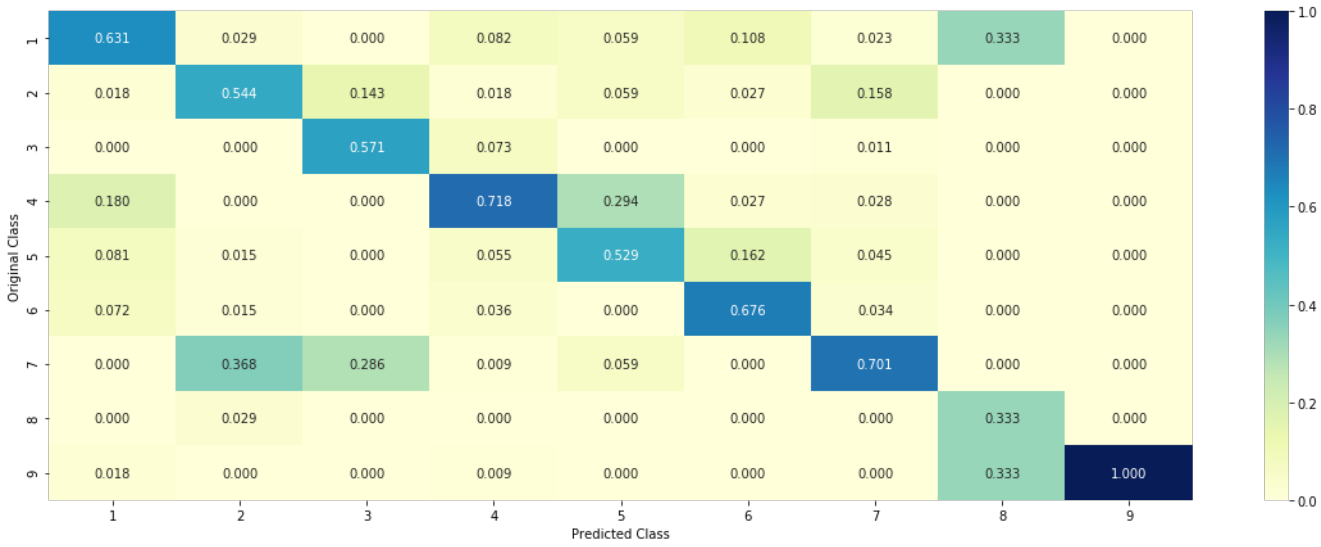
In [63]:

```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', 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-ne
ighbors-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)
```

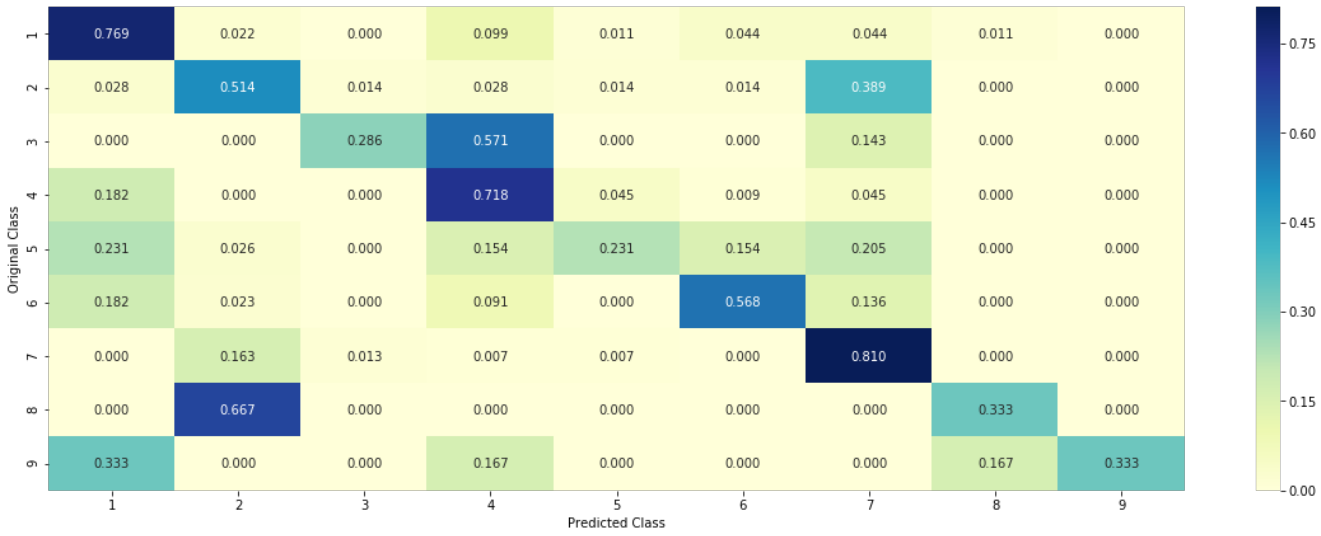
```
Log loss : 1.0619599051658761
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [64]:

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

test_point_index = 120
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha
])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y
[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 1
The  11  nearest neighbours of the test points belongs to classes [1 1 6 6 7 7 7 7 2 7 7]
Fequency of nearest points : Counter({7: 6, 1: 2, 6: 2, 2: 1})
```

4.2.4. Sample Query Point-2

In [65]:

```
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 = 128

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

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [4 4 4
4 4 5 4 5 4 4 2]
Fequency of nearest points : Counter({4: 8, 5: 2, 2: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

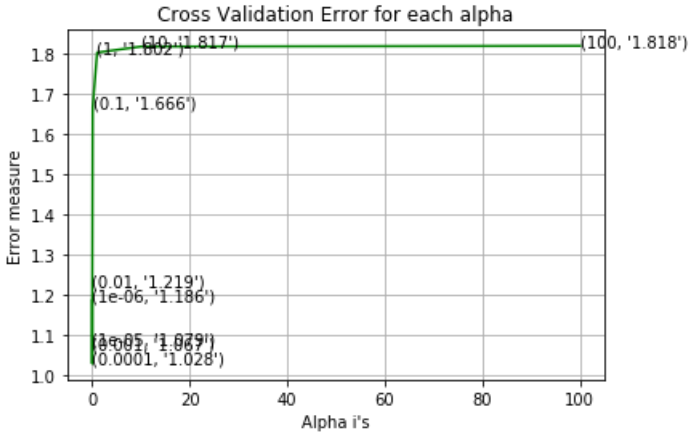
In [66]:

```
# 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_i
ter=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, 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], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1858435941125398
for alpha = 1e-05
Log Loss : 1.0785571049858604
for alpha = 0.0001
Log Loss : 1.0281654246584373
for alpha = 0.001
Log Loss : 1.066646458840037
for alpha = 0.01
```

```
Log Loss : 1.2193764079066391
for alpha = 0.1
Log Loss : 1.6656642117761804
for alpha = 1
Log Loss : 1.801676668403323
for alpha = 10
Log Loss : 1.8165657833847488
for alpha = 100
Log Loss : 1.8183294585309437
```



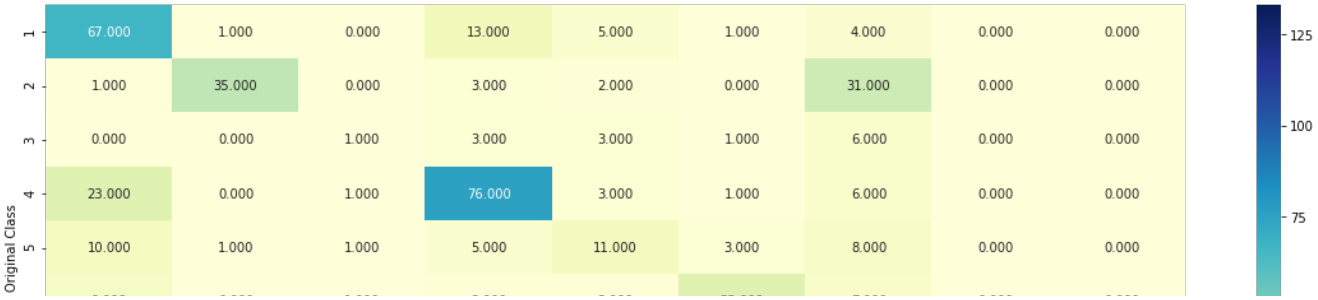

