Document Created date:25-09-2019

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/ (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 (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 (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 (https://www.youtube.com/watch?v=UwbuW7oK8rk)
3. https://www.youtube.com/watch?v=qxXRKVompI8 (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 (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 (https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation)](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

```
Step by Step Procedure i followed to achieve the target.
```

- Understanding the Businessreal world problem
- Understanding Real-world/Business objectives and constraints.
- Understanding Data overview
- Mapping the real-world problem to an ML problem
- Defining Performance Metric as per the ML problem
- importing required libraries
- Reading the gene , variation and text data,
- Exploratory Data Analysis
- Preprocessing text
- splitting the data into train, test, cv
- ploting Distribution of y_i's in Train, Test and Cross Validation datasets
- Prediction using a 'Random' Model
- Univariate Analysis on gene feature
- Univariate Analysis on variation feature
  - Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
- Conclusion - Univariate Analysis of Feature Engineering
- Stacking the three types of features (gene , variation and text)
- Apply Base Line Model(naive Bayes) to check the error and model performance , then we can apply correct ML model on top of that.
  - Baseline Model - Hyper parameter Tuning
  - Testing the model using the best hyperparameter
  - ploting Confusion matrix
  - Feature Importance, Correctly classified point
  - Feature Importance, incorrectly classified point
- Apply KNN model
  - Hyper parameter Tuning
  - Testing the model using the best hyperparameter
  - ploting Confusion matrix
  - Sample Query point -1 and Sample Query point -2
- Apply Logistic Model - Class balancing
  - Hyper parameter Tuning
  - Testing the model using the best hyperparameter
  - ploting Confusion matrix
  - Feature Importance, Correctly classified point
  - Feature Importance, incorrectly classified point
- Apply Logistic Model - without Class balancing
  - Hyper parameter Tuning
  - Testing the model using the best hyperparameter
  - ploting Confusion matrix
  - Feature Importance, Correctly classified point
  - Feature Importance, incorrectly classified point
- Applt Linear Support Vector Machines

    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - ploting Confusion matrix
    - Feature Importance, Correctly classified point
    - Feature Importance, incorrectly classified point
- Apply Random Forest Classifier (With One hot Encoding - Response Coding)
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - ploting Confusion matrix
    - Feature Importance, Correctly classified point
    - Feature Importance, incorrectly classified point
- Apply Random Forest Classifier (Without One hot Encoding)
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - ploting Confusion matrix
    - Feature Importance, Correctly classified point
    - Feature Importance, incorrectly classified point
- Stack the models
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - ploting Confusion matrix
    - Maximum Voting classifier
- **Applying all the models with TFIDF-Features - Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values**
- Univariate analysis - gene Feature
    - How to featurize this Gene feature ? , How good is this gene feature in predicting y_i?
    - Univariate analysis - variation Feature
    - How good is this gene feature in predicting y_i?
    - Univariate analysis - Text Feature
    - How good is this gene feature in predicting y_i?
- Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
- Apply ML models - Stacking the three types of features (gene , variation and Tex)
- Applly all the models along with hyperparmeter tuning , testing model , ploting confusion matrix, feature importance stpes

- **Logistic Regression with CountVectorizer(unigrams, bigrams)**
- Univariate Analysis on Gene Feature - CountVectorizer(unigrams , bi-grams)
- Univariate Analysis on Variation Feature - CountVectorizer(unigrams , bi-grams)
- Univariate Analysis on text Feature - CountVectorizer(unigrams , bi-grams)
- Normalize every feature and data preparation for ml models
- Stacking the three types of features (gene , variation and Tex)
- Apply Logistic Model - Class balancing
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - ploting Confusion matrix
    - Feature Importance, Correctly classified point
    - Feature Importance, incorrectly classified point
- Apply Logistic Model - without Class balancing
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - Ploting Confusion matrix
    - Feature Importance, Correctly classified point

    - Feature Importance, incorrectly classified point
- Observation on model performences (Conclusion)
- **Trying the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0**
- Univariate Analysis on Gene Feature - tfidfVectorizer(unigrams, bi-grams)
- Univariate Analysis on Variation Feature - tfidfVectorizer(unigrams, bi-grams)
- Univariate Analysis on text Feature - tfidfVectorizer(unigrams, bi-grams)
- Stacking the two types of features (gene and Text)
- Normalize every feature and data preparation for ml models
- Logistic Regression - With Class balancing Tfidf(bigrams) - Gene with Text
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - Ploting Confusion matrix
    - Feature Importance, Correctly classified point
    - Feature Importance, incorrectly classified point
- Logistic Regression without class balancing - Tfidf- Gene, Text
    - Hyper parameter Tuning
    - Testing the model using the best hyperparameter
    - Ploting Confusion matrix
    - Feature Importance, Correctly classified point
    - Feature Importance, incorrectly classified point
- Observation on overall model performences (Conclusion)
- Ploting the performences by table format.

- **I have keep in mind and didn't forget Our main objective : is to minimize the loss to lees than 1 , till the last line of code through out the steps .

---

3. Exploratory Data Analysis

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

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.
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

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

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

```
# loading stop words from nltk library
import nltk
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))


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

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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 59.956308953999994 seconds
```

In [6]:

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

Out[6]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

In [7]:

```
result[result.isnull().any(axis=1)] # checking if there are null values are present are
```

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]:

```
# filling the emptty cells with combining both gene and variation data
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [9]:

```
result[result['ID']==1109] # displaying the  filled nana cells
```

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

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]:

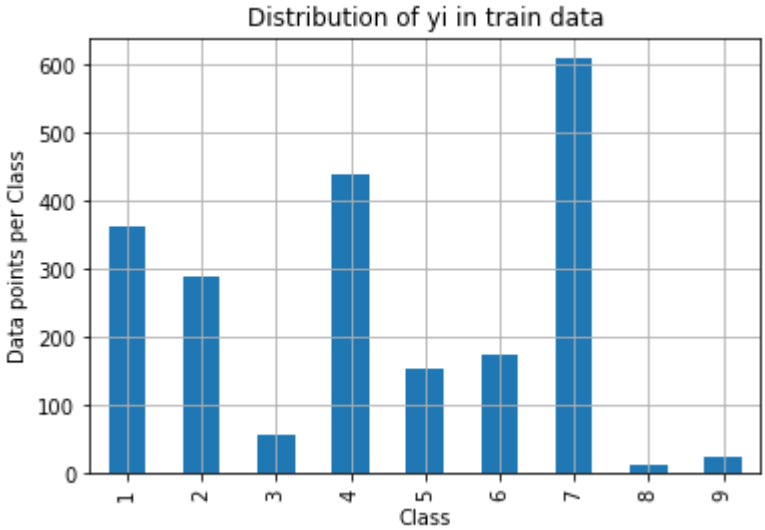
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

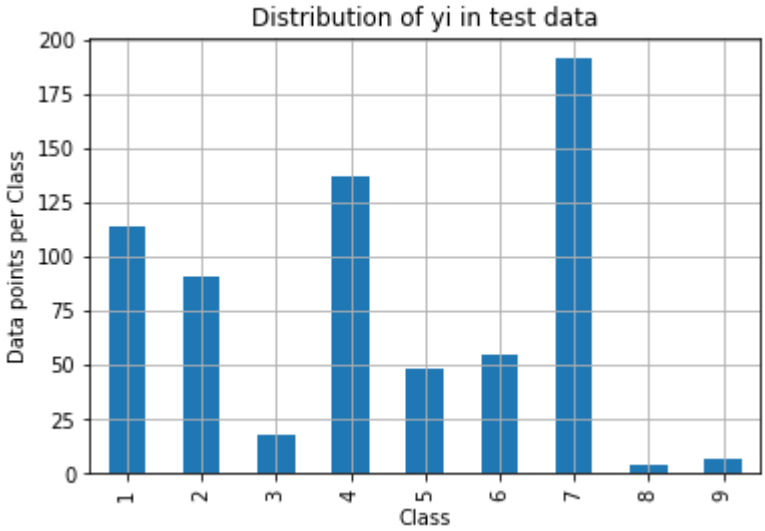
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:

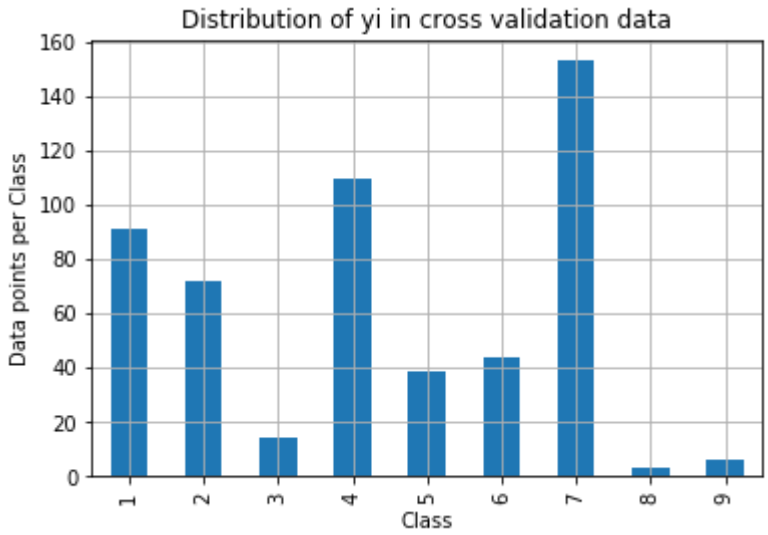
```
# cite : https://stackoverflow.com/questions/34656980/attributeerror-series-object-has-
# it returns a dict, keys as class labels and values as the number of data points in th
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
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]


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],

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], '
```




```
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.

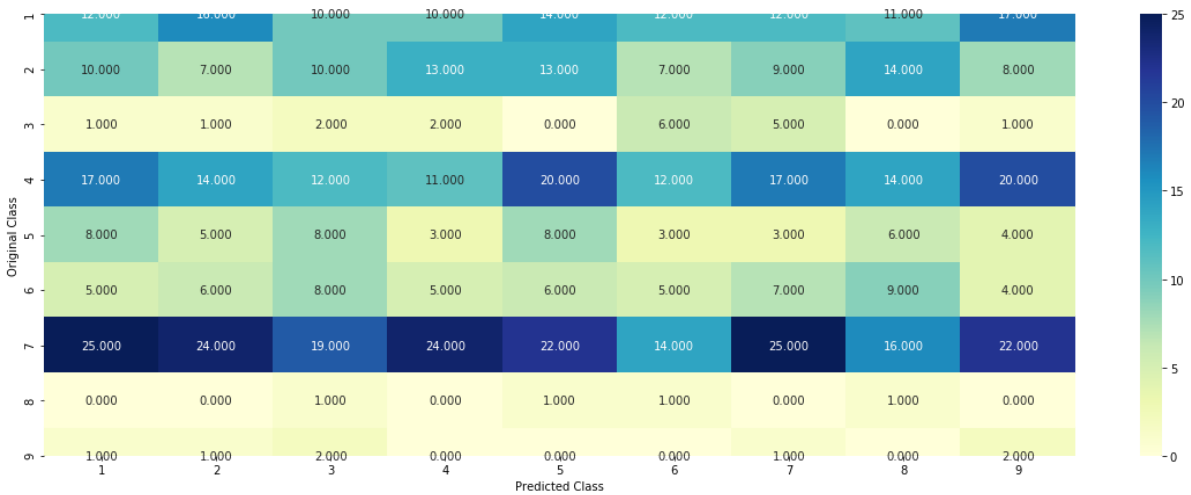
In [13]:

```
# 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 predic

    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 tı
    # 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 tı
    # 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, yticklabel
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

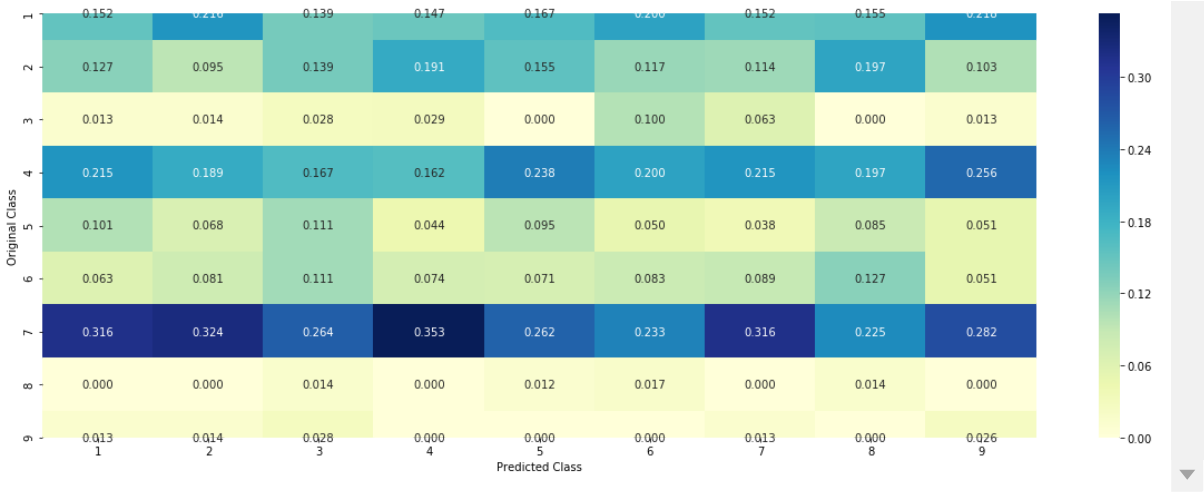
In [14]:

```
# 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_


# 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=

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

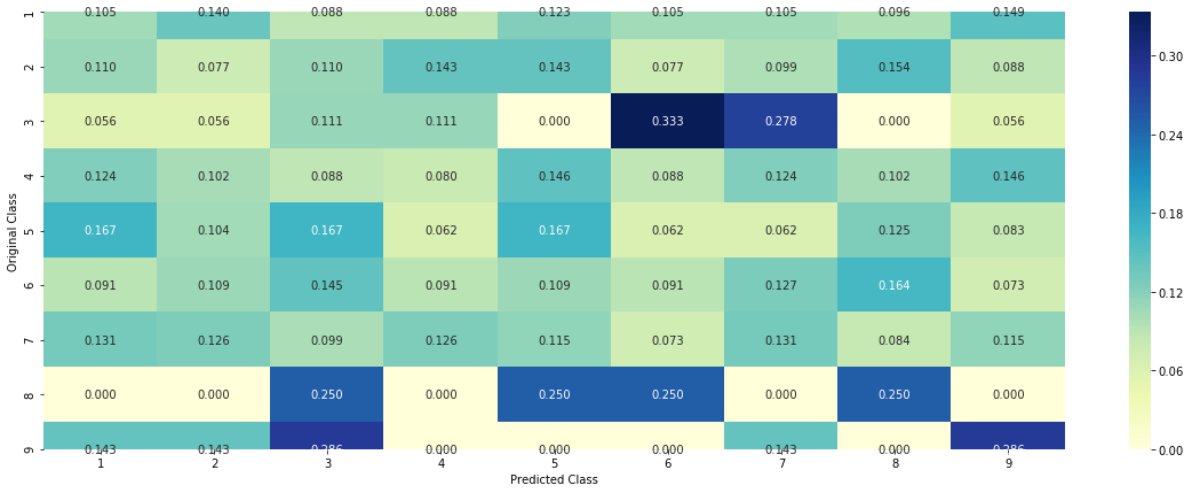
```
Log loss on Cross Validation Data using Random Model 2.4863787423391823
Log loss on Test Data using Random Model 2.5242323567896485
-------------------- Confusion matrix --------------------
```




```
-------------------- 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 dat
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10
# 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,
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in wh
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pertic
        # 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 particul
            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, 0.03787878787878788, 0.06818181818181818177, 0..
    #      'TP53': [0.32142857142857145, 0.06122448979591836 6, 0.06122448979591836 6, 0..
    #      'EGFR': [0.05681818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.0
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.
    #      ...
    #     }
    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 valu
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in
    # 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 : 229
BRCA1     162
TP53       97
EGFR       95
PTEN       92
BRCA2      82
KIT        62
BRAF       57
ALK        44
ERBB2      43
PIK3CA     38
Name: Gene, dtype: int64
```

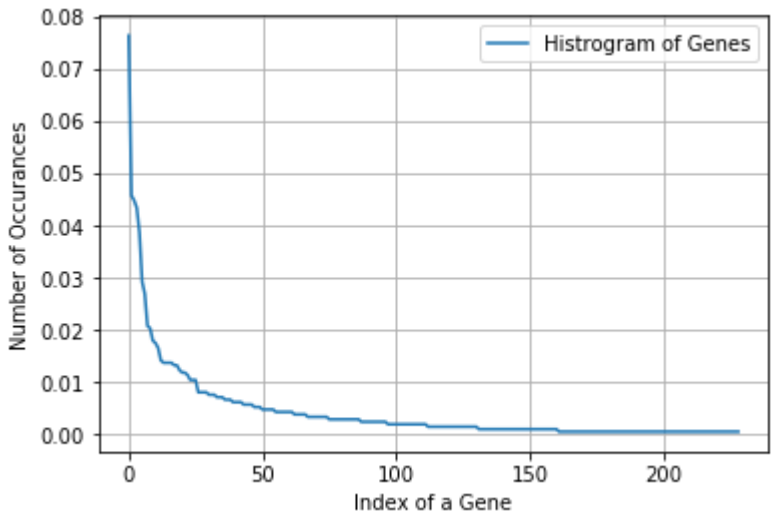
In [17]:

```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the tr
```

```
Ans: There are 229 different categories of genes in the train data, and th
ey are distibuted as follows
```