```
For values of best alpha =  0.0001 The train log loss is: 0.5660598805982662
For values of best alpha =  0.0001 The cross validation log loss is: 1.0281654246584373
For values of best alpha =  0.0001 The test log loss is: 1.0058483821264186
```

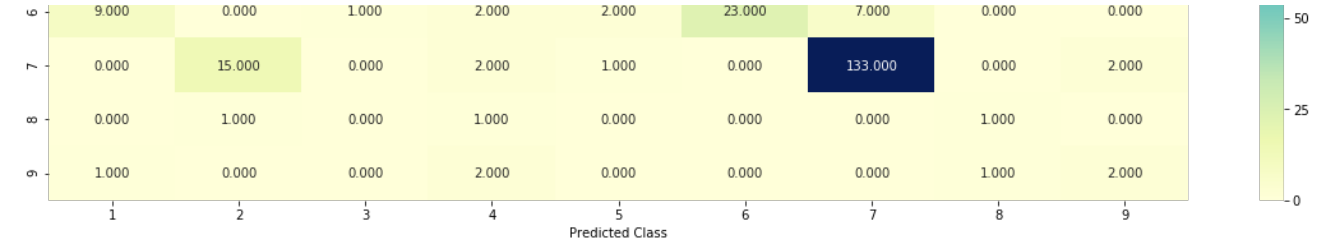
4.3.1.2. Testing the model with best hyper paramters

In [67]:

```
# 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)
```

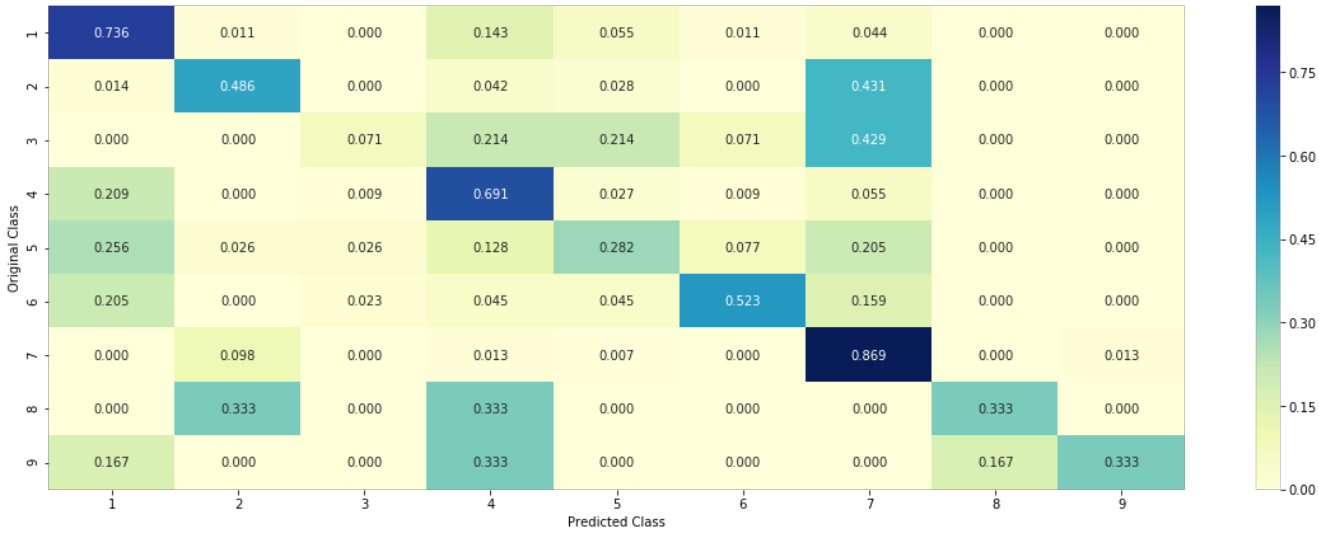
```
Log loss : 1.0281654246584373
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```





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

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



4.3.1.3. Feature Importance

In [68]:

```
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:
```

```
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

Important features from get_impfeature_names_tfidf()

In [69]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0153 0.0051 0.1344 0.8228 0.0103 0.0032 0.005  0.0027 0.0012]]
Actual Class : 4
--------------------------------------------------
64 Text feature [131] present in test data point [True]
103 Text feature [12er] present in test data point [True]
215 Text feature [101] present in test data point [True]
349 Text feature [10] present in test data point [True]
371 Text feature [121] present in test data point [True]
637 Text feature [130] present in test data point [True]
652 Text feature [123] present in test data point [True]
Out of the top  1000  features  7 are present in query point
```

4.3.2. Without Class balancing

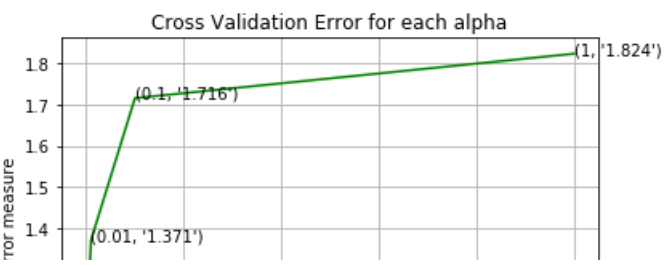
4.3.2.1. Hyper paramter tuning

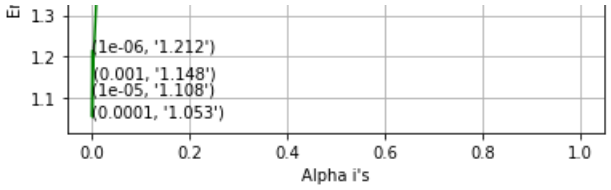
In [70]:

```
# 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_i
ter=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-in
tuition-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 validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.2120296931313257
for alpha = 1e-05
Log Loss : 1.1082560794611809
for alpha = 0.0001
Log Loss : 1.0527096898551893
for alpha = 0.001
Log Loss : 1.1479405692591806
for alpha = 0.01
Log Loss : 1.3711561082959516
for alpha = 0.1
Log Loss : 1.7158025627216136
for alpha = 1
Log Loss : 1.8237796649011144
```






```
For values of best alpha =  0.0001 The train log loss is: 0.5597084985635139
For values of best alpha =  0.0001 The cross validation log loss is: 1.0527096898551893
For values of best alpha =  0.0001 The test log loss is: 1.0188555237072658
```

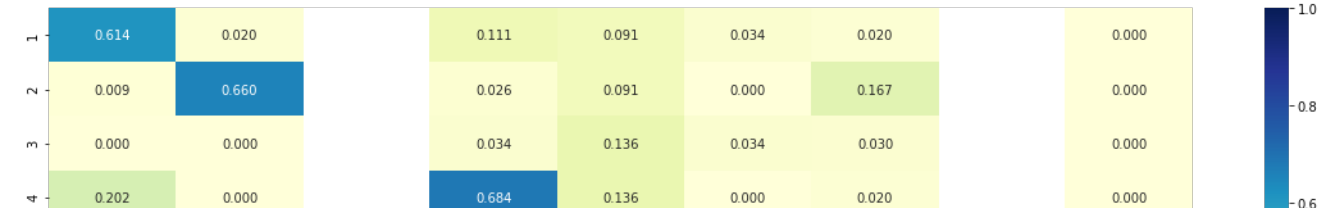
4.3.2.2. Testing model with best hyper parameters