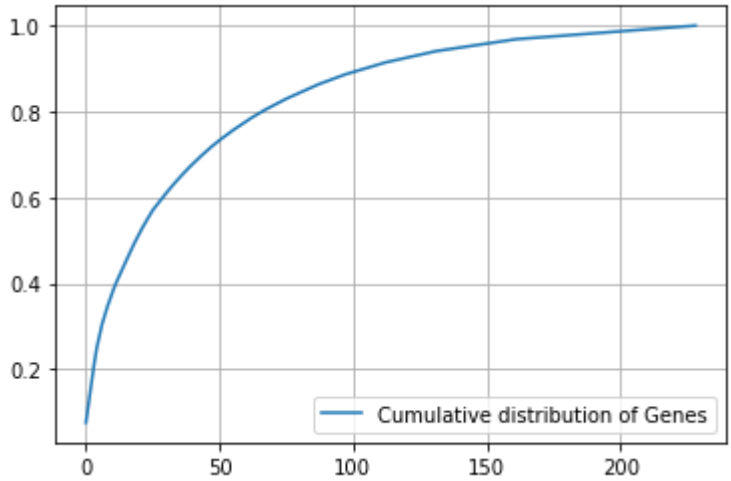
In [18]:

```
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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

1. One hot Encoding (CountVectorizer)
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 meth
```

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

In [22]:

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

In [23]:

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

Out[23]:

```
1797        AR
502       TP53
2380    PTPN11
87       CCNE1
1925       SMO
Name: Gene, dtype: object
```

In [24]:

```
gene_vectorizer.get_feature_names()[:10]
```

Out[24]:

```
['abl1', 'acvr1', 'ago2', 'akt1', 'akt2', 'akt3', 'alk', 'apc', 'ar', 'ara
f']
```

In [25]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding meth
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encodin
g method. The shape of gene feature: (2124, 229)
```

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

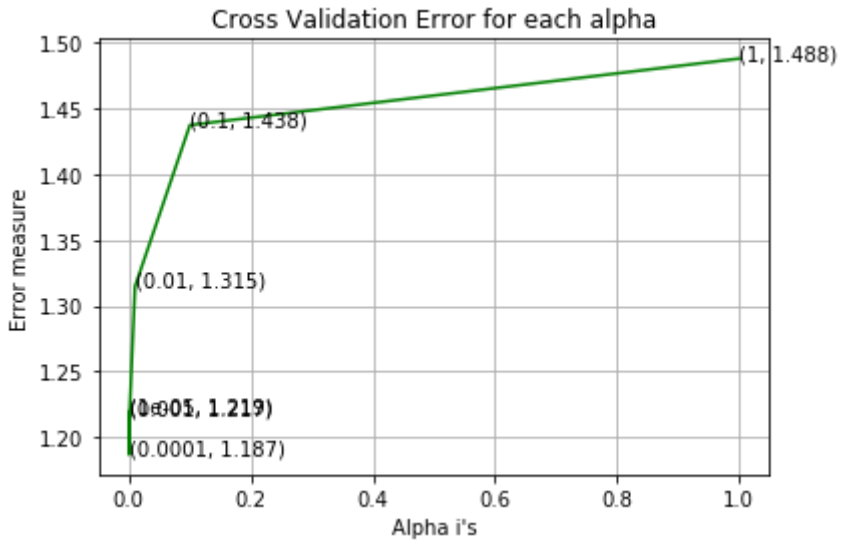
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

```
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/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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, lab

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

```
For values of alpha =  1e-05 The log loss is: 1.2191723935068697
For values of alpha =  0.0001 The log loss is: 1.1866621639086434
For values of alpha =  0.001 The log loss is: 1.216819587421816
For values of alpha =  0.01 The log loss is: 1.3146104694275906
```

```
For values of alpha =  0.1 The log loss is: 1.437520887294074
For values of alpha =  1 The log loss is: 1.4881135574667714
```




```
For values of best alpha =  0.0001 The train log loss is: 0.97606389518064
78
For values of best alpha =  0.0001 The cross validation log loss is: 1.186
6621639086434
For values of best alpha =  0.0001 The test log loss is: 1.230763328806195
2
```

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 [27]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_ge

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_covera
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cove
```

```
Q6. How many data points in Test and CV datasets are covered by the  229
genes in train dataset?
Ans
1. In test data 640 out of 665 : 96.2406015037594
2. In cross validation data 514 out of  532 : 96.61654135338345
```

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 [28]:

```
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                      24
Overexpression                5
Q61H                          3
G12S                          2
G67R                          2
Promoter_Hypermethylation     2
P34R                          2
Name: Variation, dtype: int64
```

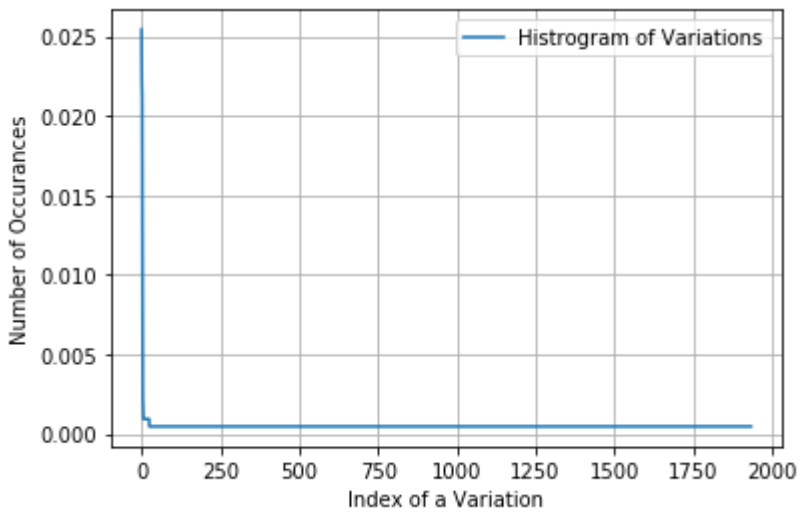
In [29]:

```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations
```

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

In [30]:

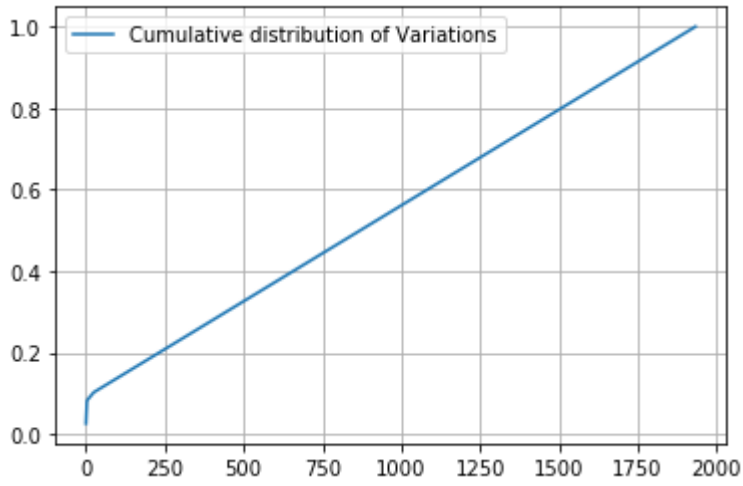
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:

```
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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)](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 [32]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tr
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tes
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df
```

In [33]:

```
print("train_variation_feature_responseCoding is a converted feature using the response
```

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

In [34]:

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

In [35]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot en
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-
hot encoding method. The shape of Variation feature: (2124, 1965)
```

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

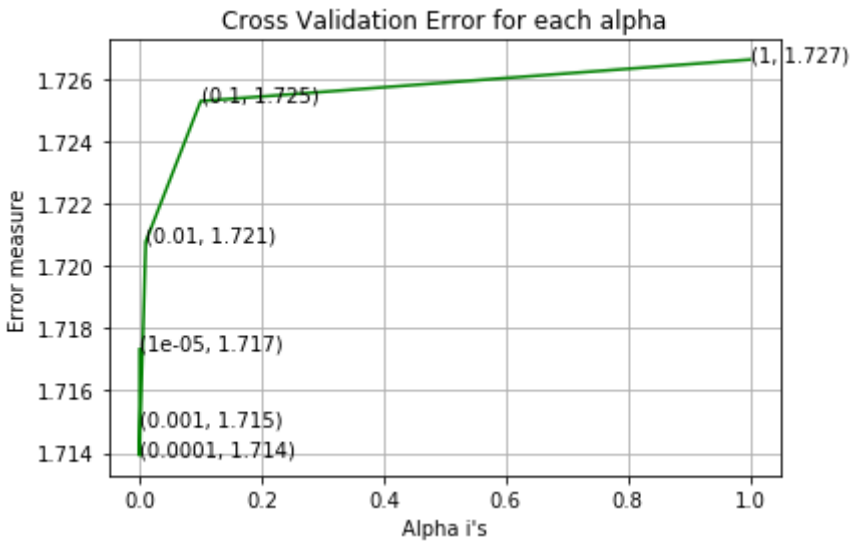
Let's build a model just like the earlier!

In [36]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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, lab

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

```
For values of alpha =  1e-05 The log loss is: 1.717330635154986
For values of alpha =  0.0001 The log loss is: 1.7139133414417451
For values of alpha =  0.001 The log loss is: 1.7148904146798534
```

```
For values of alpha =  0.01 The log loss is: 1.7207883105331212
For values of alpha =  0.1 The log loss is: 1.7253020016739256
For values of alpha =  1 The log loss is: 1.7266358839103675
```




```
For values of best alpha =  0.0001 The train log loss is: 0.68841425680804
57
For values of best alpha =  0.0001 The cross validation log loss is: 1.713
9133414417451
For values of best alpha =  0.0001 The test log loss is: 1.68512590646508
```

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 [37]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " 
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation']))).shap
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_covera
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cove
```

```
Q12. How many data points are covered by total  1933  genes in test and cr
oss validation data sets?
Ans
1. In test data 76 out of 665 : 11.428571428571429
2. In cross validation data 49 out of  532 : 9.210526315789473
```

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 [38]:

```
# 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 [39]:

```
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(wo
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT
            row_index += 1
    return text_feature_responseCoding
```

In [40]:

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

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it
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 : 54503
```

In [41]:

```
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 [42]:

```
#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 [43]:

```
# 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_fea
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featur
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_resp
```

In [44]:

```
# 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 [45]:

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

In [46]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur[:10]))
```

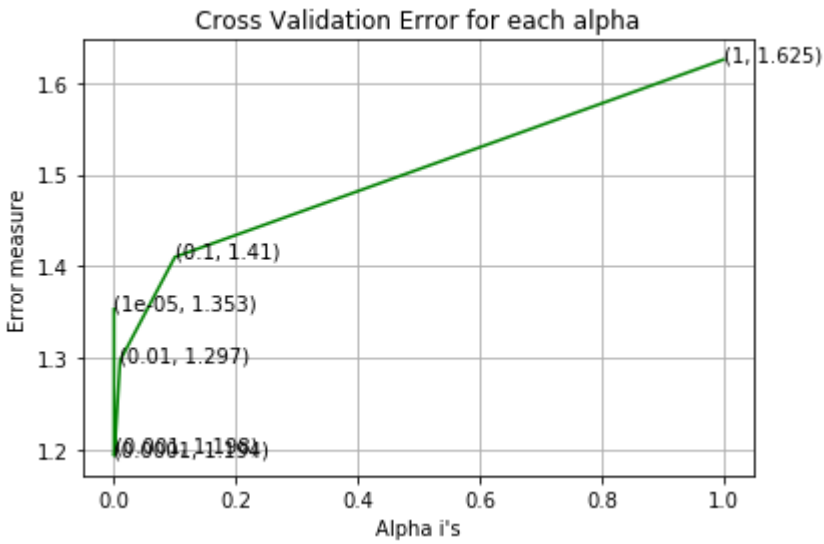
```
Counter({152709: 1, 118652: 1, 82393: 1, 67828: 1, 67689: 1, 67112: 1, 654
06: 1, 64159: 1, 63156: 1, 56138: 1})
```

In [47]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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, lab

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

```
For values of alpha =  1e-05 The log loss is: 1.3533918195863113
For values of alpha =  0.0001 The log loss is: 1.1939788611238822
```

```
For values of alpha =  0.001 The log loss is: 1.1979058331921417
For values of alpha =  0.01 The log loss is: 1.297470916618555
For values of alpha =  0.1 The log loss is: 1.410157698170598
For values of alpha =  1 The log loss is: 1.6249648078191579
```




```
For values of best alpha =  0.0001 The train log loss is: 0.67600580055840
95
For values of best alpha =  0.0001 The cross validation log loss is: 1.193
9788611238822
For values of best alpha =  0.0001 The test log loss is: 1.221234158857284
8
```

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

Ans. Yes, it seems like!

In [48]:

```
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 [49]:

```
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 da
```

```
96.631 % of word of test data appeared in train data
96.431 % of word of Cross Validation appeared in train data
```

3.2.4 Conclusion - Univariate Analysis of Feature Engineering

In [214]:

```
pt = PrettyTable()

pt.field_names = ["Feature", "Vectorizer", "Best alpha", "Train log-loss", "CV log-loss
                  "Stability ststus "]
pt.add_row(["Gene","CountVectorizer", 0.0001,  0.976, 1.186, 1.230, "Stable"])
pt.add_row(["Variation", "CountVectorizer",0.0001, 0.688, 1.713, 1.685,"Less Stable" ])
pt.add_row(["Text", "CountVectorizer",0.0001, 0.67, 1.193, 1.221, "Stable"])
print(pt)
```

```
+-----------+-----------------+------------+----------------+-------------
+----------------+-------------------+
|  Feature  |    Vectorizer   | Best alpha | Train log-loss | CV log-loss
| Test log-loss  | Stability ststus  |
+-----------+-----------------+------------+----------------+-------------
+----------------+-------------------+
|    Gene   | CountVectorizer |   0.0001   |     0.976      |    1.186
|      1.23      |       Stable      |
| Variation | CountVectorizer |   0.0001   |     0.688      |    1.713
|     1.685      |    Less Stable    |
|    Text   | CountVectorizer |   0.0001   |      0.67      |    1.193
|     1.221      |       Stable      |
+-----------+-----------------+------------+----------------+-------------
+----------------+-------------------+
```

4. Machine Learning Models

In [50]:

```
#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 e
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_
    plot_confusion_matrix(test_y, pred_y)
```

In [51]:

```
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 [52]:

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

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

Stacking the three types of features

In [53]:

```
# 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_f
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feat
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_on

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_vari
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variati
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fea

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_r
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_resp
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCo
```

In [54]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCo
print("(number of data points * number of features) in test data = ", test_x_onehotCodi
print("(number of data points * number of features) in cross validation data =", cv_x_o
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 5669
7)
(number of data points * number of features) in test data =  (665, 56697)
(number of data points * number of features) in cross validation data = (5
32, 56697)
```

In [55]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseC
print("(number of data points * number of features) in test data = ", test_x_responseCo
print("(number of data points * number of features) in cross validation data =", cv_x_r
```

```
 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 = (5
32, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

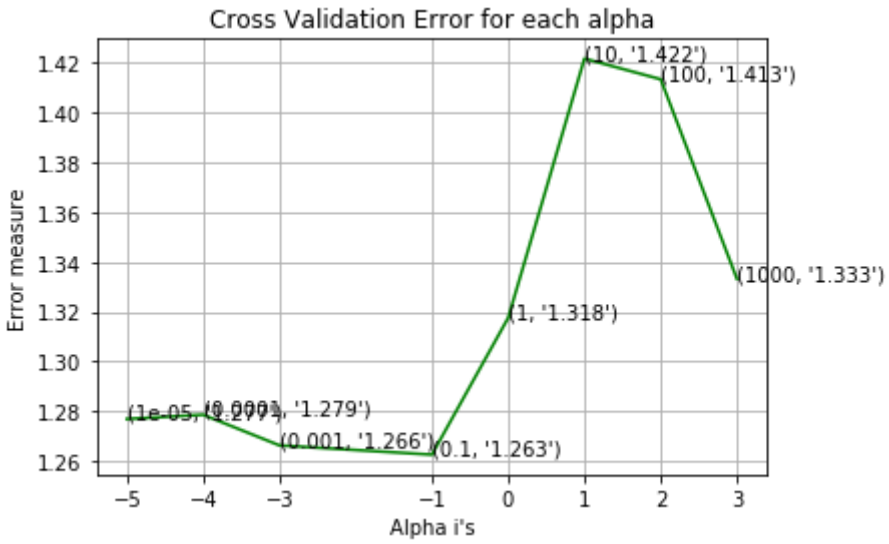
4.1.1.1. Hyper parameter tuning

In [56]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
# -------------------------
# 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/n
# -----------------------


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


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
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-05
Log Loss : 1.2769579890814804
for alpha = 0.0001
Log Loss : 1.2786522379536758
for alpha = 0.001
Log Loss : 1.2663645964444012
for alpha = 0.1
Log Loss : 1.2626410608542329
for alpha = 1
Log Loss : 1.317536172309893
for alpha = 10
Log Loss : 1.4215575848992155
for alpha = 100
Log Loss : 1.4132676880100534
for alpha = 1000
Log Loss : 1.333190021292817
```