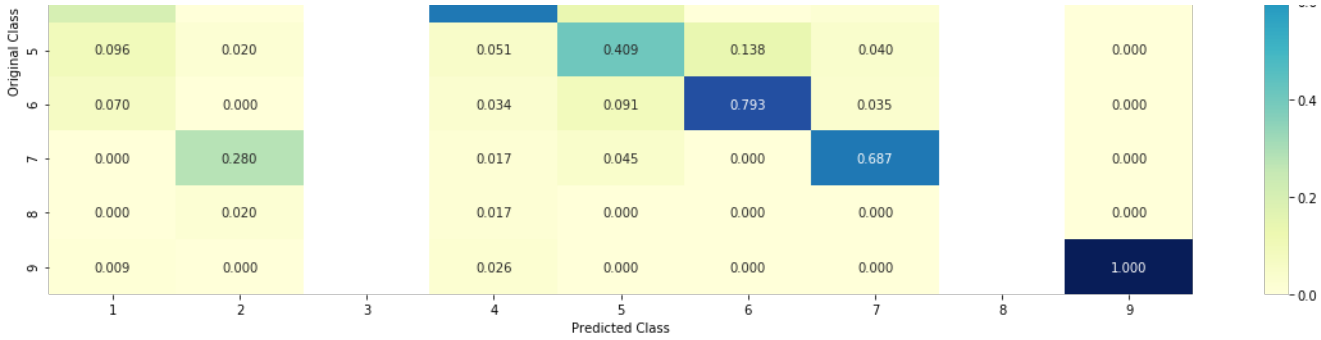
In [71]:

```
# 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_i
ter=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:
#-------------------------------

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

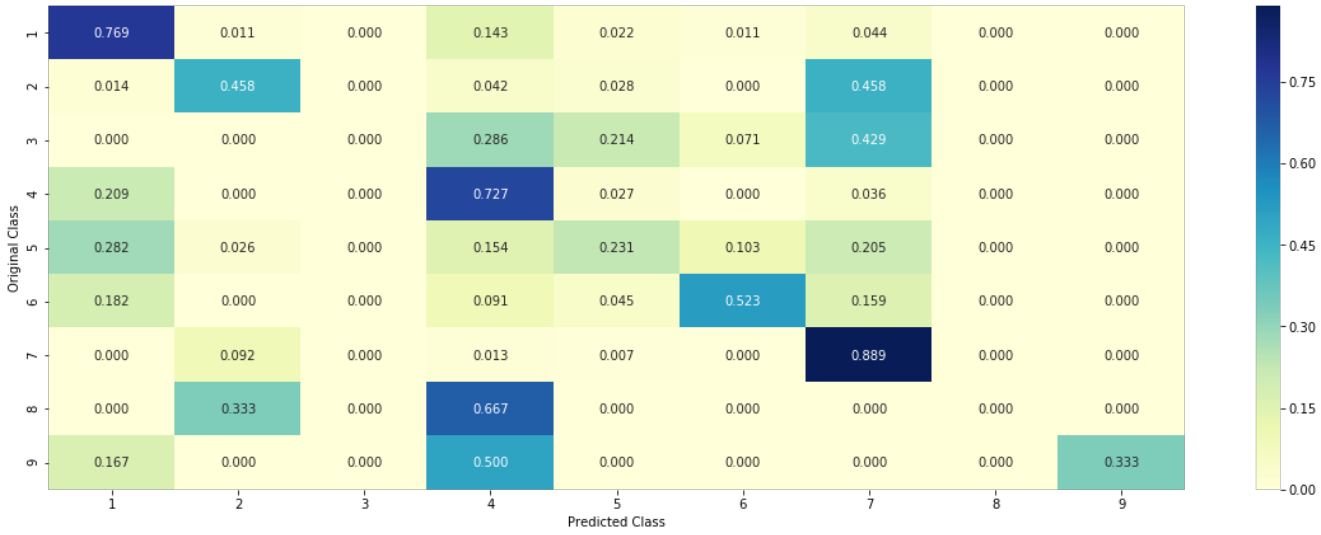
```
Log loss : 1.0527096898551893
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```


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






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



Important features from get_impfeature_names_tfidf()

In [72]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 200
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_tfidf(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc
[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[5.800e-03 3.619e-01 1.460e-02 1.000e-03 1.640e-02 4.800e-03 5.949
e-01
  5.000e-04 1.000e-04]]
Actual Class : 7
--------------------------------------------------
124 Text feature [13] present in test data point [True]
194 Text feature [10] present in test data point [True]
370 Text feature [100] present in test data point [True]
441 Text feature [11] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

Using CountVectorizer with bigram and trigram

In [73]:

```
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding_bigram = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_bigram = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_bigram = gene_vectorizer.transform(cv_df['Gene'])
```

In [74]:

```
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding_bigram =
variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_bigram = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_bigram = variation_vectorizer.transform(cv_df['Variation'])
```

In [75]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) vector
train_text_feature_onehotCoding_bigram = normalize(train_text_feature_onehotCoding_bigram, axis=0)

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

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

In [76]:

```
train_gene_var_onehotCoding_bigram =
hstack((train_gene_feature_onehotCoding_bigram,train_variation_feature_onehotCoding_bigram))
test_gene_var_onehotCoding_bigram =
hstack((test_gene_feature_onehotCoding_bigram,test_variation_feature_onehotCoding_bigram))
cv_gene_var_onehotCoding_bigram =
hstack((cv_gene_feature_onehotCoding_bigram,cv_variation_feature_onehotCoding_bigram))

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

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

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

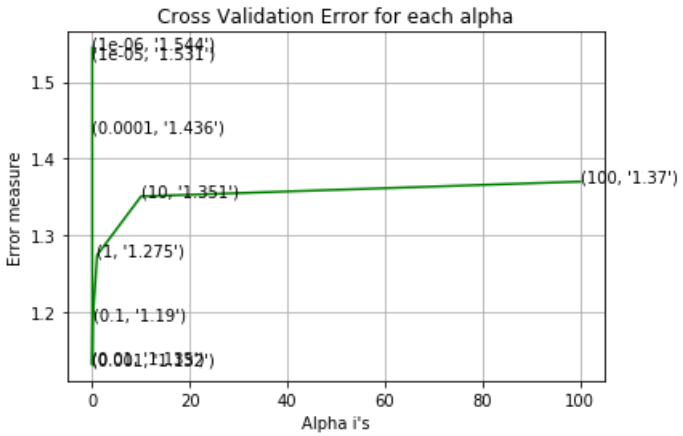
Logistic Regression with class balancing using bigrams

In [77]:

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.5444706855858827
for alpha = 1e-05
Log Loss : 1.5312579746323522
for alpha = 0.0001
Log Loss : 1.4357839399537011
for alpha = 0.001
Log Loss : 1.1317323328179836
for alpha = 0.01
Log Loss : 1.1348226112589024
for alpha = 0.1
Log Loss : 1.1899025375543038
for alpha = 1
Log Loss : 1.2745791494675336
for alpha = 10
Log Loss : 1.3510074640580056
for alpha = 100
Log Loss : 1.3702254957090563
```



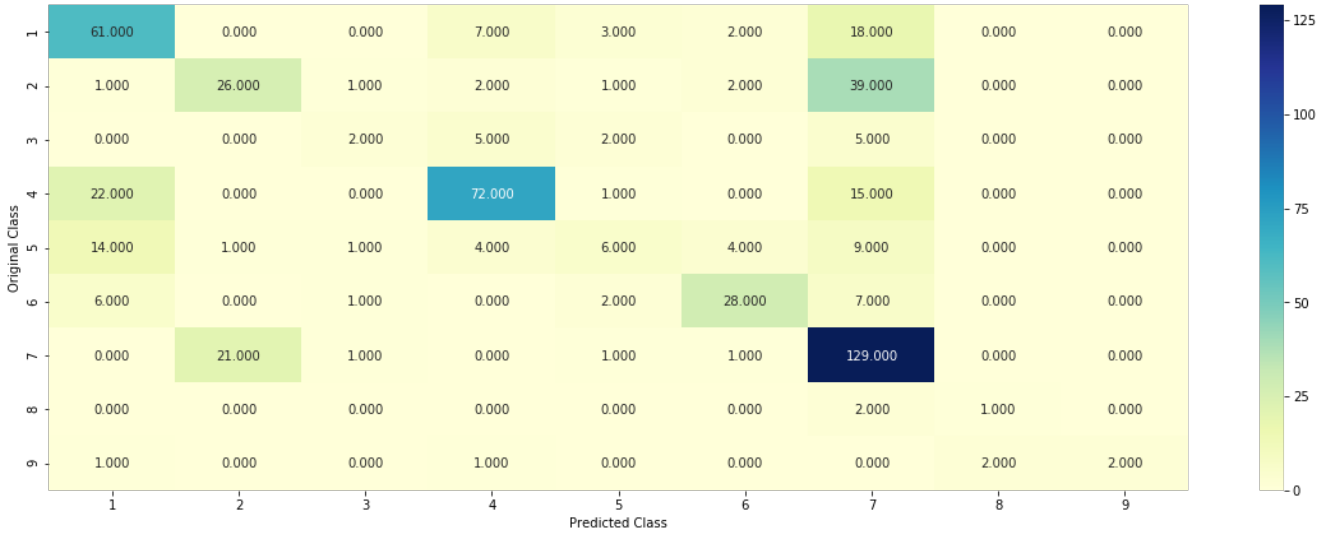

```
For values of best alpha =  0.001 The train log loss is: 0.753251975074713
For values of best alpha =  0.001 The cross validation log loss is: 1.1317323328179836
For values of best alpha =  0.001 The test log loss is: 1.1757063168958024
```