```
For values of best alpha =  0.1 The train log loss is: 0.827874338125093
For values of best alpha =  0.1 The cross validation log loss is: 1.262641
0608542329
For values of best alpha =  0.1 The test log loss is: 1.29128364232883
```

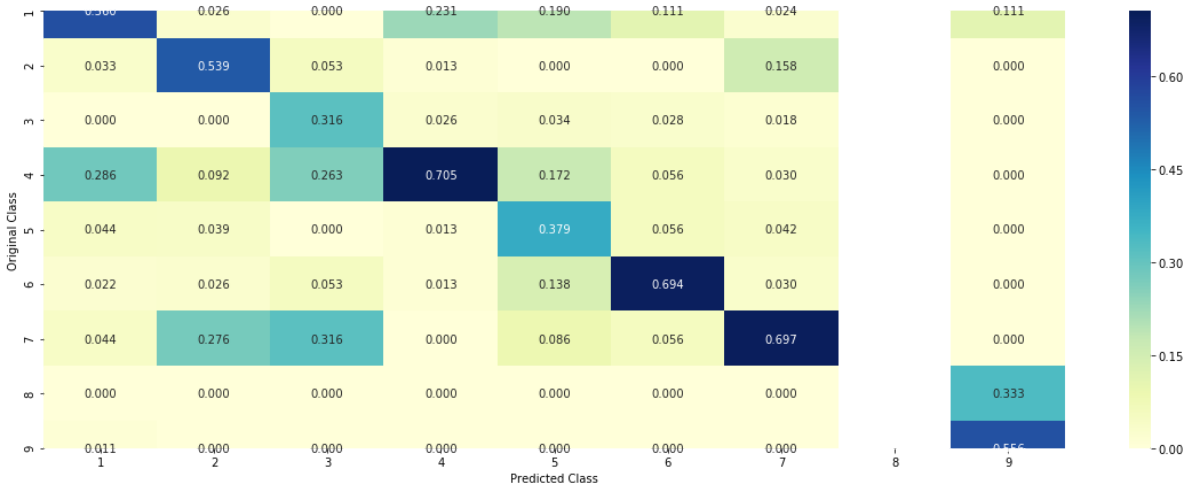
4.1.1.2. Testing the model with best hyper paramters

In [57]:

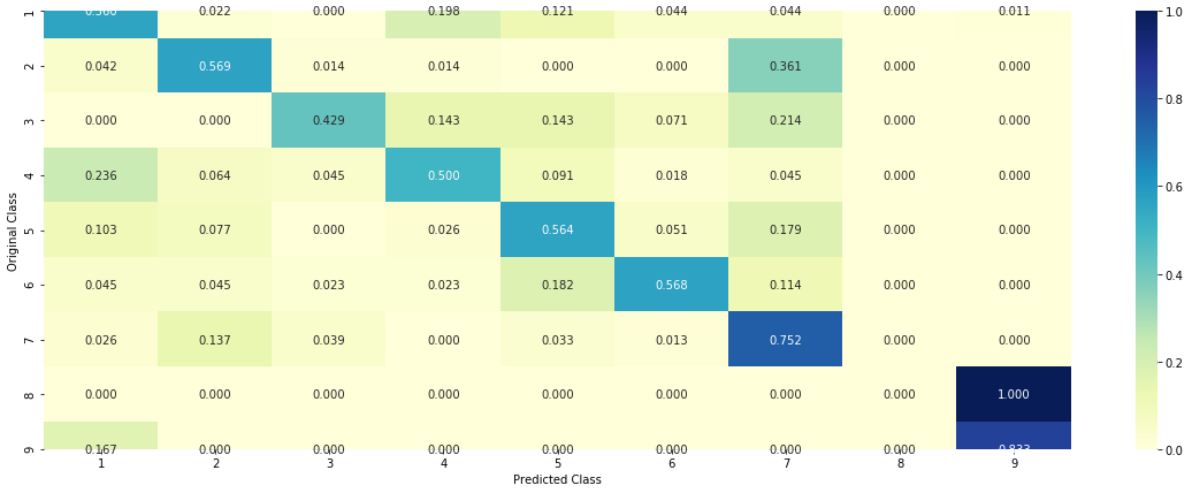
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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/n
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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 estimat
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehot
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.2626410608542329
Number of missclassified point : 0.39849624060150374
-------------------- Confusion matrix --------------------
```

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



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



4.1.1.3. Feature Importance, Correctly classified point

In [58]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.593  0.0707 0.0135 0.1031 0.0362 0.037
0.1377 0.0045 0.0043]]
Actual Class : 7
--------------------------------------------------
15 Text feature [protein] present in test data point [True]
16 Text feature [affect] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [one] present in test data point [True]
19 Text feature [two] present in test data point [True]
20 Text feature [type] present in test data point [True]
21 Text feature [loss] present in test data point [True]
22 Text feature [conserved] present in test data point [True]
23 Text feature [containing] present in test data point [True]
24 Text feature [dna] present in test data point [True]
25 Text feature [binding] present in test data point [True]
26 Text feature [region] present in test data point [True]
28 Text feature [wild] present in test data point [True]
29 Text feature [sequence] present in test data point [True]
30 Text feature [located] present in test data point [True]
31 Text feature [surface] present in test data point [True]
32 Text feature [reduced] present in test data point [True]
33 Text feature [large] present in test data point [True]
34 Text feature [involved] present in test data point [True]
35 Text feature [form] present in test data point [True]
36 Text feature [specifically] present in test data point [True]
37 Text feature [possible] present in test data point [True]
38 Text feature [remaining] present in test data point [True]
39 Text feature [remains] present in test data point [True]
40 Text feature [acids] present in test data point [True]
41 Text feature [terminal] present in test data point [True]
42 Text feature [possibility] present in test data point [True]
43 Text feature [structure] present in test data point [True]
44 Text feature [effect] present in test data point [True]
45 Text feature [contains] present in test data point [True]
46 Text feature [amino] present in test data point [True]
47 Text feature [panel] present in test data point [True]
48 Text feature [sequences] present in test data point [True]
49 Text feature [results] present in test data point [True]
50 Text feature [used] present in test data point [True]
51 Text feature [indicate] present in test data point [True]
52 Text feature [five] present in test data point [True]
53 Text feature [therefore] present in test data point [True]
54 Text feature [data] present in test data point [True]
55 Text feature [important] present in test data point [True]
56 Text feature [indicating] present in test data point [True]
58 Text feature [corresponding] present in test data point [True]
59 Text feature [using] present in test data point [True]
```

```
60 Text feature [specific] present in test data point [True]
61 Text feature [four] present in test data point [True]
62 Text feature [additional] present in test data point [True]
63 Text feature [reveal] present in test data point [True]
64 Text feature [three] present in test data point [True]
65 Text feature [determined] present in test data point [True]
66 Text feature [contain] present in test data point [True]
67 Text feature [also] present in test data point [True]
68 Text feature [likely] present in test data point [True]
69 Text feature [analysis] present in test data point [True]
70 Text feature [2c] present in test data point [True]
71 Text feature [table] present in test data point [True]
72 Text feature [addition] present in test data point [True]
73 Text feature [identified] present in test data point [True]
74 Text feature [present] present in test data point [True]
76 Text feature [result] present in test data point [True]
77 Text feature [identify] present in test data point [True]
78 Text feature [peptide] present in test data point [True]
79 Text feature [discussion] present in test data point [True]
80 Text feature [deletion] present in test data point [True]
81 Text feature [indicated] present in test data point [True]
82 Text feature [specificity] present in test data point [True]
83 Text feature [interactions] present in test data point [True]
84 Text feature [proteins] present in test data point [True]
85 Text feature [well] present in test data point [True]
86 Text feature [structural] present in test data point [True]
87 Text feature [may] present in test data point [True]
88 Text feature [highly] present in test data point [True]
89 Text feature [genetic] present in test data point [True]
90 Text feature [least] present in test data point [True]
91 Text feature [1b] present in test data point [True]
92 Text feature [within] present in test data point [True]
93 Text feature [figure] present in test data point [True]
94 Text feature [gene] present in test data point [True]
95 Text feature [control] present in test data point [True]
96 Text feature [critical] present in test data point [True]
97 Text feature [six] present in test data point [True]
98 Text feature [previous] present in test data point [True]
99 Text feature [respectively] present in test data point [True]
Out of the top  100  features  82 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.1006 0.5384 0.0144 0.1099 0.039  0.0401
0.1482 0.0048 0.0046]]
Actual Class : 7
--------------------------------------------------
17 Text feature [identified] present in test data point [True]
18 Text feature [harbored] present in test data point [True]
19 Text feature [novel] present in test data point [True]
25 Text feature [clinical] present in test data point [True]
26 Text feature [sequencing] present in test data point [True]
27 Text feature [present] present in test data point [True]
28 Text feature [including] present in test data point [True]
29 Text feature [molecular] present in test data point [True]
30 Text feature [another] present in test data point [True]
31 Text feature [using] present in test data point [True]
32 Text feature [mutations] present in test data point [True]
33 Text feature [harbor] present in test data point [True]
34 Text feature [therapeutic] present in test data point [True]
35 Text feature [common] present in test data point [True]
36 Text feature [confirmed] present in test data point [True]
37 Text feature [identify] present in test data point [True]
38 Text feature [new] present in test data point [True]
39 Text feature [identification] present in test data point [True]
40 Text feature [well] present in test data point [True]
41 Text feature [different] present in test data point [True]
42 Text feature [found] present in test data point [True]
43 Text feature [kinase] present in test data point [True]
44 Text feature [patient] present in test data point [True]
45 Text feature [need] present in test data point [True]
46 Text feature [performed] present in test data point [True]
47 Text feature [previously] present in test data point [True]
48 Text feature [potential] present in test data point [True]
49 Text feature [may] present in test data point [True]
50 Text feature [highly] present in test data point [True]
51 Text feature [revealed] present in test data point [True]
52 Text feature [12] present in test data point [True]
53 Text feature [10] present in test data point [True]
54 Text feature [case] present in test data point [True]
55 Text feature [reported] present in test data point [True]
56 Text feature [described] present in test data point [True]
57 Text feature [33] present in test data point [True]
58 Text feature [observed] present in test data point [True]
59 Text feature [one] present in test data point [True]
60 Text feature [15] present in test data point [True]
61 Text feature [respectively] present in test data point [True]
62 Text feature [go] present in test data point [True]
63 Text feature [informed] present in test data point [True]
64 Text feature [studies] present in test data point [True]
```

```
65 Text feature [recently] present in test data point [True]
66 Text feature [findings] present in test data point [True]
67 Text feature [also] present in test data point [True]
68 Text feature [across] present in test data point [True]
69 Text feature [characterized] present in test data point [True]
70 Text feature [gene] present in test data point [True]
71 Text feature [number] present in test data point [True]
72 Text feature [tissue] present in test data point [True]
73 Text feature [small] present in test data point [True]
74 Text feature [detection] present in test data point [True]
75 Text feature [mutational] present in test data point [True]
76 Text feature [mutation] present in test data point [True]
77 Text feature [therapy] present in test data point [True]
78 Text feature [harboring] present in test data point [True]
80 Text feature [similar] present in test data point [True]
81 Text feature [per] present in test data point [True]
82 Text feature [distinct] present in test data point [True]
83 Text feature [analysis] present in test data point [True]
84 Text feature [additional] present in test data point [True]
85 Text feature [specific] present in test data point [True]
86 Text feature [40] present in test data point [True]
87 Text feature [samples] present in test data point [True]
88 Text feature [however] present in test data point [True]
89 Text feature [care] present in test data point [True]
90 Text feature [first] present in test data point [True]
91 Text feature [approved] present in test data point [True]
92 Text feature [cases] present in test data point [True]
93 Text feature [table] present in test data point [True]
94 Text feature [respond] present in test data point [True]
95 Text feature [inhibitor] present in test data point [True]
96 Text feature [study] present in test data point [True]
97 Text feature [non] present in test data point [True]
98 Text feature [subsequently] present in test data point [True]
Out of the top  100  features  76 are present in query point
```

4.2. K Nearest Neighbour Classification

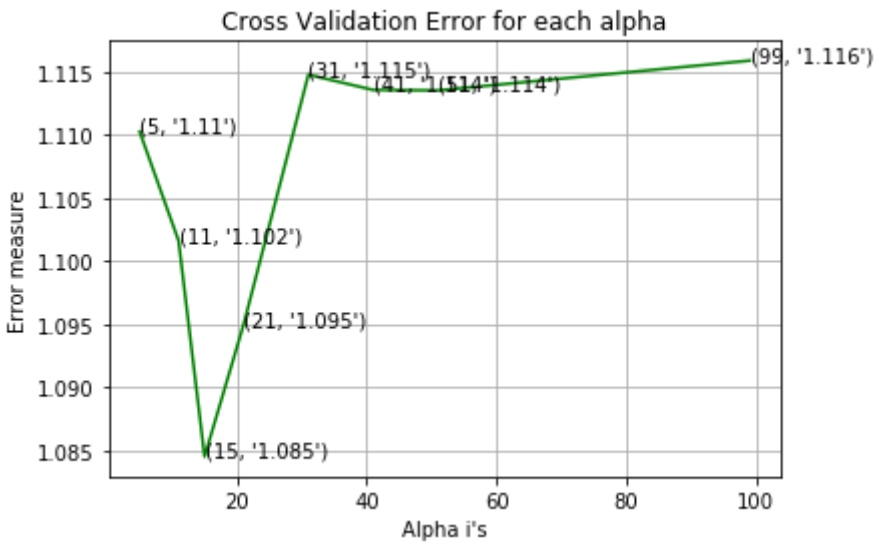
4.2.1. Hyper parameter tuning

In [60]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
# 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
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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
    # to avoid rounding error while multiplying probabilites we use log-probability est
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 1.110251149556719
for alpha = 11
Log Loss : 1.1016129555035976
for alpha = 15
Log Loss : 1.0845401916340172
for alpha = 21
Log Loss : 1.0949624971411414
for alpha = 31
Log Loss : 1.1147599671519892
for alpha = 41
Log Loss : 1.1135502765295497
for alpha = 51
Log Loss : 1.1135142881564921
for alpha = 99
Log Loss : 1.11587262838981
```




```
For values of best alpha =  15 The train log loss is: 0.6558752411673646
For values of best alpha =  15 The cross validation log loss is: 1.0845401
916340172
For values of best alpha =  15 The test log loss is: 1.0871043190180913
```

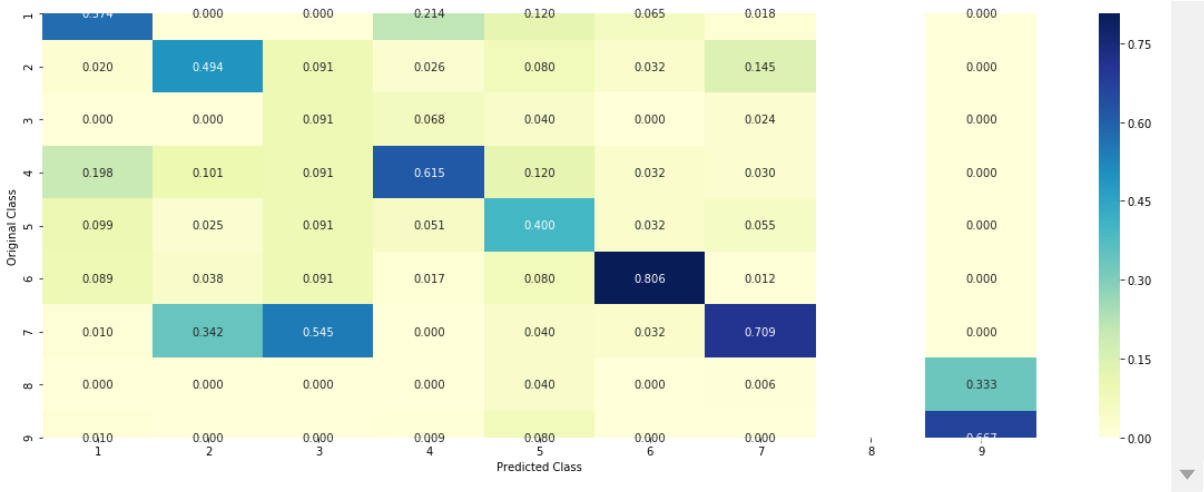
4.2.2. Testing the model with best hyper paramters

In [61]:

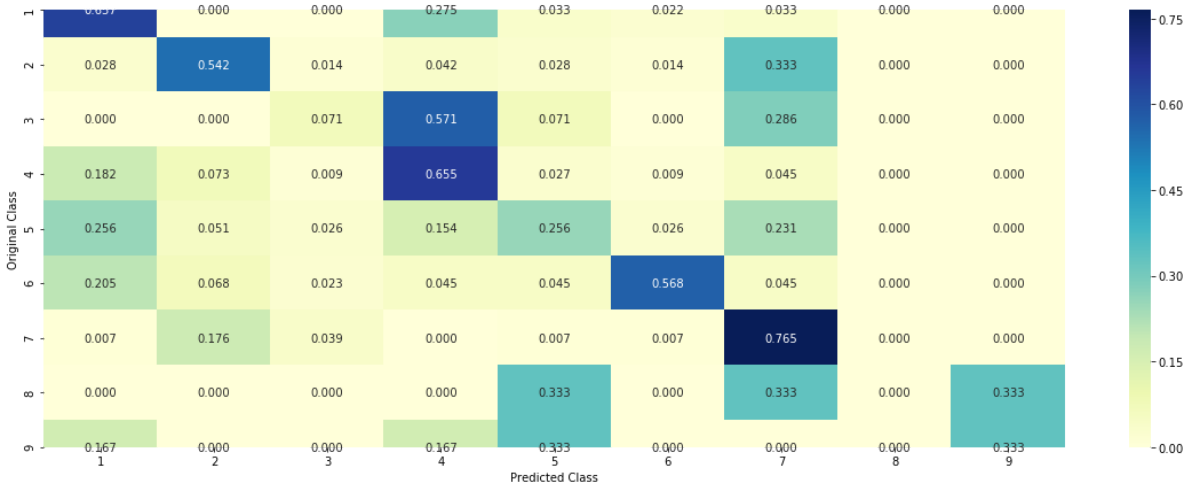
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# 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
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding,
```

```
Log loss : 1.0845401916340172
Number of mis-classified points : 0.39097744360902253
-------------------- Confusion matrix --------------------
```


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



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



4.2.3.Sample Query point -1

In [62]:

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

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

```
Predicted Class : 1
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [1 1 1 1
1 1 1 6 1 1 1 1 1 5 1]
Fequency of nearest points : Counter({1: 13, 6: 1, 5: 1})
```

4.2.4. Sample Query Point-2

In [63]:

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

test_point_index = 100

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points be
longs to classes [7 7 2 7 7 7 7 7 7 7 7 5 7 2 7]
Fequency of nearest points : Counter({7: 12, 2: 2, 5: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

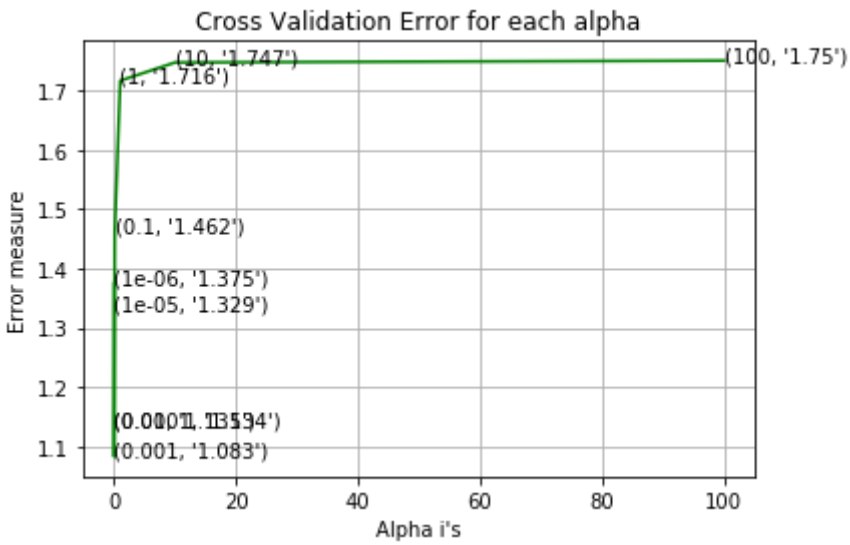
4.3.1.1. Hyper paramter tuning

In [64]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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', ran
    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
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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', los
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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.3746369495243174
for alpha = 1e-05
Log Loss : 1.3287051661171339
for alpha = 0.0001
Log Loss : 1.1343192616962556
for alpha = 0.001
Log Loss : 1.0827876725312422
for alpha = 0.01
Log Loss : 1.1346769715504261
for alpha = 0.1
Log Loss : 1.4616447001405646
for alpha = 1
Log Loss : 1.7163022407601805
for alpha = 10
Log Loss : 1.747018506674435
for alpha = 100
Log Loss : 1.7501483913560347
```




```
For values of best alpha =  0.001 The train log loss is: 0.516388827857123
8
For values of best alpha =  0.001 The cross validation log loss is: 1.0827
876725312422
For values of best alpha =  0.001 The test log loss is: 1.1159730441707065
```

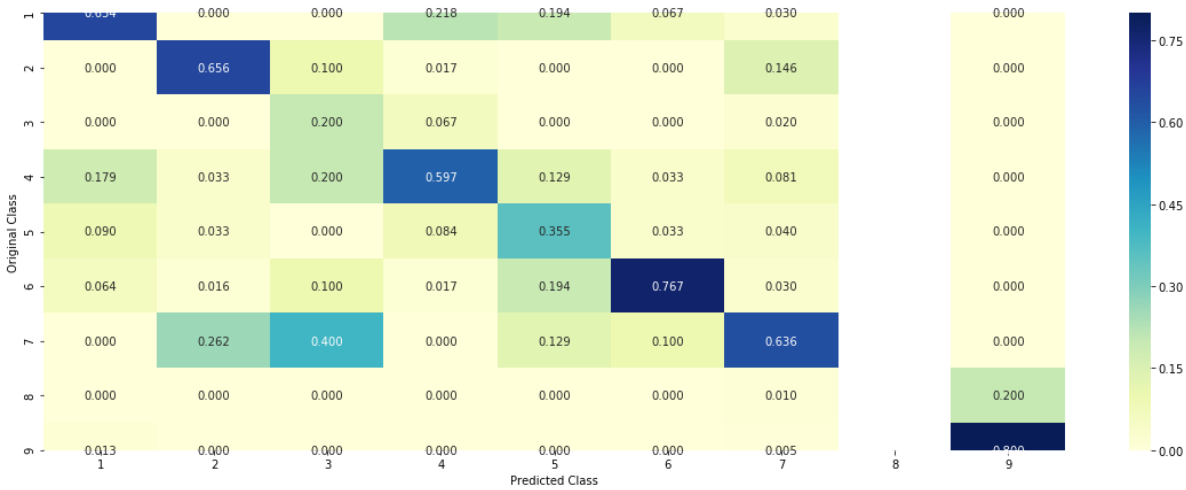
4.3.1.2. Testing the model with best hyper paramters

In [65]:

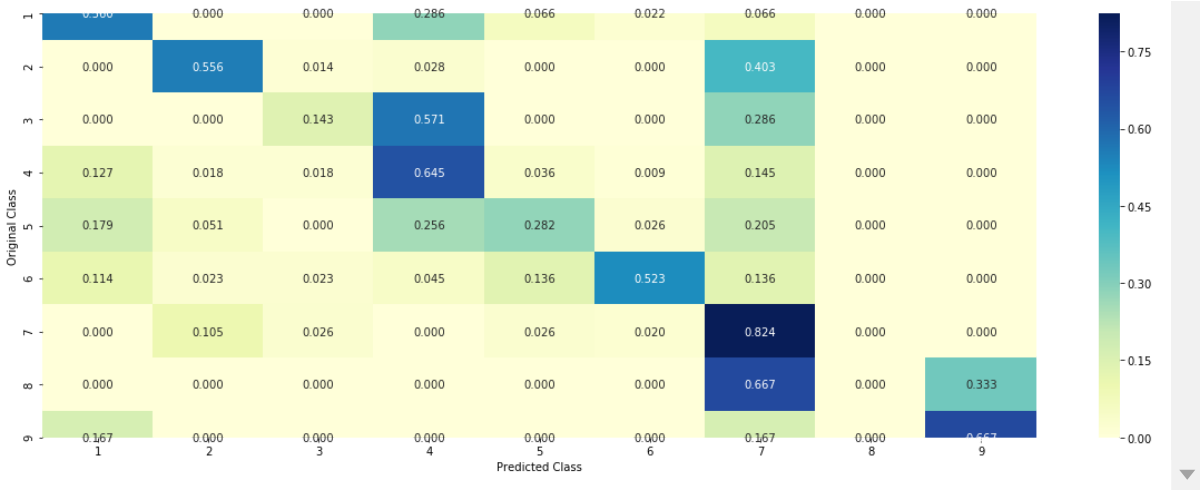
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/g
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_
```

```
Log loss : 1.0827876725312422
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```


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




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



4.3.1.3. Feature Importance

In [66]:

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

4.3.1.3.1. Correctly Classified point

In [67]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.959e-01 1.200e-03 3.000e-04 2.000e-04
3.000e-04 1.000e-04 8.000e-04
  1.000e-03 2.000e-04]]
Actual Class : 7
--------------------------------------------------
19 Text feature [appears] present in test data point [True]
70 Text feature [selenium] present in test data point [True]
74 Text feature [serous] present in test data point [True]
115 Text feature [wall] present in test data point [True]
119 Text feature [sections] present in test data point [True]
153 Text feature [terminal] present in test data point [True]
181 Text feature [g12r] present in test data point [True]
293 Text feature [source] present in test data point [True]
295 Text feature [variants] present in test data point [True]
329 Text feature [atomic] present in test data point [True]
345 Text feature [interactions] present in test data point [True]
357 Text feature [beamline] present in test data point [True]
392 Text feature [2004] present in test data point [True]
404 Text feature [fifth] present in test data point [True]
423 Text feature [deviations] present in test data point [True]
464 Text feature [loading] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [68]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0053 0.2884 0.0014 0.0038 0.0015 0.0012
0.6948 0.002  0.0016]]
Actual Class : 7
--------------------------------------------------
14 Text feature [assessed] present in test data point [True]
36 Text feature [clinical] present in test data point [True]
49 Text feature [versions] present in test data point [True]
50 Text feature [detection] present in test data point [True]
57 Text feature [screened] present in test data point [True]
102 Text feature [tmprss2] present in test data point [True]
110 Text feature [20] present in test data point [True]
123 Text feature [true] present in test data point [True]
154 Text feature [left] present in test data point [True]
161 Text feature [2005] present in test data point [True]
184 Text feature [compared] present in test data point [True]
208 Text feature [next] present in test data point [True]
216 Text feature [ampure] present in test data point [True]
249 Text feature [longitudinal] present in test data point [True]
295 Text feature [micrometer] present in test data point [True]
305 Text feature [rounding] present in test data point [True]
325 Text feature [however] present in test data point [True]
410 Text feature [myc] present in test data point [True]
Out of the top  500  features  18 are present in query point
```

4.3.2. Without Class balancing

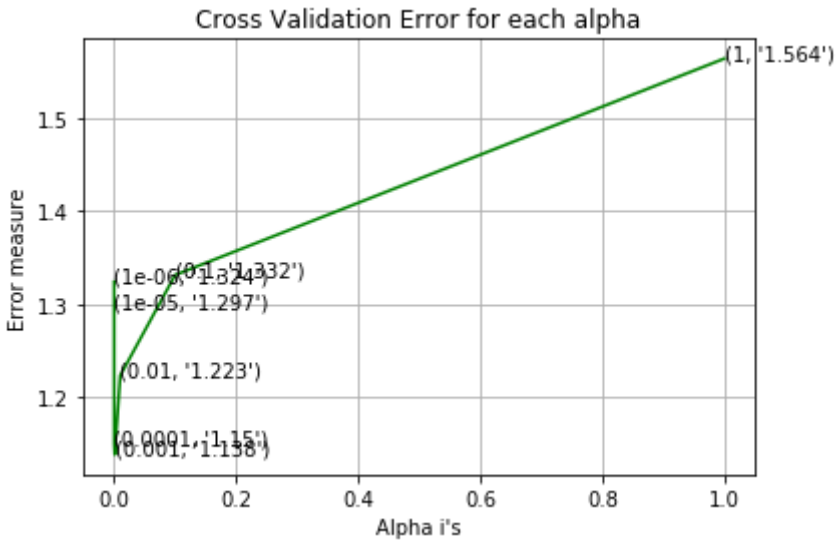
4.3.2.1. Hyper paramter tuning

In [69]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# -----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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
    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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.324333684866676
for alpha = 1e-05
Log Loss : 1.2970095105374884
for alpha = 0.0001
Log Loss : 1.1499285968783841
for alpha = 0.001
Log Loss : 1.138428166679973
for alpha = 0.01
Log Loss : 1.2234242955921995
for alpha = 0.1
Log Loss : 1.3315701061503395
for alpha = 1
Log Loss : 1.5641875108214855
```




```
For values of best alpha =  0.001 The train log loss is: 0.513566237455390
5
For values of best alpha =  0.001 The cross validation log loss is: 1.1384
28166679973
For values of best alpha =  0.001 The test log loss is: 1.1224683423782458
```

4.3.2.2. Testing model with best hyper parameters

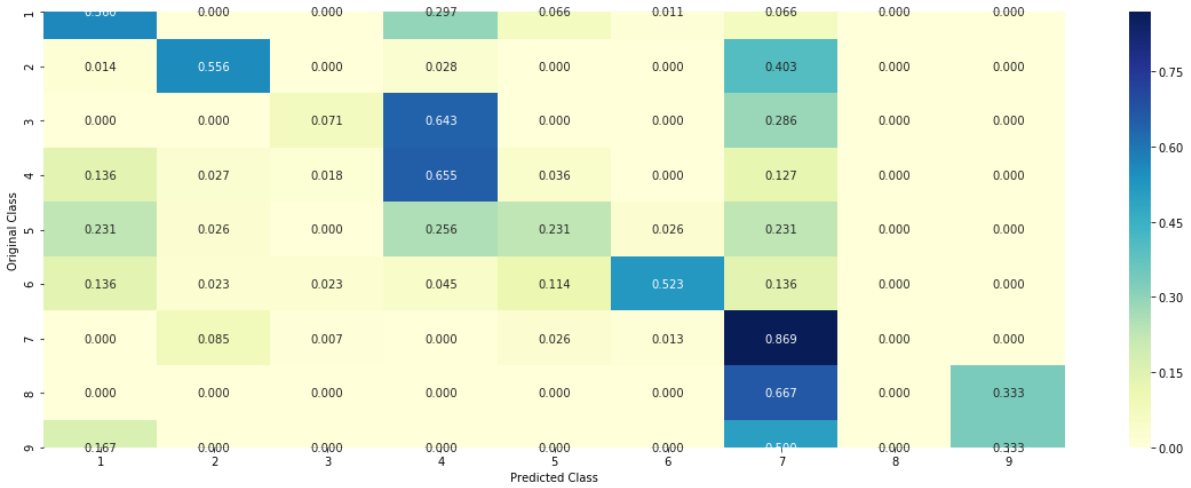
In [70]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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_
```

```
Log loss : 1.138428166679973
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```


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

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



4.3.2.3. Feature Importance, Correctly Classified point

In [71]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[9.956e-01 1.300e-03 1.000e-04 3.000e-04
2.000e-04 1.000e-04 1.600e-03
  6.000e-04 0.000e+00]]
Actual Class : 7
--------------------------------------------------
28 Text feature [framework] present in test data point [True]
39 Text feature [84] present in test data point [True]
69 Text feature [provides] present in test data point [True]
95 Text feature [allowing] present in test data point [True]
134 Text feature [g12r] present in test data point [True]
161 Text feature [following] present in test data point [True]
183 Text feature [apoptosis] present in test data point [True]
299 Text feature [sections] present in test data point [True]
396 Text feature [2004] present in test data point [True]
413 Text feature [difficult] present in test data point [True]
417 Text feature [upregulated] present in test data point [True]
420 Text feature [thereby] present in test data point [True]
427 Text feature [418] present in test data point [True]
429 Text feature [atomic] present in test data point [True]
486 Text feature [43] present in test data point [True]
491 Text feature [structurally] present in test data point [True]
497 Text feature [setup] present in test data point [True]
Out of the top  500  features  17 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [72]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[5.200e-03 2.526e-01 8.000e-04 4.000e-03
1.300e-03 9.000e-04 7.337e-01
  1.400e-03 1.000e-04]]
Actual Class : 7
--------------------------------------------------
25 Text feature [implicated] present in test data point [True]
39 Text feature [package] present in test data point [True]
60 Text feature [beyond] present in test data point [True]
66 Text feature [end] present in test data point [True]
71 Text feature [revealed] present in test data point [True]
80 Text feature [matched] present in test data point [True]
87 Text feature [promoter] present in test data point [True]
88 Text feature [animals] present in test data point [True]
102 Text feature [hd] present in test data point [True]
106 Text feature [calipers] present in test data point [True]
127 Text feature [show] present in test data point [True]
133 Text feature [ii] present in test data point [True]
147 Text feature [urothelial] present in test data point [True]
```

4.4. Linear Support Vector Machines

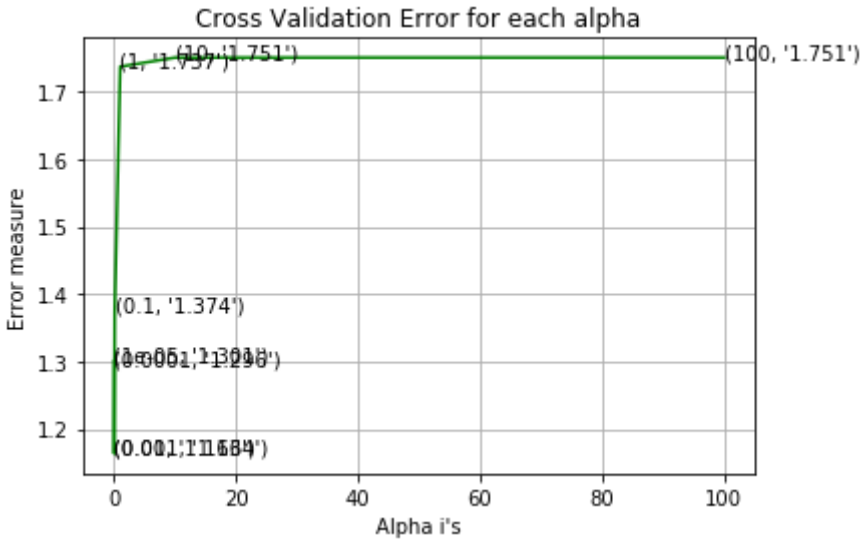
4.4.1. Hyper paramter tuning

In [73]:

```
# read more about support vector machines with linear kernals here http://scikit-learn.