In [78]:

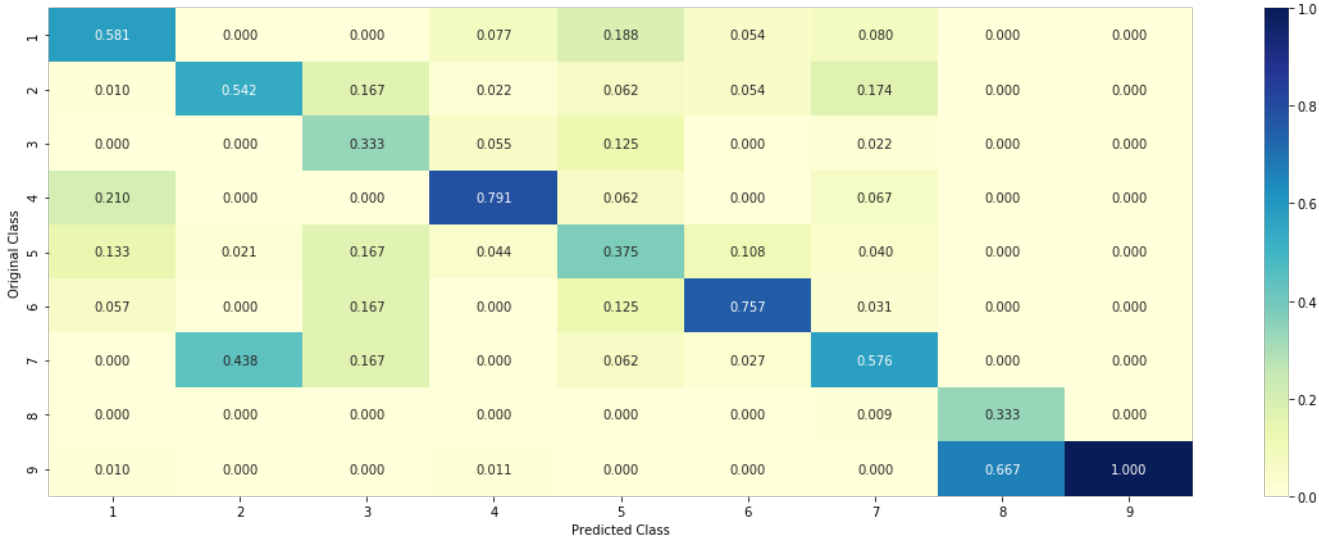
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
```

```
predict_and_plot_confusion_matrix(train_x_onehotCoding_bigram, train_y, cv_x_onehotCoding_bigram,
cv_y, clf)
```

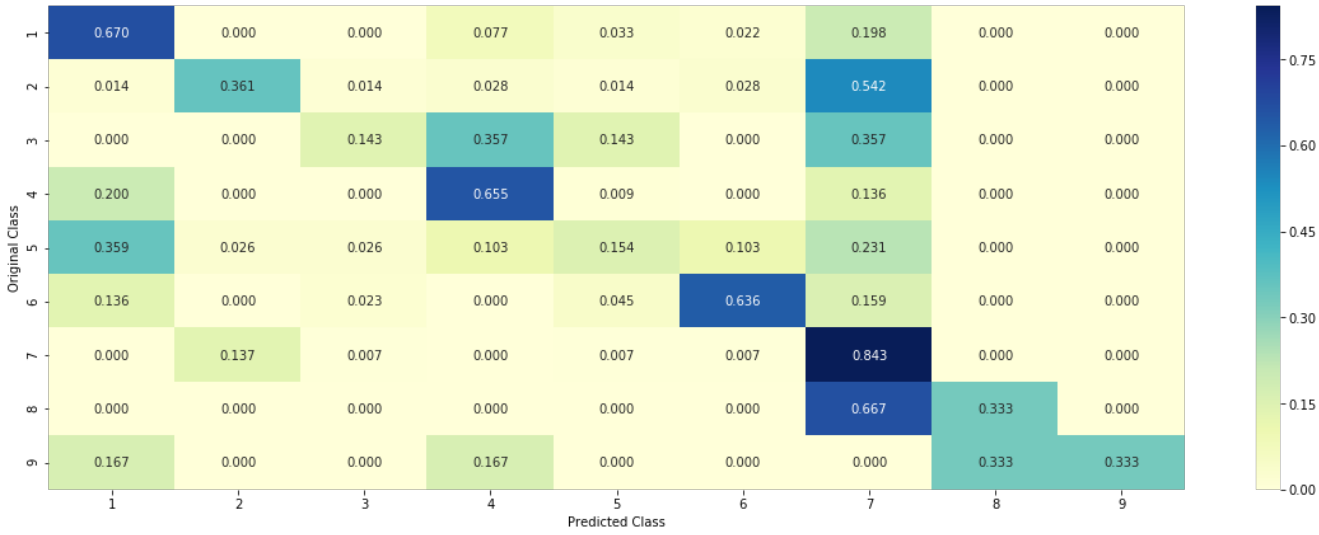
```
Log loss : 1.1317323328179836
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```




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




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



4.4. Linear Support Vector Machines

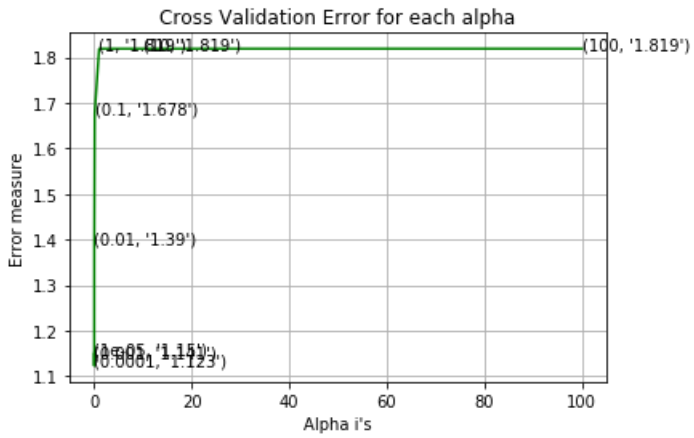
4.4.1. Hyper paramter tuning

In [79]:

```
# read more about support vector machines with linear kernals here http://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, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given 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='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', r
andom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for C = 1e-05
Log Loss : 1.1495744403460242
for C = 0.0001
Log Loss : 1.123041563243662
for C = 0.001
Log Loss : 1.1410062782646229
for C = 0.01
Log Loss : 1.3901537893194293
for C = 0.1
Log Loss : 1.677738476147656
for C = 1
Log Loss : 1.8188657439740468
for C = 10
Log Loss : 1.8188657378119724
for C = 100
Log Loss : 1.8188657627533449
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5847247973032835
For values of best alpha =  0.0001 The cross validation log loss is: 1.123041563243662
For values of best alpha =  0.0001 The test log loss is: 1.087065018350493
```

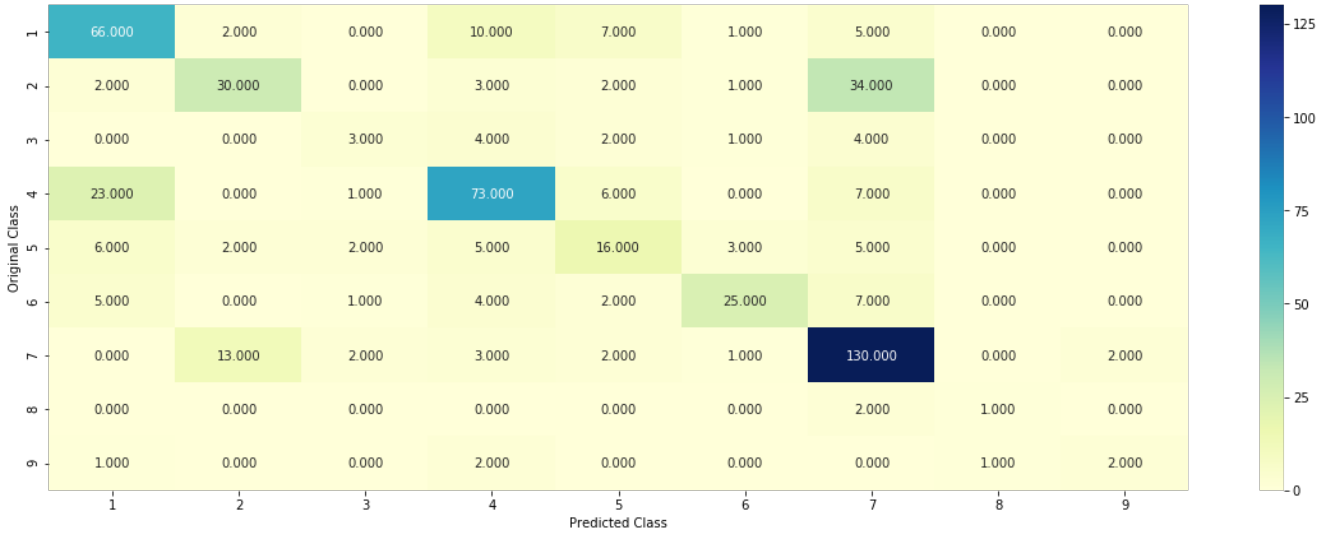
4.4.2. Testing model with best hyper parameters

In [80]:

```
# read more about support vector machines with linear kernals here http://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, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given 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/
# --------------------------------
```

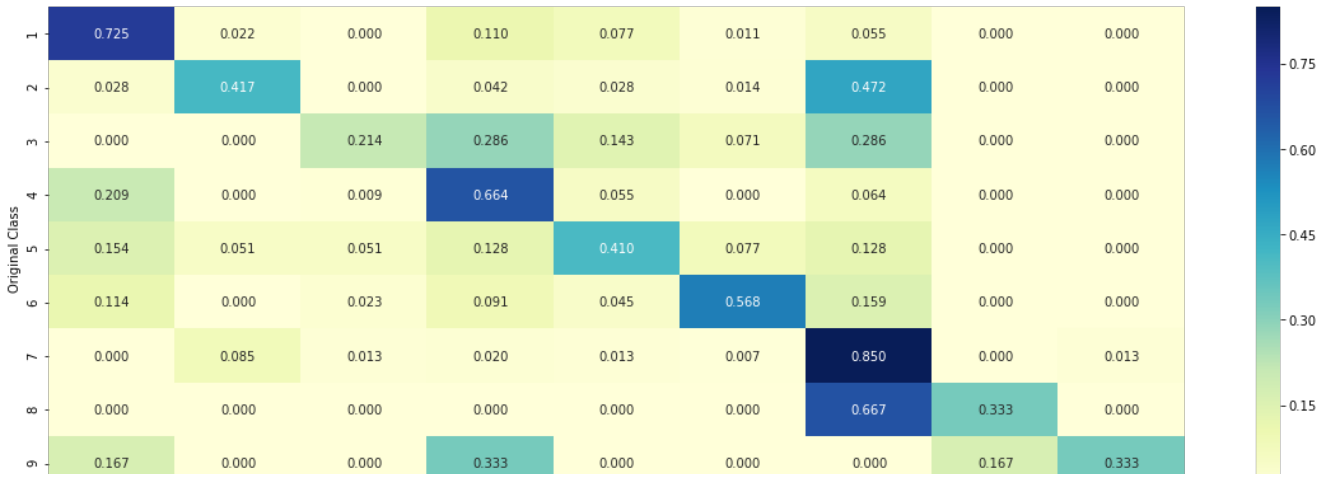
```
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)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

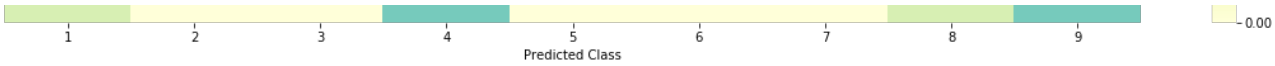
```
Log loss : 1.123041563243662
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




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


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





Feature Importance using get_impfeature_names_tfidf()

In [81]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5977 0.0735 0.0165 0.1503 0.0816 0.0315 0.0456 0.0012 0.0021]]
Actual Class : 1
--------------------------------------------------
150 Text feature [0886] present in test data point [True]
352 Text feature [11] present in test data point [True]
401 Text feature [105] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

In [82]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[3.050e-02 1.132e-01 2.000e-04 7.141e-01 1.690e-02 4.960e-02 7.130
e-02
  7.000e-04 3.500e-03]]
Actual Class : 4
--------------------------------------------------
192 Text feature [118c] present in test data point [True]
236 Text feature [10] present in test data point [True]
434 Text feature [000] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [83]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
```

```
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_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 given 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-fores
t-and-their-construction-2/
# --------------------------------


# 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 = [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]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[i],cv_log_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)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

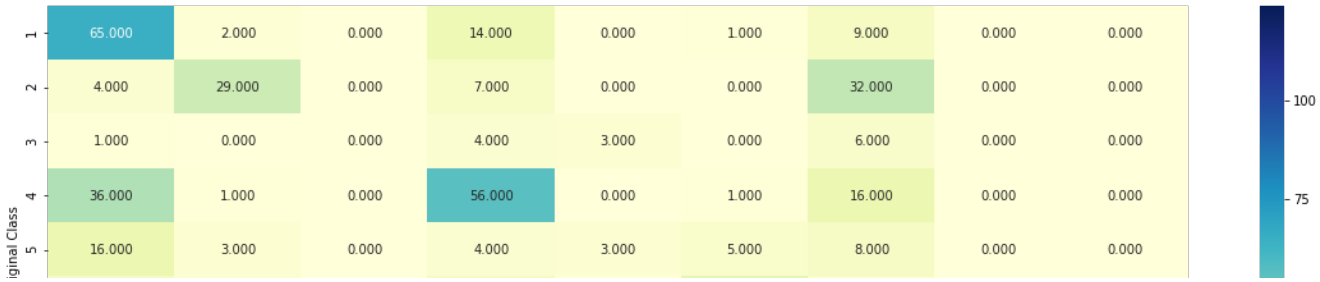
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.249935609933836
for n_estimators = 100 and max depth =  10
Log Loss : 1.2641790193170712
for n_estimators = 200 and max depth =  5
Log Loss : 1.245727999206042
for n_estimators = 200 and max depth =  10
Log Loss : 1.2576724689298706
for n_estimators = 500 and max depth =  5
Log Loss : 1.2404025851839897
for n_estimators = 500 and max depth =  10
Log Loss : 1.255978839865334
for n_estimators = 1000 and max depth =  5
Log Loss : 1.23527891840472
for n_estimators = 1000 and max depth =  10
Log Loss : 1.254970383218532
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2325185829220278
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2518214276775816
For values of best estimator =  2000 The train log loss is: 0.8473319955895278
For values of best estimator =  2000 The cross validation log loss is: 1.232518582922028
For values of best estimator =  2000 The test log loss is: 1.2174382741076177
```

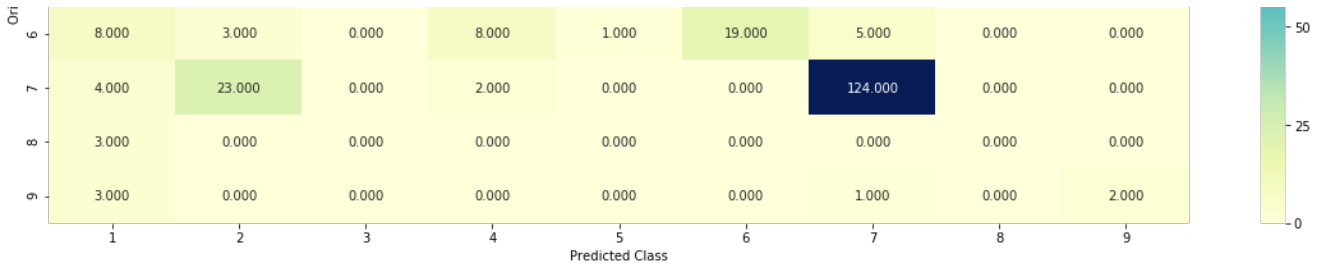
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [84]:

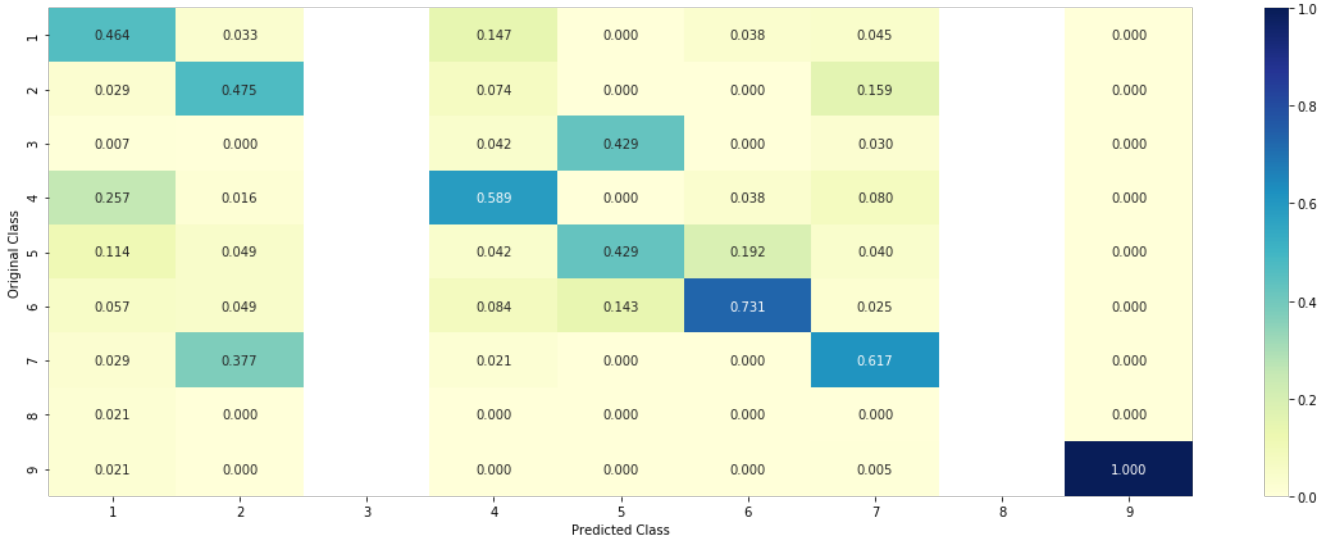
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_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 given 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-fores
t-and-their-construction-2/
# --------------------------------

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