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

# 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/m
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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',
    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
    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', los
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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for C = 1e-05
Log Loss : 1.3008677134873228
for C = 0.0001
Log Loss : 1.2961781706908548
for C = 0.001
Log Loss : 1.1642217669961243
for C = 0.01
Log Loss : 1.163323231344656
for C = 0.1
Log Loss : 1.3744092553390994
for C = 1
Log Loss : 1.7371816191531229
for C = 10
Log Loss : 1.7506130877030062
for C = 100
Log Loss : 1.7506146600525092
```




```
For values of best alpha =  0.01 The train log loss is: 0.7142784280480103
For values of best alpha =  0.01 The cross validation log loss is: 1.16332
3231344656
For values of best alpha =  0.01 The test log loss is: 1.1815970840255945
```

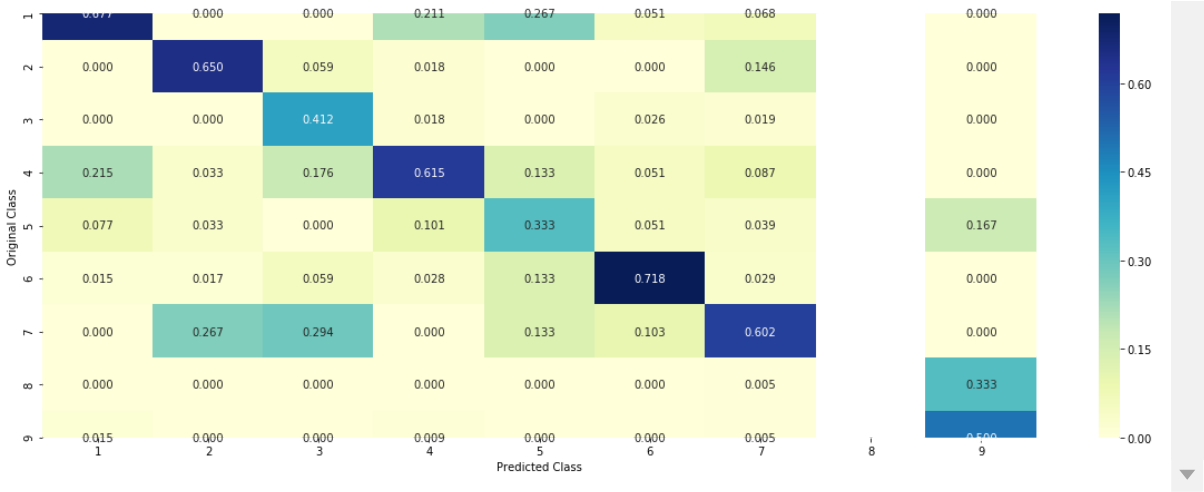
4.4.2. Testing model with best hyper parameters

In [74]:

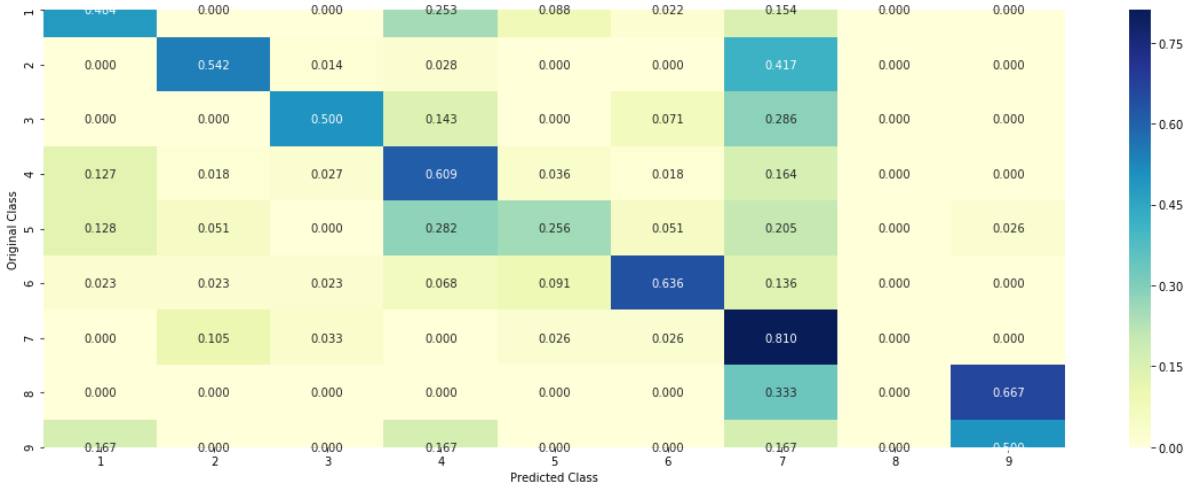
```
# read more about support vector machines with linear kernals here http://scikit-learn.

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

# 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/m
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balance
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y,
```

```
Log loss : 1.163323231344656
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```


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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [75]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[9.02e-01 1.88e-02 2.60e-03 1.40e-02 8.70e
-03 5.90e-03 4.68e-02 5.00e-04
  7.00e-04]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [76]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0223 0.4605 0.0041 0.0227 0.0119 0.0083
0.4678 0.0009 0.0013]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [77]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None
# 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/r
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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, ran
        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_, ep
        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_err
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', ma
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 i
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validatio
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss i
```

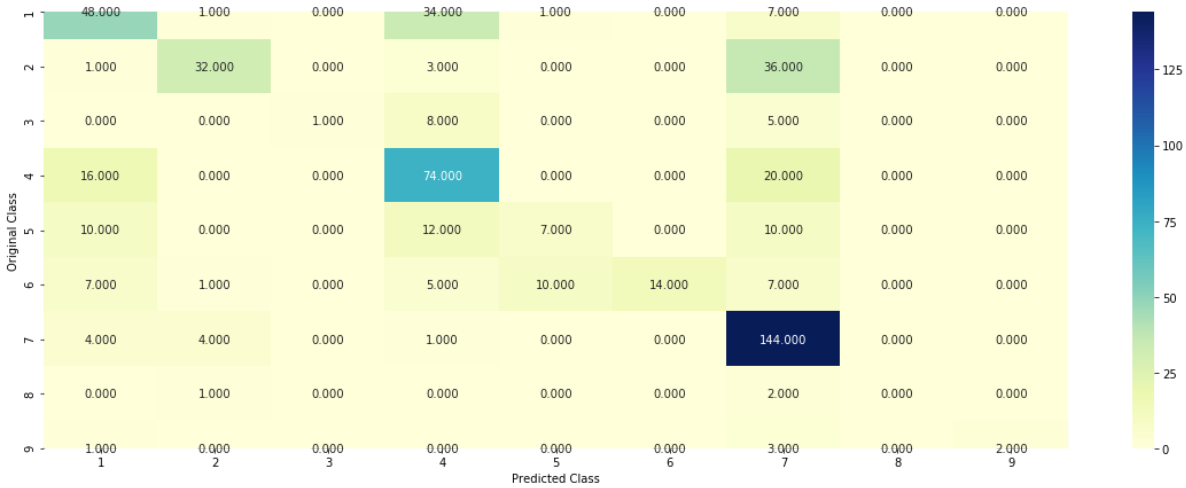
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2605003210365398
for n_estimators = 100 and max depth =  10
Log Loss : 1.193816234070744
for n_estimators = 200 and max depth =  5
Log Loss : 1.2517056807724507
for n_estimators = 200 and max depth =  10
Log Loss : 1.1939432285103007
for n_estimators = 500 and max depth =  5
Log Loss : 1.2430580662035637
for n_estimators = 500 and max depth =  10
Log Loss : 1.184416359430552
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2383697991201568
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1846564690258288
for n_estimators = 2000 and max depth =  5
Log Loss : 1.237935276341979
for n_estimators = 2000 and max depth =  10
Log Loss : 1.184041550473973
For values of best estimator =  2000 The train log loss is: 0.674080610586
456
For values of best estimator =  2000 The cross validation log loss is: 1.1
84041550473973
For values of best estimator =  2000 The test log loss is: 1.2231922697792
608
```

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

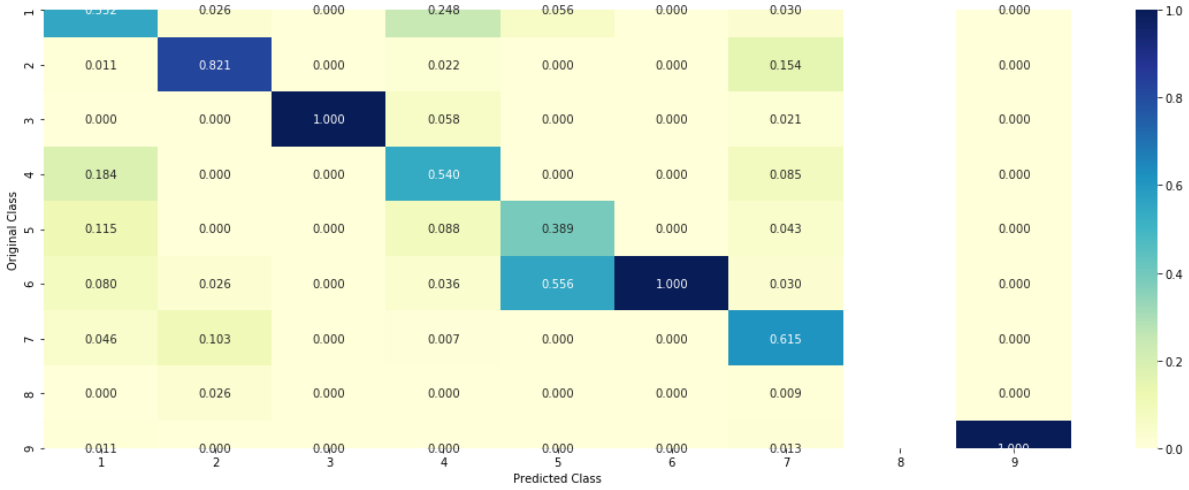
In [78]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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/r
# -------------------------------

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

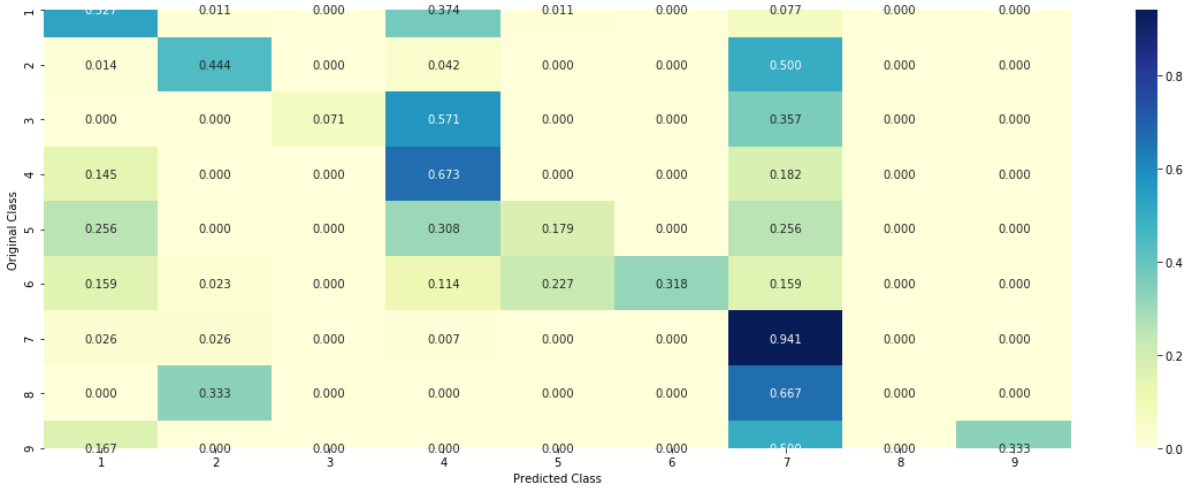
```
Log loss : 1.184041550473973
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```




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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [79]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', m
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_onehotCod
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_
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6956 0.0432 0.0148 0.0817 0.0396 0.03
41 0.0794 0.0048 0.0069]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitor] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [treatment] present in test data point [True]
7 Text feature [activated] present in test data point [True]
10 Text feature [missense] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [inhibition] present in test data point [True]
14 Text feature [signaling] present in test data point [True]
15 Text feature [erk] present in test data point [True]
16 Text feature [oncogenic] present in test data point [True]
20 Text feature [kinases] present in test data point [True]
22 Text feature [functional] present in test data point [True]
23 Text feature [growth] present in test data point [True]
24 Text feature [akt] present in test data point [True]
26 Text feature [cells] present in test data point [True]
29 Text feature [loss] present in test data point [True]
34 Text feature [trials] present in test data point [True]
35 Text feature [downstream] present in test data point [True]
38 Text feature [expressing] present in test data point [True]
39 Text feature [unstable] present in test data point [True]
41 Text feature [drug] present in test data point [True]
42 Text feature [resistance] present in test data point [True]
53 Text feature [variants] present in test data point [True]
54 Text feature [patients] present in test data point [True]
56 Text feature [expression] present in test data point [True]
61 Text feature [factor] present in test data point [True]
63 Text feature [proliferation] present in test data point [True]
64 Text feature [variant] present in test data point [True]
66 Text feature [mapk] present in test data point [True]
67 Text feature [lines] present in test data point [True]
69 Text feature [ras] present in test data point [True]
70 Text feature [protein] present in test data point [True]
71 Text feature [kit] present in test data point [True]
```

```
74 Text feature [survival] present in test data point [True]
76 Text feature [ligand] present in test data point [True]
77 Text feature [activate] present in test data point [True]
81 Text feature [assays] present in test data point [True]
86 Text feature [ovarian] present in test data point [True]
89 Text feature [cell] present in test data point [True]
97 Text feature [clinical] present in test data point [True]
Out of the top  100  features  44 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [80]:

```
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_onehotCod
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_
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0344 0.4487 0.0125 0.0239 0.03   0.02
62 0.4147 0.0043 0.0052]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitor] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [treatment] present in test data point [True]
7 Text feature [activated] present in test data point [True]
8 Text feature [tyrosine] present in test data point [True]
9 Text feature [constitutive] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [inhibition] present in test data point [True]
14 Text feature [signaling] present in test data point [True]
16 Text feature [oncogenic] present in test data point [True]
18 Text feature [receptor] present in test data point [True]
20 Text feature [kinases] present in test data point [True]
22 Text feature [functional] present in test data point [True]
23 Text feature [growth] present in test data point [True]
24 Text feature [akt] present in test data point [True]
25 Text feature [therapeutic] present in test data point [True]
26 Text feature [cells] present in test data point [True]
29 Text feature [loss] present in test data point [True]
32 Text feature [therapy] present in test data point [True]
34 Text feature [trials] present in test data point [True]
35 Text feature [downstream] present in test data point [True]
37 Text feature [phospho] present in test data point [True]
38 Text feature [expressing] present in test data point [True]
40 Text feature [efficacy] present in test data point [True]
41 Text feature [drug] present in test data point [True]
43 Text feature [imatinib] present in test data point [True]
44 Text feature [respond] present in test data point [True]
46 Text feature [pathogenic] present in test data point [True]
47 Text feature [transforming] present in test data point [True]
49 Text feature [treated] present in test data point [True]
50 Text feature [erk1] present in test data point [True]
52 Text feature [truncating] present in test data point [True]
53 Text feature [variants] present in test data point [True]
54 Text feature [patients] present in test data point [True]
55 Text feature [inhibited] present in test data point [True]
56 Text feature [expression] present in test data point [True]
57 Text feature [months] present in test data point [True]
59 Text feature [starved] present in test data point [True]
```

```
61 Text feature [factor] present in test data point [True]
63 Text feature [proliferation] present in test data point [True]
64 Text feature [variant] present in test data point [True]
66 Text feature [mapk] present in test data point [True]
67 Text feature [lines] present in test data point [True]
68 Text feature [inositol] present in test data point [True]
69 Text feature [ras] present in test data point [True]
70 Text feature [protein] present in test data point [True]
71 Text feature [kit] present in test data point [True]
72 Text feature [sensitivity] present in test data point [True]
74 Text feature [survival] present in test data point [True]
77 Text feature [activate] present in test data point [True]
79 Text feature [tkis] present in test data point [True]
81 Text feature [assays] present in test data point [True]
86 Text feature [ovarian] present in test data point [True]
89 Text feature [cell] present in test data point [True]
90 Text feature [oncogene] present in test data point [True]
95 Text feature [patient] present in test data point [True]
97 Text feature [clinical] present in test data point [True]
Out of the top  100  features  62 are present in query point
```

4.5.3. Random forest -Hyper paramter tuning (With Response Coding)

In [81]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None
# 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/r
# --------------------------------


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, ran
        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_, ep
        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_err
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', ma
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.126266522848456
for n_estimators = 10 and max depth =  3
Log Loss : 1.591456175412291
for n_estimators = 10 and max depth =  5
Log Loss : 1.5288408514351315
for n_estimators = 10 and max depth =  10
Log Loss : 1.6901381321014912
for n_estimators = 50 and max depth =  2
Log Loss : 1.6679325564535277
for n_estimators = 50 and max depth =  3
Log Loss : 1.419401830449453
for n_estimators = 50 and max depth =  5
Log Loss : 1.3433852366657955
for n_estimators = 50 and max depth =  10
Log Loss : 1.6895423245351904
for n_estimators = 100 and max depth =  2
Log Loss : 1.5194208997855525
for n_estimators = 100 and max depth =  3
Log Loss : 1.3878777084432041
for n_estimators = 100 and max depth =  5
Log Loss : 1.3279613923463618
for n_estimators = 100 and max depth =  10
Log Loss : 1.6790894772412293
for n_estimators = 200 and max depth =  2
Log Loss : 1.5547455724305201
for n_estimators = 200 and max depth =  3
Log Loss : 1.4092047338755838
for n_estimators = 200 and max depth =  5
Log Loss : 1.3853506485049492
for n_estimators = 200 and max depth =  10
Log Loss : 1.650164817204003
for n_estimators = 500 and max depth =  2
Log Loss : 1.5791174072125347
for n_estimators = 500 and max depth =  3
Log Loss : 1.46856404233198
for n_estimators = 500 and max depth =  5
Log Loss : 1.4008060820968653
for n_estimators = 500 and max depth =  10
Log Loss : 1.6765087056456853
for n_estimators = 1000 and max depth =  2
Log Loss : 1.587759530803378
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4811010761850463
for n_estimators = 1000 and max depth =  5
```

```
Log Loss : 1.390089421363068
for n_estimators = 1000 and max depth =  10
Log Loss : 1.663890353229022
For values of best alpha =  100 The train log loss is: 0.07037105400327831
For values of best alpha =  100 The cross validation log loss is: 1.327961
392346362
For values of best alpha =  100 The test log loss is: 1.3090433067186926
```