```
Log loss : 1.232518582922028
Number of mis-classified points : 0.4398496240601504
-------------------- Confusion matrix --------------------
```





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



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

4.5.3. Feature Importance

In [85]:

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

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

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4555 0.0275 0.0112 0.3416 0.05   0.0453 0.0556 0.0043 0.009 ]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

In [86]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.202  0.0437 0.0111 0.5399 0.0545 0.0928 0.0454 0.0045 0.0062]]
Actuall Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [87]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_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 given 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-fores
t-and-their-construction-2/
# -------------------------------


# 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
```

```
# 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]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[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)], criterion='gini', max_depth=max
_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:"
,log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_
test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.132092222864272
for n_estimators = 10 and max depth =  3
Log Loss : 1.626756959478506
for n_estimators = 10 and max depth =  5
Log Loss : 1.7488842843251684
for n_estimators = 10 and max depth =  10
Log Loss : 1.7410866709951807
for n_estimators = 50 and max depth =  2
Log Loss : 1.6818395876262373
for n_estimators = 50 and max depth =  3
Log Loss : 1.3971095271759628
for n_estimators = 50 and max depth =  5
Log Loss : 1.4229411111525658
for n_estimators = 50 and max depth =  10
Log Loss : 1.647013406977923
for n_estimators = 100 and max depth =  2
Log Loss : 1.540054375446614
for n_estimators = 100 and max depth =  3
Log Loss : 1.4477543405306252
for n_estimators = 100 and max depth =  5
Log Loss : 1.4088787530197973
for n_estimators = 100 and max depth =  10
Log Loss : 1.6805753119723683
for n_estimators = 200 and max depth =  2
Log Loss : 1.598982470733
for n_estimators = 200 and max depth =  3
```

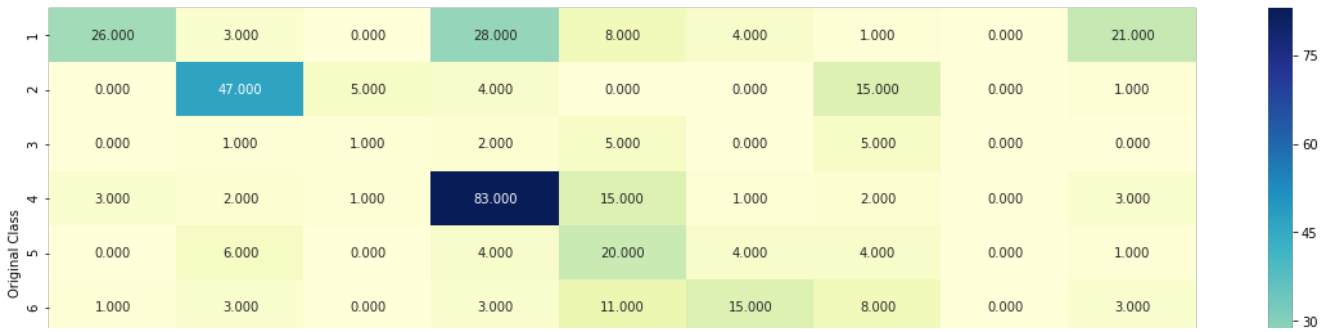
```
Log Loss : 1.4362564961770856
for n_estimators = 200 and max depth =  5
Log Loss : 1.425106650450802
for n_estimators = 200 and max depth =  10
Log Loss : 1.6966183485262034
for n_estimators = 500 and max depth =  2
Log Loss : 1.6334135639704894
for n_estimators = 500 and max depth =  3
Log Loss : 1.502581265751669
for n_estimators = 500 and max depth =  5
Log Loss : 1.4568798202717743
for n_estimators = 500 and max depth =  10
Log Loss : 1.737447470984081
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6254176904615565
for n_estimators = 1000 and max depth =  3
Log Loss : 1.501166314831916
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4206217805960257
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7467815423521034
For values of best alpha =  50 The train log loss is: 0.14232835396844948
For values of best alpha =  50 The cross validation log loss is: 1.3971095271759626
For values of best alpha =  50 The test log loss is: 1.4181237144940935
```

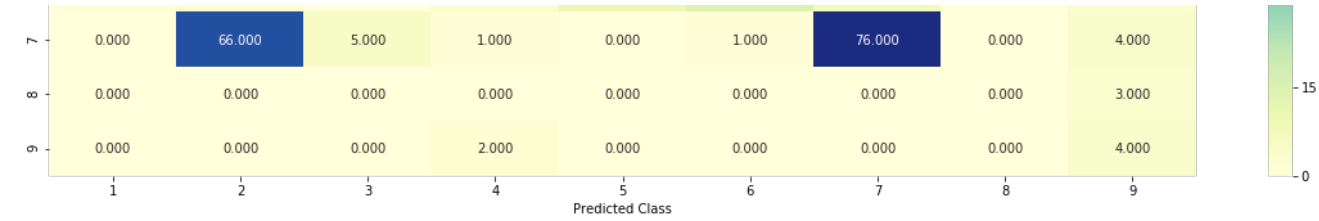
4.5.4. Testing model with best hyper parameters (Response Coding)

In [88]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_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 given 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='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.3971095271759628
Number of mis-classified points : 0.48872180451127817
-------------------- 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 [89]:

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


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0785 0.0027 0.0294 0.7382 0.0447 0.0488 0.0015 0.0239 0.0323]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [90]:

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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
```

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

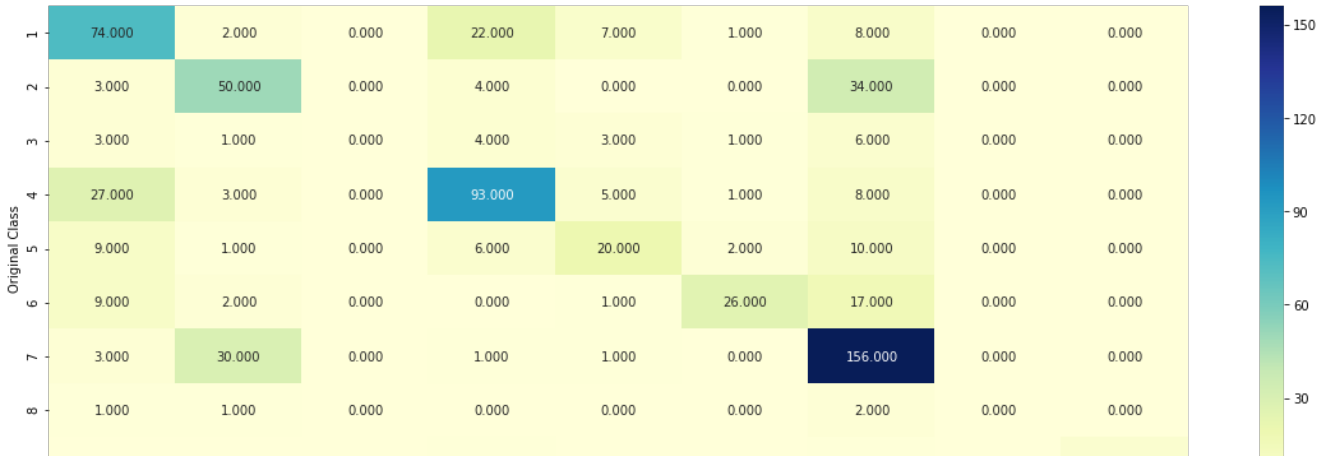
```
Logistic Regression :  Log Loss: 1.07
Support vector machines : Log Loss: 1.82
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.035
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.514
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.155
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.180
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.272
```

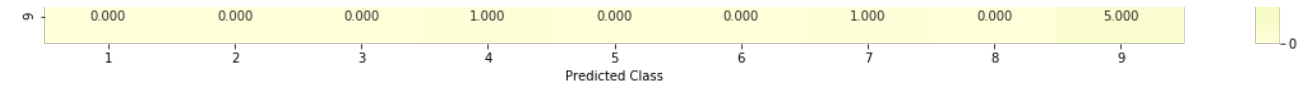
4.7.2 testing the model with the best hyper parameters

In [91]:

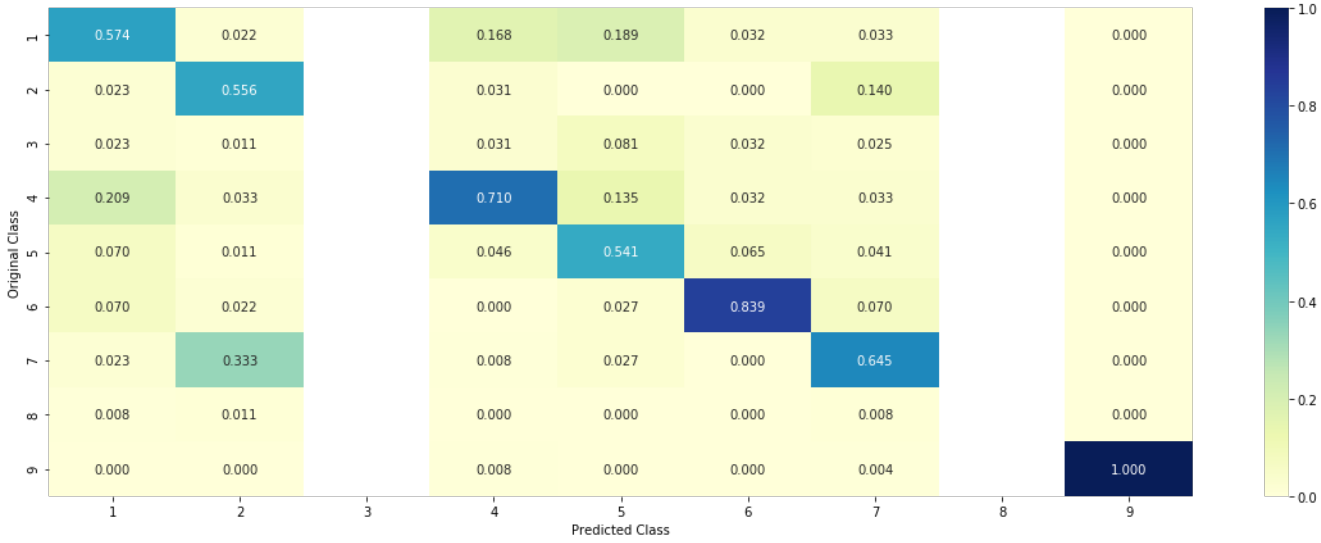
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.8015576620944015
Log loss (CV) on the stacking classifier : 1.1550156767409936
Log loss (test) on the stacking classifier : 1.1435228634704806
Number of missclassified point : 0.362406015037594
-------------------- Confusion matrix --------------------
```

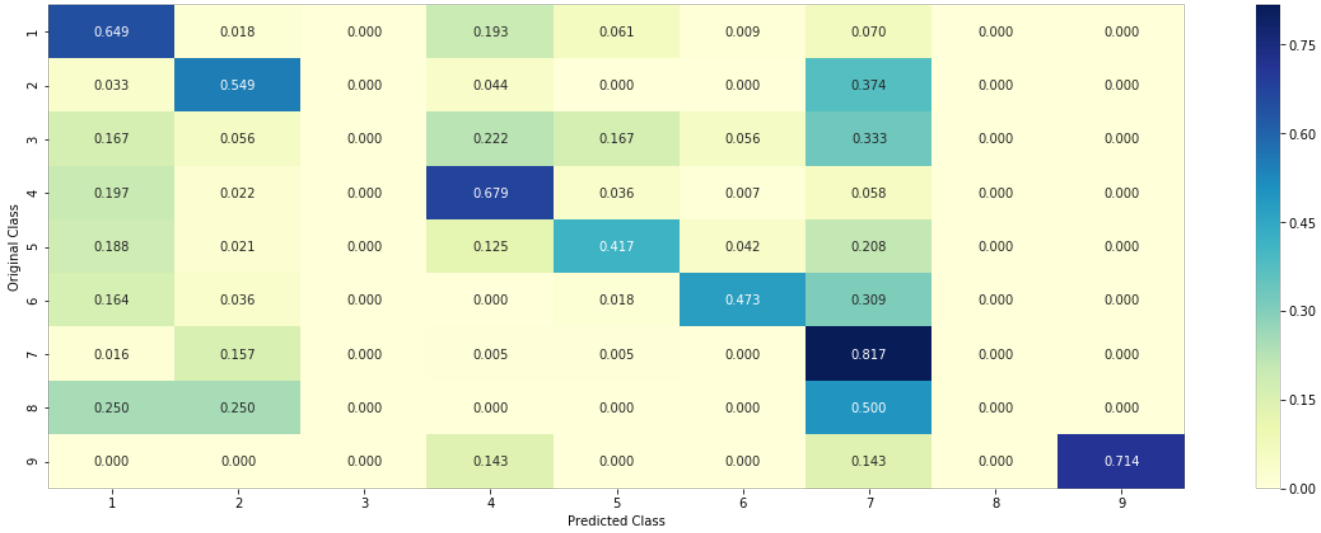




---------- Precision matrix (Columm Sum=1) ----------



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



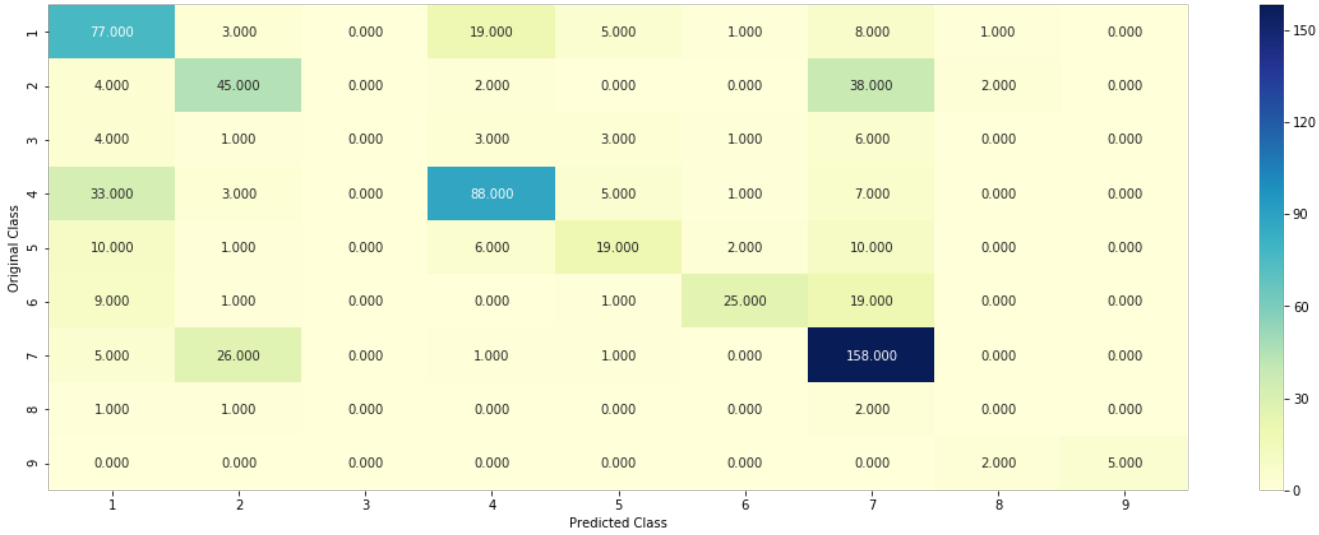
4.7.3 Maximum Voting classifier

In [92]:

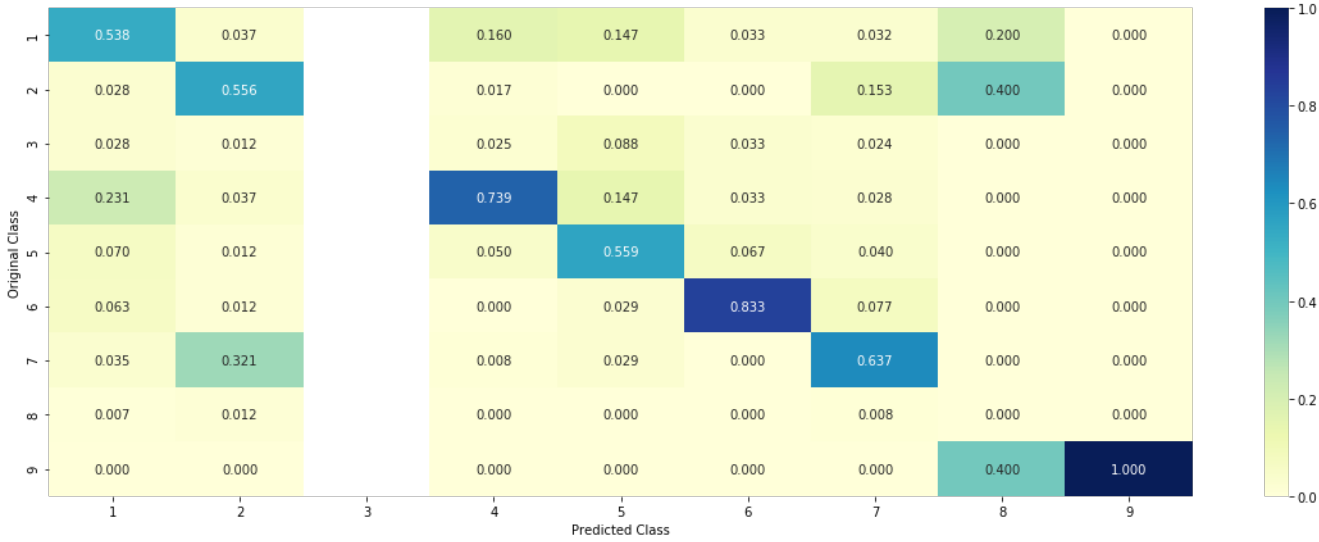
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y,
vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)-
test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.9429178578265246
Log loss (CV) on the VotingClassifier : 1.2094069452016218
```

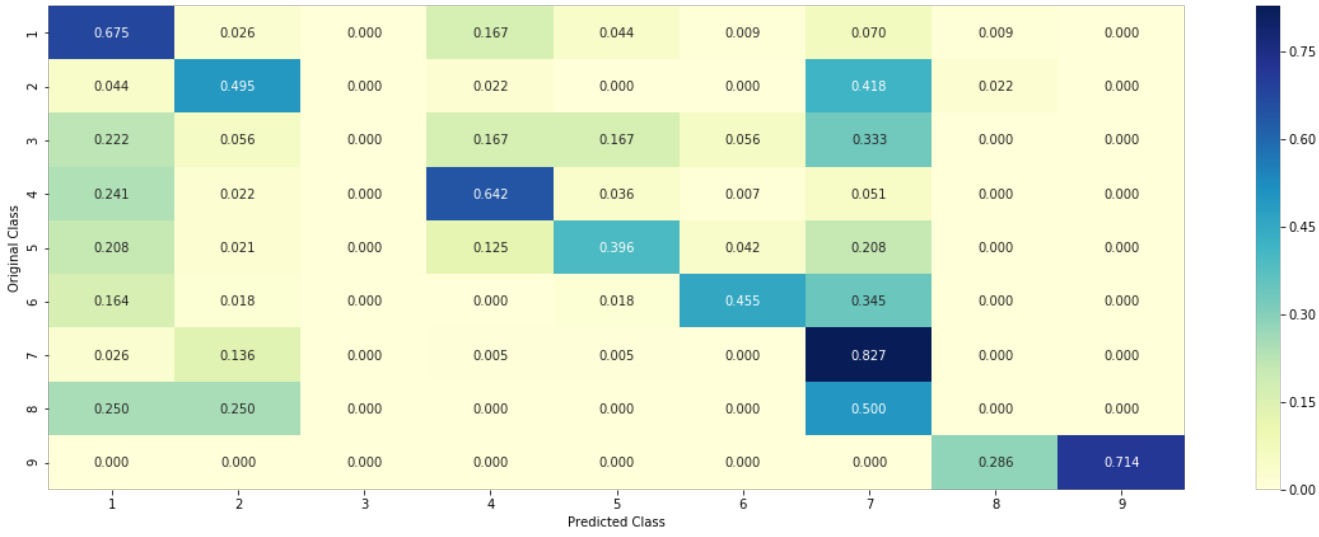
Log loss (test) on the VotingClassifier : 1.1953007125705173
Number of missclassified point : 0.37293233082706767
---------- Confusion matrix ----------



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



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



Observation and Summary:

Observation and Summary:

- In Logistic regression model with class balancing and by using tfidf vectorizer we got a log loss of 1.005
- Performed tfidf vectorizer on each model and found better logloss (i.e reduced logloss) for each model
- Performed count vectorizer with (including bigram) on Logistic regression model and got a logloss of 1.17
- The best model we got is Logistic regression with 1.005 test logloss using tfidf vectorization

In [104]:

```
from prettytable import PrettyTable
x = PrettyTable(['model' , 'Train logloss' , 'cv logloss' , 'Test logloss' , 'missclassified points
'])
x.add_row(['naive bayes',0.74,1.2,1.16,0.37])
x.add_row(['knn',0.618,1.06,1.09,0.34])
x.add_row(['logistic regression with class balancing',0.56,1.02,1.005,0.34])
x.add_row(['logistic regression without class balancing',0.55,1.05,1.01,0.33])
x.add_row(['logistic regression with class balancing\n using Bi-gram',0.753,1.13,1.17,0.385])
x.add_row(['linear svm',0.58,1.123,1.087,0.34])
x.add_row(['random forest',0.847,1.23,1.217,0.43])
x.add_row(['max voting classifier',0.942,1.20,1.195,0.372])

print(x)
```

```
+-------------------------------------------+--------------+------------+-------------+--------
------------+
|                   model                   | Train logloss | cv logloss | Test logloss |
missclassified points |
+-------------------------------------------+--------------+------------+-------------+--------
------------+
|                naive bayes                |      0.74     |    1.2     |     1.16     |
0.37         |
|                  knn                      |     0.618     |    1.06    |     1.09     |
0.34         |
|   logistic regression with class balancing  |      0.56     |    1.02    |    1.005     |
0.34         |
| logistic regression without class balancing |      0.55     |    1.05    |     1.01     |
0.33         |
|   logistic regression with class balancing  |     0.753     |    1.13    |     1.17     |
0.385         |
|                using Bi-gram                |               |            |              |
|
|                 linear svm                  |      0.58     |   1.123    |    1.087     |
0.34         |
|               random forest                 |     0.847     |    1.23    |    1.217     |
0.43         |
|            max voting classifier            |     0.942     |    1.2     |    1.195     |
.372         |
+-------------------------------------------+--------------+------------+-------------+--------
------------+
```