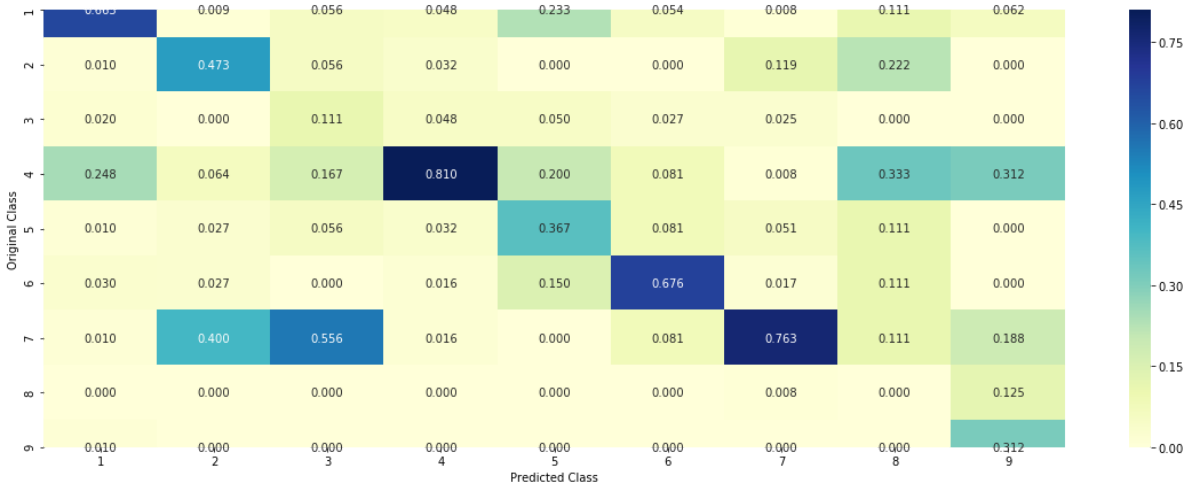
4.5.4. Testing model with best hyper parameters (Response Coding)

In [82]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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/r
# -------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,c
```

```
Log loss : 1.327961392346362
Number of mis-classified points : 0.40977443609022557
-------------------- 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 [83]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3299 0.0453 0.072  0.0476 0.0377 0.044
0.0131 0.2336 0.1769]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [84]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseC
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 : 2
Predicted Class Probabilities: [[0.0275 0.546  0.0901 0.0205 0.0343 0.0574
0.1751 0.0333 0.0159]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene 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 [85]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# read more about support vector machines with linear kernals here http://scikit-learn.
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

# 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/m
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit-learn.
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None
# 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/r
# --------------------------------


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', rand
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
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pr
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_one
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_classifi
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_l
    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: 1.08
Support vector machines : Log Loss: 1.74
Naive Bayes : Log Loss: 1.27
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.819
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.727
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.349
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.198
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.467
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.810
```

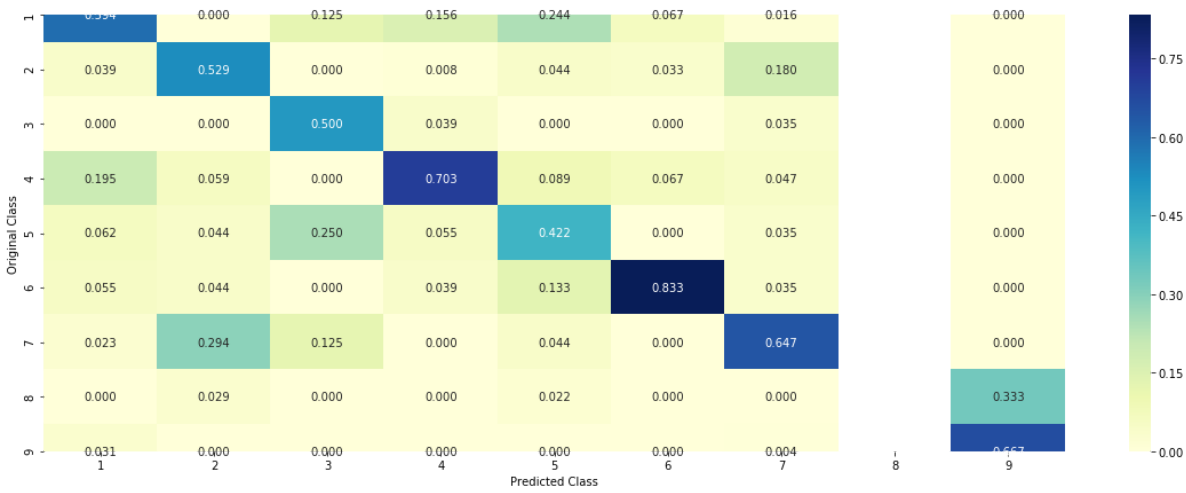
4.7.2 testing the model with the best hyper parameters

In [86]:

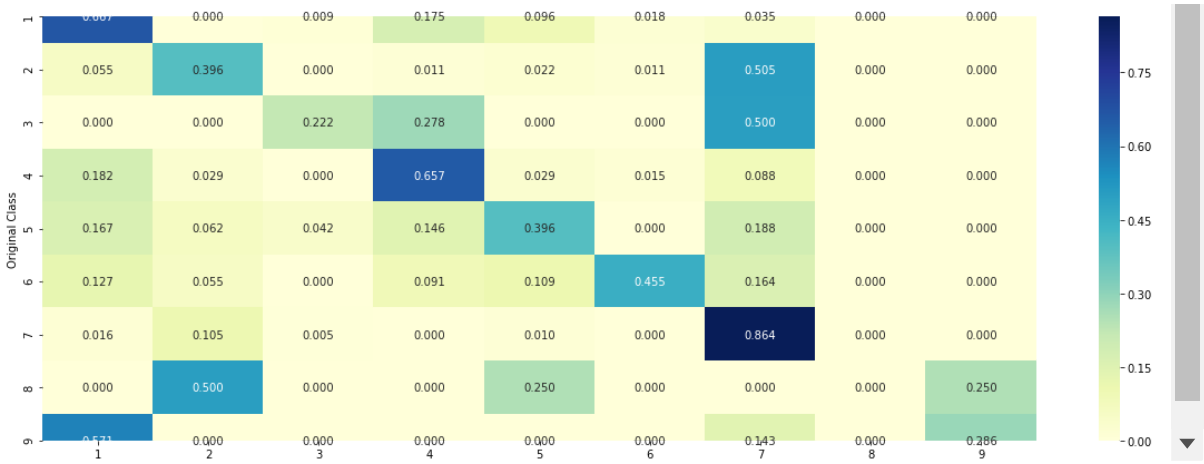
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
sclf.fit(train_x_onehotCoding, train_y)

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_onehotC
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.48458378186459394
Log loss (CV) on the stacking classifier : 1.1982116126454456
Log loss (test) on the stacking classifier : 1.191234769338613
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```


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




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



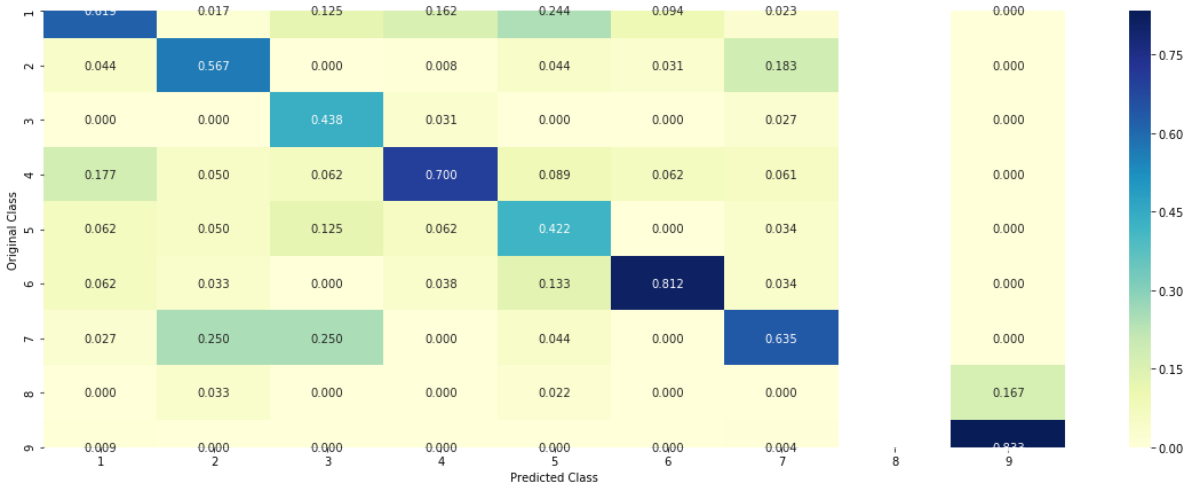
4.7.3 Maximum Voting classifier

In [87]:

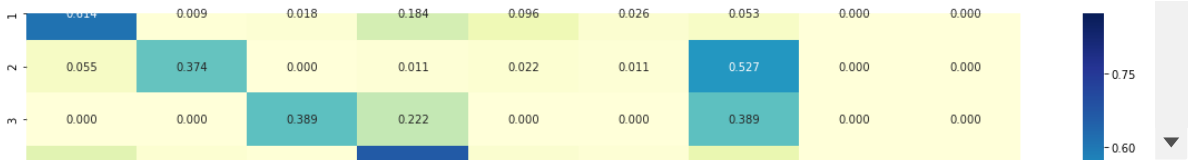
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifi
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier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

```
Log loss (train) on the VotingClassifier : 0.8415685282898164
Log loss (CV) on the VotingClassifier : 1.1936476052130194
Log loss (test) on the VotingClassifier : 1.224374399415632
Number of missclassified point : 0.3699248120300752
-------------------- Confusion matrix --------------------
```


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




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



5. 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

A1. Applying all the models with TFIDF-Features

A2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values

1.1 univariate analysis - gene Feature

Q1. 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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

1. One hot Encoding (tfidfvectorizer)
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 [88]:

```
#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 [89]:

```
# building a tfidfvectorizer with all the words that occured minimum 10 times in train
gene_tfidf_vectorizer = TfidfVectorizer()

train_gene_feature_tfidf_Coding = gene_tfidf_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tfidf_Coding = gene_tfidf_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tfidf_Coding = gene_tfidf_vectorizer.transform(cv_df['Gene'])

print(train_gene_feature_tfidf_Coding.shape)
print(test_gene_feature_tfidf_Coding.shape)
print(cv_gene_feature_tfidf_Coding.shape)
```

```
(2124, 229)
(665, 229)
(532, 229)
```

In [101]:

```
gene_tfidf_vectorizer.get_feature_names()[:10]
```

Out[101]:

```
['abl1', 'acvr1', 'ago2', 'akt1', 'akt2', 'akt3', 'alk', 'apc', 'ar', 'ara
f']
```

How good is this gene feature in predicting y_i?

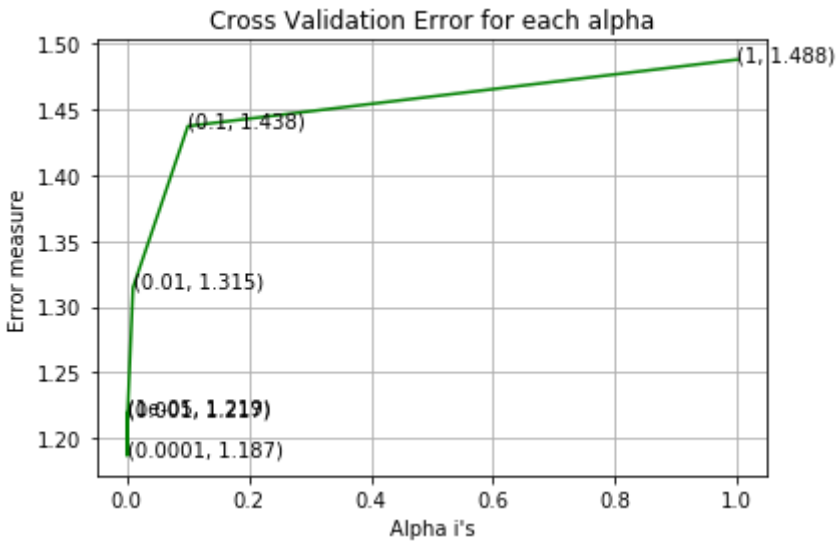
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (tfidf one hot encoded) to predict y_i.

In [103]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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_tfidf_Coding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_tfidf_Coding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_tfidf_Coding)
    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, lab

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_tfidf_Coding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_tfidf_Coding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_gene_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_gene_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
For values of alpha =  1e-05 The log loss is: 1.2191723935068697
For values of alpha =  0.0001 The log loss is: 1.1866621639086434
For values of alpha =  0.001 The log loss is: 1.216819587421816
```

```
For values of alpha =  0.01 The log loss is: 1.3146104694275906
For values of alpha =  0.1 The log loss is: 1.437520887294074
For values of alpha =  1 The log loss is: 1.4881135574667714
```




```
For values of best alpha =  0.0001 The train log loss is: 0.97606389518064
78
For values of best alpha =  0.0001 The cross validation log loss is: 1.186
6621639086434
For values of best alpha =  0.0001 The test log loss is: 1.230763328806195
2
```

1.2 Univarative Analysis - Variation

In [98]:

```
# building a tfidfvectorizer with all the words that occured minimum 10 times in train
variation_tfidf_vectorizer = TfidfVectorizer()

train_variation_feature_tfidf_Coding = variation_tfidf_vectorizer.fit_transform(train_d
test_variation_feature_tfidf_Coding = variation_tfidf_vectorizer.transform(test_df['Var
cv_variation_feature_tfidf_Coding = variation_tfidf_vectorizer.transform(cv_df['Variati

print(train_variation_feature_tfidf_Coding.shape)
print(test_variation_feature_tfidf_Coding.shape)
print(cv_variation_feature_tfidf_Coding.shape)
```

```
(2124, 1965)
(665, 1965)
(532, 1965)
```

In [99]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

chi = SelectKBest(chi2,k=1000).fit(train_variation_feature_tfidf_Coding, y_train)

train_variation_feature_tfidf_Coding = chi.transform(train_variation_feature_tfidf_Codi
test_variation_feature_tfidf_Coding  = chi.transform(test_variation_feature_tfidf_Codin
cv_variation_feature_tfidf_Coding    = chi.transform(cv_variation_feature_tfidf_Coding)

print(train_variation_feature_tfidf_Coding.shape)
print(test_variation_feature_tfidf_Coding.shape)
print(cv_variation_feature_tfidf_Coding.shape)
```

```
(2124, 1000)
(665, 1000)
(532, 1000)
```

How good is this Variation feature in predicting y_i?

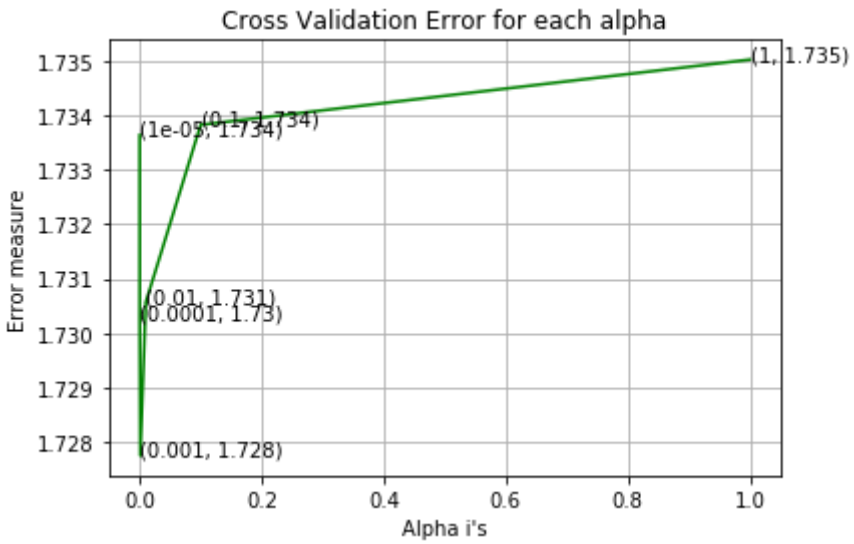
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, **we will build a logistic regression model using only Variation feature (tfidf one hot encoded) to predict y_i.**

In [104]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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_tfidf_Coding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_tfidf_Coding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf_Coding)
    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, lab

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_tfidf_Coding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_tfidf_Coding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_variation_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
For values of alpha =  1e-05 The log loss is: 1.7336350764347808
For values of alpha =  0.0001 The log loss is: 1.7302675915271597
```

```
For values of alpha =  0.001 The log loss is: 1.7277445915555776
For values of alpha =  0.01 The log loss is: 1.730563590443488
For values of alpha =  0.1 The log loss is: 1.7338244968885133
For values of alpha =  1 The log loss is: 1.7350235398434422
```




```
For values of best alpha =  0.001 The train log loss is: 1.309718866256018
For values of best alpha =  0.001 The cross validation log loss is: 1.7277
445915555776
For values of best alpha =  0.001 The test log loss is: 1.701941130456166
```

1.3 Univarative Analysis - text

In [92]:

```
# building a TfidfVectorizer with all the words that occured minimum 10 times in train
text_tfidf_vectorizer = TfidfVectorizer(lowercase=False,min_df=10)

train_text_feature_tfidf_Coding = text_tfidf_vectorizer.fit_transform(train_df['TEXT'])
test_text_feature_tfidf_Coding = text_tfidf_vectorizer.transform(test_df['TEXT'])
cv_text_feature_tfidf_Coding = text_tfidf_vectorizer.transform(cv_df['TEXT'])

print(train_text_feature_tfidf_Coding.shape)
print(test_text_feature_tfidf_Coding.shape)
print(cv_text_feature_tfidf_Coding.shape)
```

```
(2124, 25296)
(665, 25296)
(532, 25296)
```

In [94]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

chi = SelectKBest(chi2, k=1000).fit(train_text_feature_tfidf_Coding, y_train)

train_text_feature_tfidf_Coding = chi.transform(train_text_feature_tfidf_Coding)
test_text_feature_tfidf_Coding = chi.transform(test_text_feature_tfidf_Coding)
cv_text_feature_tfidf_Coding = chi.transform(cv_text_feature_tfidf_Coding)

print(train_text_feature_tfidf_Coding.shape)
print(test_text_feature_tfidf_Coding.shape)
print(cv_text_feature_tfidf_Coding.shape)
```

```
(2124, 1000)
(665, 1000)
(532, 1000)
```

How good is this Text feature in predicting y_i

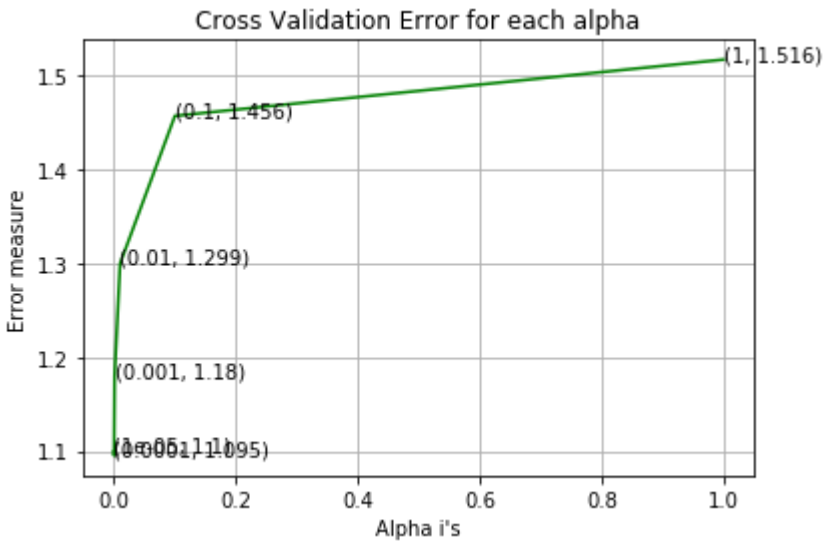
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, **we will build a logistic regression model using only Text feature (tfidf one hot encoded) to predict y_i.**

In [105]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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_tfidf_Coding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_tfidf_Coding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_tfidf_Coding)
    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, lab

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_tfidf_Coding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_tfidf_Coding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_text_feature_tfidf_Coding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
For values of alpha =  1e-05 The log loss is: 1.0997194477427663
For values of alpha =  0.0001 The log loss is: 1.0950827527902376
```

```
For values of alpha =  0.001 The log loss is: 1.1795062918286803
For values of alpha =  0.01 The log loss is: 1.2994087852057346
For values of alpha =  0.1 The log loss is: 1.456493994424038
For values of alpha =  1 The log loss is: 1.516351230587227
```




```
For values of best alpha =  0.0001 The train log loss is: 0.82787394225317
28
For values of best alpha =  0.0001 The cross validation log loss is: 1.095
0827527902376
For values of best alpha =  0.0001 The test log loss is: 1.138966499898305
8
```

1.4 Conclusion - Univariate Analysis of Feature Engineering - tfidfVec

In [215]:

```
pt = PrettyTable()

pt.field_names = ["Feature", "Vectorizer", "Best alpha", "Train log-loss", "CV log-loss
                  "Stability status "]
pt.add_row(["Gene","TfidfVectorizer", 0.001,  0.976, 1.186, 1.230, "Stable"])
pt.add_row(["Variation", "TfidfVectorizer",0.001, 1.309, 1.727, 1.70,"Less Stable" ])
pt.add_row(["Text", "TfidfVectorizer",0.0001, 0.827, 1.095, 1.138, "Stable"])
print(pt)
```

```
+-----------+-----------------+------------+----------------+-------------
+----------------+-------------------+
|  Feature  |    Vectorizer   | Best alpha | Train log-loss | CV log-loss
| Test log-loss  | Stability ststus  |
+-----------+-----------------+------------+----------------+-------------
+----------------+-------------------+
|    Gene   | TfidfVectorizer |   0.001    |     0.976      |    1.186
|      1.23      |       Stable      |
| Variation | TfidfVectorizer |   0.001    |     1.309      |    1.727
|      1.7       |    Less Stable    |
|    Text   | TfidfVectorizer |   0.0001   |     0.827      |    1.095
|     1.138      |       Stable      |
+-----------+-----------------+------------+----------------+-------------
+----------------+-------------------+
```

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

Ans. Yes, Text feature is Stable

2. Machine Learning Models

In [106]:

```
#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 e
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_
    plot_confusion_matrix(test_y, pred_y)
```

In [107]:

```
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 [108]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer(lowercase=False,min_df=10)
    var_count_vec = TfidfVectorizer(lowercase=False,min_df=10)
    text_count_vec = TfidfVectorizer(lowercase=False,min_df=10)

    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.vocabulary_.keys())
    fea2_len = len(var_count_vec.vocabulary_.keys())

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

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

Stacking the three types of features

In [109]:

```
# 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_tfidf_Coding,train_variation_f
test_gene_var_onehotCoding = hstack((test_gene_feature_tfidf_Coding, test_variation_fea
cv_gene_var_onehotCoding = hstack((cv_gene_feature_tfidf_Coding, cv_variation_feature_t

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf_Co


test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_tfidf_Codin


cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf_Coding)).to



train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_vari
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variati
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fea

train_x_responseCoding = hstack((train_gene_var_responseCoding, train_text_feature_tfid
test_x_responseCoding = hstack((test_gene_var_responseCoding, test_text_feature_tfidf_C
cv_x_responseCoding = hstack((cv_gene_var_responseCoding, cv_text_feature_tfidf_Coding)

train_y = np.array(list(train_df['Class']))
test_y = np.array(list(test_df['Class']))
cv_y = np.array(list(cv_df['Class']))
```

In [110]:

```
print("One hot encoding  tfidf features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCo
print("(number of data points * number of features) in test data = ", test_x_onehotCodi
print("(number of data points * number of features) in cross validation data =", cv_x_o
```

```
One hot encoding  tfidf features :
(number of data points * number of features) in train data =  (2124, 2229)
(number of data points * number of features) in test data =  (665, 2229)
(number of data points * number of features) in cross validation data = (5
32, 2229)
```

In [111]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseC
print("(number of data points * number of features) in test data = ", test_x_responseCo
print("(number of data points * number of features) in cross validation data =", cv_x_r
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 1018)
(number of data points * number of features) in test data =  (665, 1018)
(number of data points * number of features) in cross validation data = (5
32, 1018)
```

In [112]:

```
# https://machinelearningmastery.com/how-to-fix-futurewarning-messages-in-scikit-learn/
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
```

2.1. Base Line Model

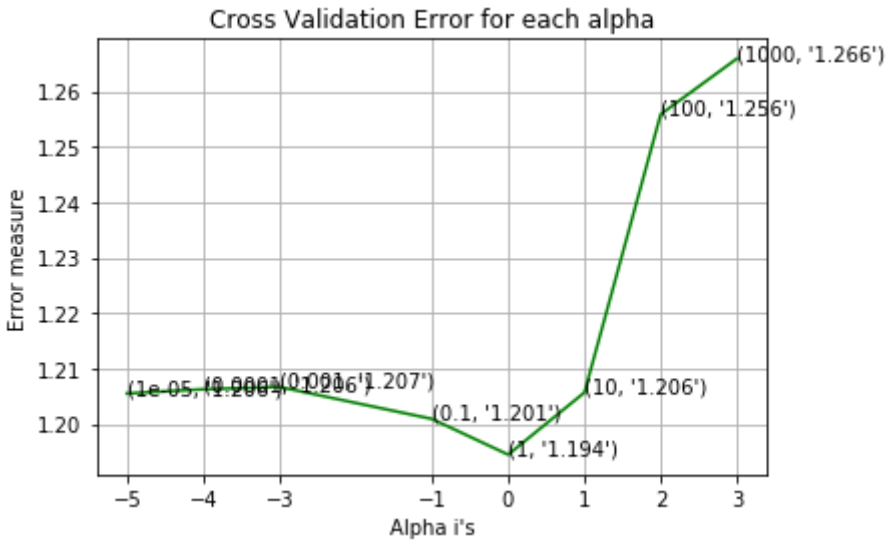
2.1.1. Naive Bayes - hyperparameter Tuning

In [113]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
# -------------------------
# 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/n
# -----------------------


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


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
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-05
Log Loss : 1.2055099573740382
for alpha = 0.0001
Log Loss : 1.2062671976006034
for alpha = 0.001
Log Loss : 1.2066904322426826
for alpha = 0.1
Log Loss : 1.2009212284130706
for alpha = 1
Log Loss : 1.1944747547978425
for alpha = 10
Log Loss : 1.2056796215570302
for alpha = 100
Log Loss : 1.255869439130139
for alpha = 1000
Log Loss : 1.2659146200076357
```



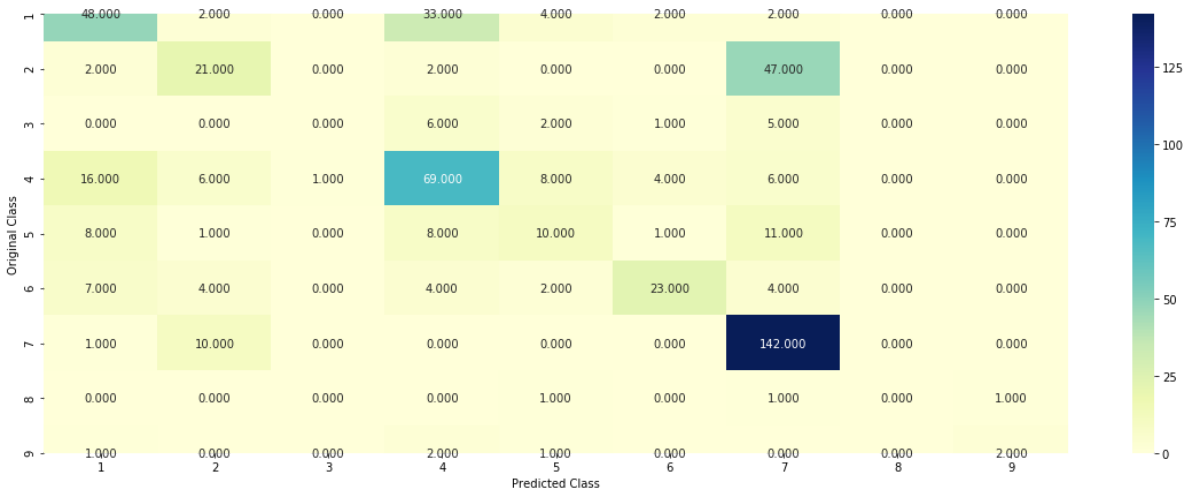

```
For values of best alpha =  1 The train log loss is: 0.8610537917305304
For values of best alpha =  1 The cross validation log loss is: 1.19447475
47978425
For values of best alpha =  1 The test log loss is: 1.1910017150133108
```

2.1.2 Testing the model with best hyperparameter

In [114]:

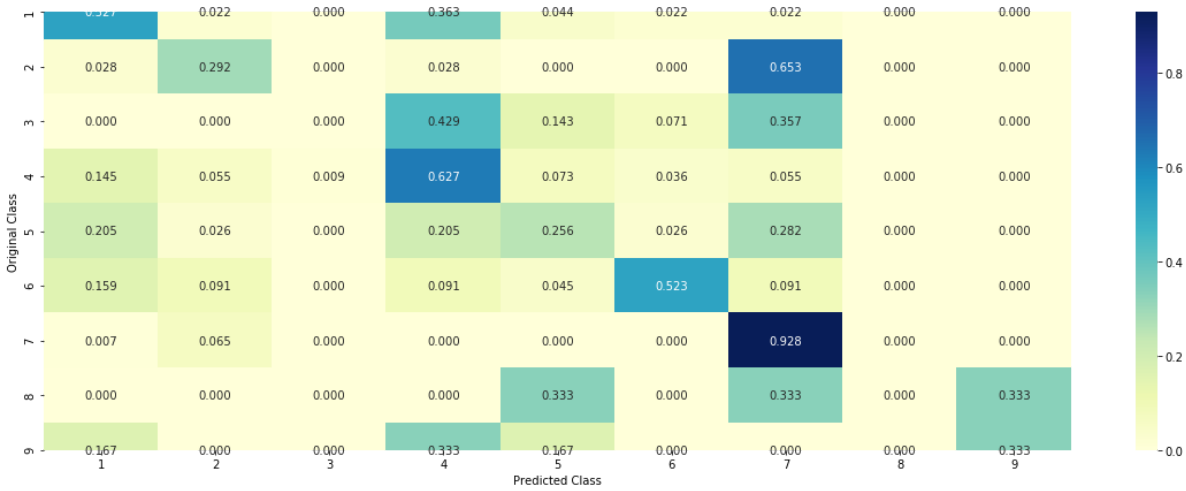
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/m
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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/n
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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 estimat
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehot
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.1944747547978425
Number of missclassified point : 0.40789473684210525
-------------------- Confusion matrix --------------------
```



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


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



2.1.3 Feature importance correctly classified points

In [115]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0481 0.0606 0.0185 0.044  0.0444 0.0396
0.7354 0.004  0.0053]]
Actual Class : 2
--------------------------------------------------
48 Text feature [44] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

2.1.4 Feature importance incorrectly classified points

In [116]:

```
test_point_index = 59
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_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0575 0.2654 0.0214 0.0639 0.0519 0.0425
0.487  0.0045 0.0059]]
Actual Class : 5
--------------------------------------------------
81 Text feature [590] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

2.2 K Nearest Neighbour Classification

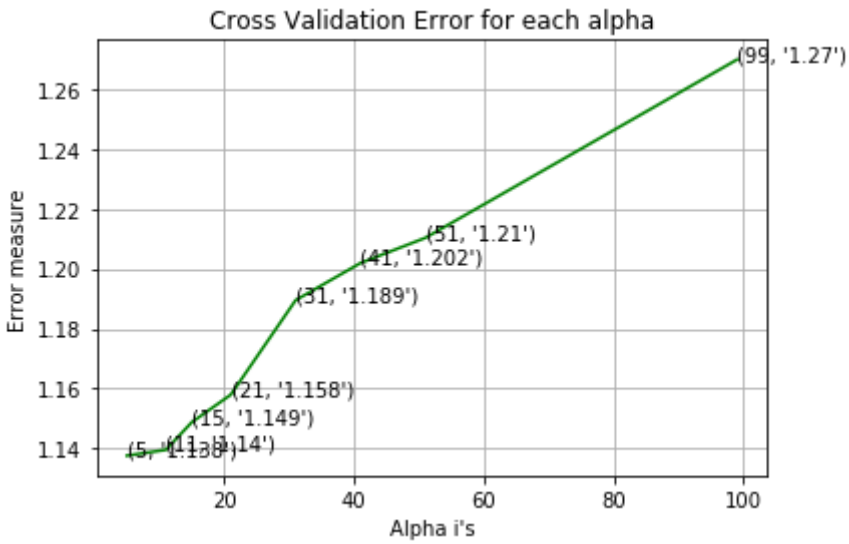
2.2.1 Hyperparameter Tuning

In [117]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
# 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
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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
    # to avoid rounding error while multiplying probabilites we use log-probability est
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 1.137566889715041
for alpha = 11
Log Loss : 1.1395010544886606
for alpha = 15
Log Loss : 1.1487235647543486
for alpha = 21
Log Loss : 1.1578983980234099
for alpha = 31
Log Loss : 1.1894804189414532
for alpha = 41
Log Loss : 1.2019039187619085
for alpha = 51
Log Loss : 1.2103233854607311
for alpha = 99
Log Loss : 1.2698462221899336
```




```
For values of best alpha =  5 The train log loss is: 0.7692618448086055
For values of best alpha =  5 The cross validation log loss is: 1.13756688
9715041
For values of best alpha =  5 The test log loss is: 1.1394406128779941
```

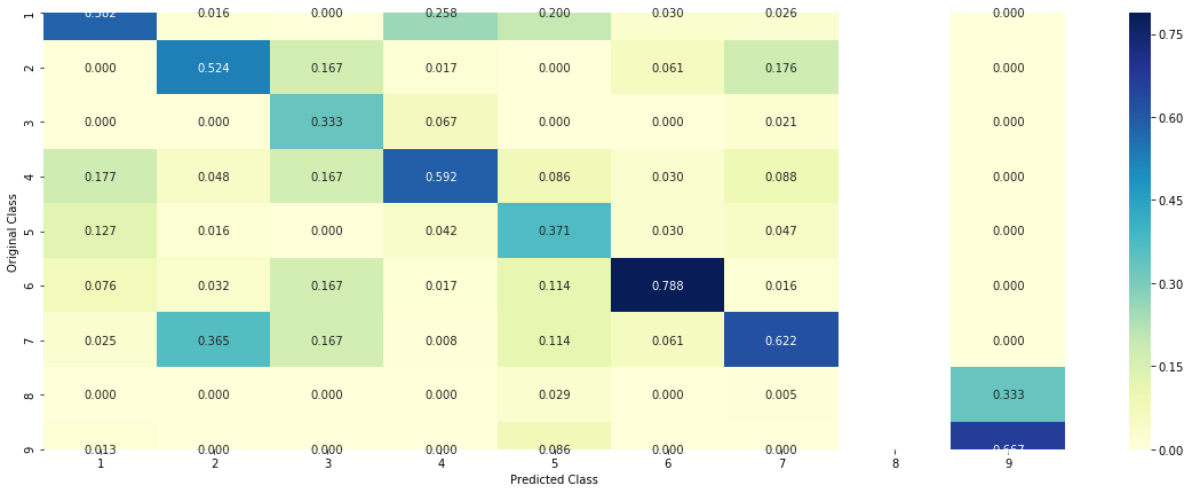
2.2.2 Testing the model with best hyperparameter

In [118]:

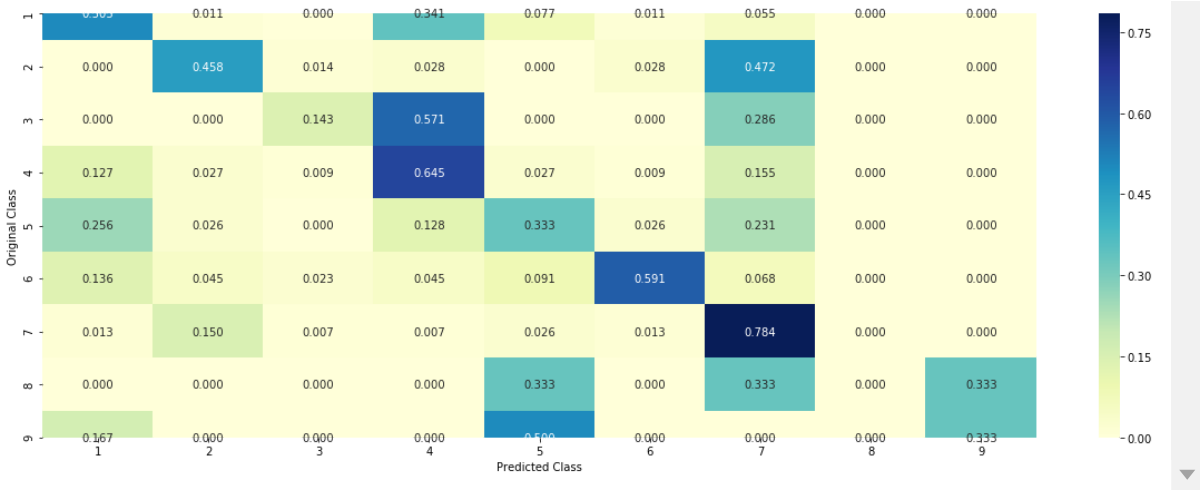
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/ge
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# 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
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding,
```

```
Log loss : 1.137566889715041
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```


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




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



2.2.3 Sample Query point -1

In [119]:

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

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

In [123]:

```
print(neighbors)
print(train_y[473])
```

```
(array([[0.33812545, 0.37864038, 0.44590971, 0.47355361, 0.47363432]]), ar
ray([[  12,  331,  666, 1388, 1591]], dtype=int64))
2
```

2.2.4 Sample Query point -2

In [124]:

```
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 = 59

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), alph
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the tes
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

2.3. Logistic Regression - with Class Balancing

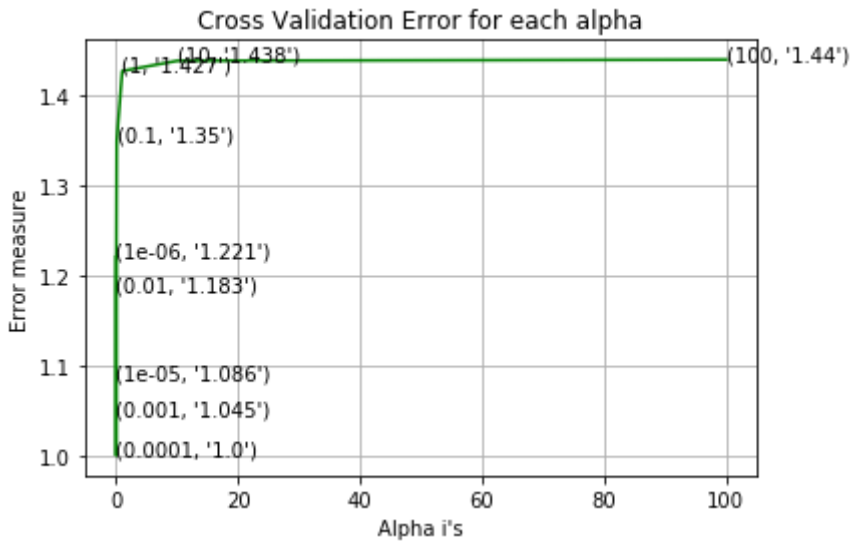
2.3.1 Hyperparameter Tuning

In [125]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# -----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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', ran
    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
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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', los
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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.2209613950741443
for alpha = 1e-05
Log Loss : 1.0861166356435763
for alpha = 0.0001
Log Loss : 1.0002183445682262
for alpha = 0.001
Log Loss : 1.0451055768725548
for alpha = 0.01
Log Loss : 1.1828049000365477
for alpha = 0.1
Log Loss : 1.3496245243305074
for alpha = 1
Log Loss : 1.4271102107165778
for alpha = 10
Log Loss : 1.4384408042975694
for alpha = 100
Log Loss : 1.4398085974886439
```




```
For values of best alpha =  0.0001 The train log loss is: 0.46441182987344
58
For values of best alpha =  0.0001 The cross validation log loss is: 1.000
2183445682262
For values of best alpha =  0.0001 The test log loss is: 1.035066947560388
9
```

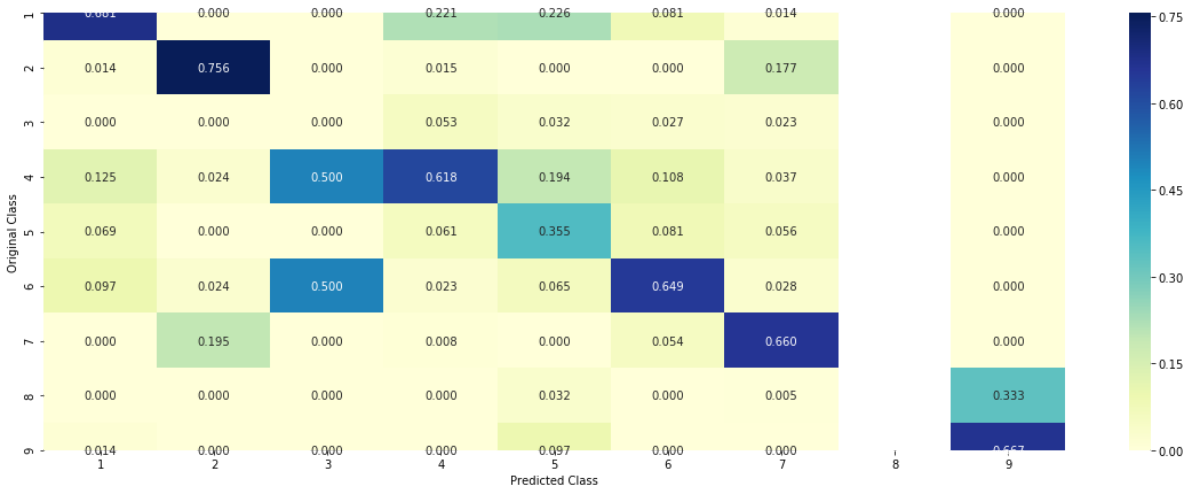
2.3.2 Testing the model with best hyperparameter

In [126]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/g
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_
```

```
Log loss : 1.0002183445682262
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```


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



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

2.3.3 Feature importance

In [127]:

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

2.3.3.1 feature importance correctly classified points

In [128]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 0
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_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2381 0.0071 0.0473 0.2564 0.1986 0.24
1  0.0066 0.0033 0.0017]]
Actual Class : 5
--------------------------------------------------
2 Text feature [02] present in test data point [True]
15 Text feature [49] present in test data point [True]
31 Text feature [10] present in test data point [True]
35 Text feature [2b] present in test data point [True]
49 Text feature [18] present in test data point [True]
57 Text feature [293t] present in test data point [True]
75 Text feature [323] present in test data point [True]
91 Text feature [2001] present in test data point [True]
93 Text feature [1059] present in test data point [True]
111 Text feature [08] present in test data point [True]
116 Text feature [1653] present in test data point [True]
142 Text feature [35] present in test data point [True]
145 Text feature [03] present in test data point [True]
167 Text feature [1a] present in test data point [True]
179 Text feature [28] present in test data point [True]
185 Text feature [1988] present in test data point [True]
187 Text feature [00] present in test data point [True]
190 Text feature [22] present in test data point [True]
195 Text feature [1998] present in test data point [True]
199 Text feature [000] present in test data point [True]
201 Text feature [2004] present in test data point [True]
215 Text feature [0001] present in test data point [True]
238 Text feature [44] present in test data point [True]
264 Text feature [037] present in test data point [True]
290 Text feature [2008] present in test data point [True]
293 Text feature [20] present in test data point [True]
297 Text feature [801] present in test data point [True]
305 Text feature [538] present in test data point [True]
315 Text feature [600185] present in test data point [True]
329 Text feature [220] present in test data point [True]
339 Text feature [355] present in test data point [True]
340 Text feature [001] present in test data point [True]
347 Text feature [54] present in test data point [True]
349 Text feature [1997] present in test data point [True]
351 Text feature [385] present in test data point [True]
361 Text feature [127] present in test data point [True]
362 Text feature [32] present in test data point [True]
375 Text feature [2009] present in test data point [True]
385 Text feature [2c] present in test data point [True]
```

```
392 Text feature [1559] present in test data point [True]
395 Text feature [1760] present in test data point [True]
406 Text feature [143] present in test data point [True]
415 Text feature [62] present in test data point [True]
417 Text feature [1974] present in test data point [True]
444 Text feature [1366] present in test data point [True]
456 Text feature [26] present in test data point [True]
480 Text feature [1989] present in test data point [True]
484 Text feature [267] present in test data point [True]
490 Text feature [112] present in test data point [True]
492 Text feature [31] present in test data point [True]
497 Text feature [53] present in test data point [True]
Out of the top  500  features  51 are present in query point
```

2.3.3.2 Feature imporatnce incorrectly classifed points

In [130]:

```
test_point_index = 361
no_feature = 59
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_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.3377 0.3499 0.0117 0.1541 0.0197 0.0178
0.0966 0.0065 0.0061]]
Actual Class : 2
--------------------------------------------------
2 Text feature [49] present in test data point [True]
6 Text feature [1c] present in test data point [True]
30 Text feature [1998b] present in test data point [True]
41 Text feature [1996a] present in test data point [True]
Out of the top  59  features  4 are present in query point
```

2.4 Logistic Regression without Class balacing

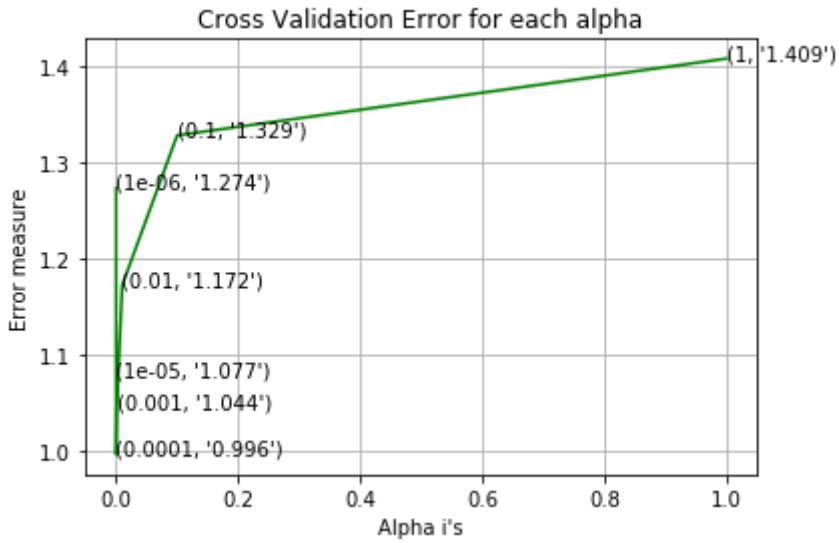
2.4.1 Hyperparameter Tuning

In [131]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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
    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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.2735591517046048
for alpha = 1e-05
Log Loss : 1.0767764421347212
for alpha = 0.0001
Log Loss : 0.9957880259218105
for alpha = 0.001
Log Loss : 1.043501479004176
for alpha = 0.01
Log Loss : 1.172498361194591
for alpha = 0.1
Log Loss : 1.328706040804446
for alpha = 1
Log Loss : 1.4087091690177886
```




```
For values of best alpha =  0.0001 The train log loss is: 0.444557386921
3398
For values of best alpha =  0.0001 The cross validation log loss is: 0.9
957880259218105
```

```
For values of best alpha =  0.0001 The test log loss is: 1.0335266051292
196
```

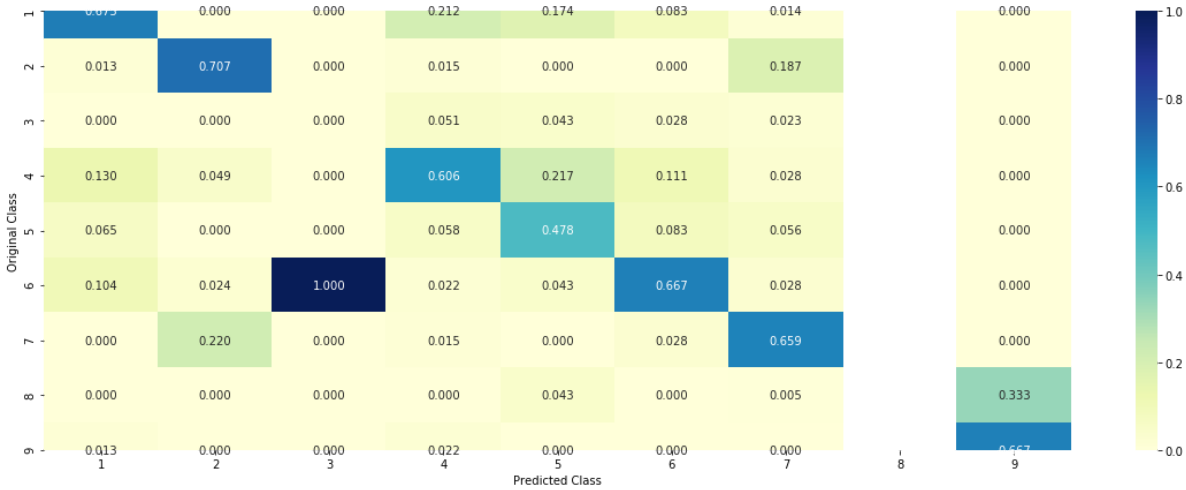
2.4.2 Testing the model with best hypereparameter

In [132]:

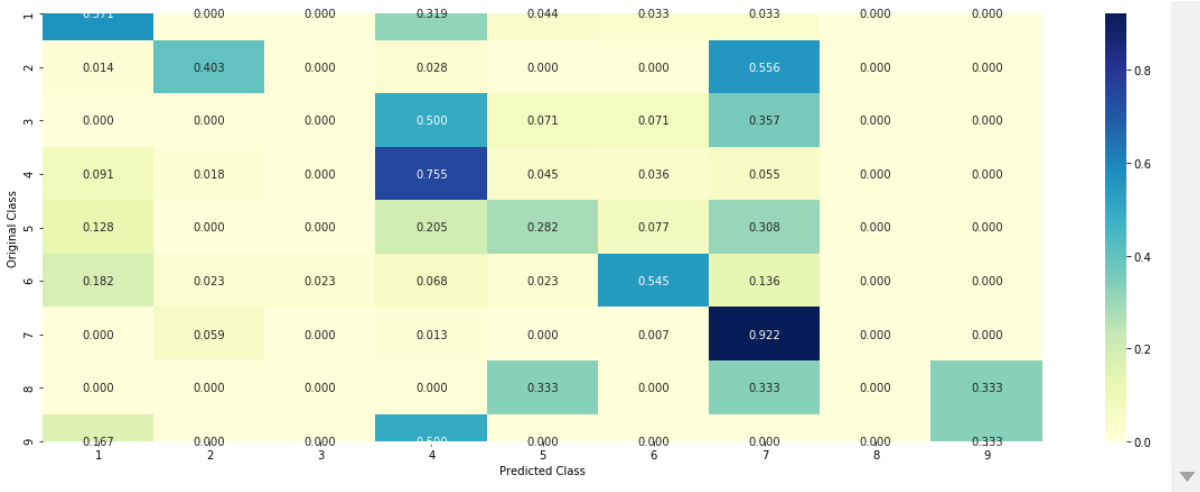
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# 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_
```

```
Log loss : 0.9957880259218105
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```


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




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



2.4.3 Feature importance correctly classified points

In [133]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 50
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_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0223 0.0547 0.0048 0.0204 0.037  0.0629
0.7923 0.003  0.0026]]
Actual Class : 2
--------------------------------------------------
13 Gene feature [KRAS] present in test data point [True]
50 Text feature [1993] present in test data point [True]
66 Text feature [12] present in test data point [True]
119 Text feature [16] present in test data point [True]
127 Text feature [1995] present in test data point [True]
142 Text feature [2000] present in test data point [True]
194 Text feature [15] present in test data point [True]
199 Text feature [1987] present in test data point [True]
313 Text feature [1b] present in test data point [True]
401 Text feature [1990] present in test data point [True]
417 Text feature [13] present in test data point [True]
445 Text feature [0003] present in test data point [True]
479 Text feature [1996] present in test data point [True]
Out of the top  500  features  13 are present in query point
```

2.4.4. Feature importance incorrectly classified points

In [134]:

```
test_point_index = 125
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_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2636 0.0256 0.0082 0.4037 0.0854 0.185
0.0164 0.0063 0.0058]]
Actual Class : 6
--------------------------------------------------
25 Text feature [2b] present in test data point [True]
29 Text feature [10] present in test data point [True]
37 Text feature [18] present in test data point [True]
149 Text feature [22] present in test data point [True]
151 Text feature [28] present in test data point [True]
159 Text feature [35] present in test data point [True]
174 Text feature [1988] present in test data point [True]
231 Text feature [20] present in test data point [True]
330 Text feature [1997] present in test data point [True]
404 Text feature [53] present in test data point [True]
439 Text feature [2c] present in test data point [True]
452 Text feature [26] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

2.5 Linear Support vector Machine

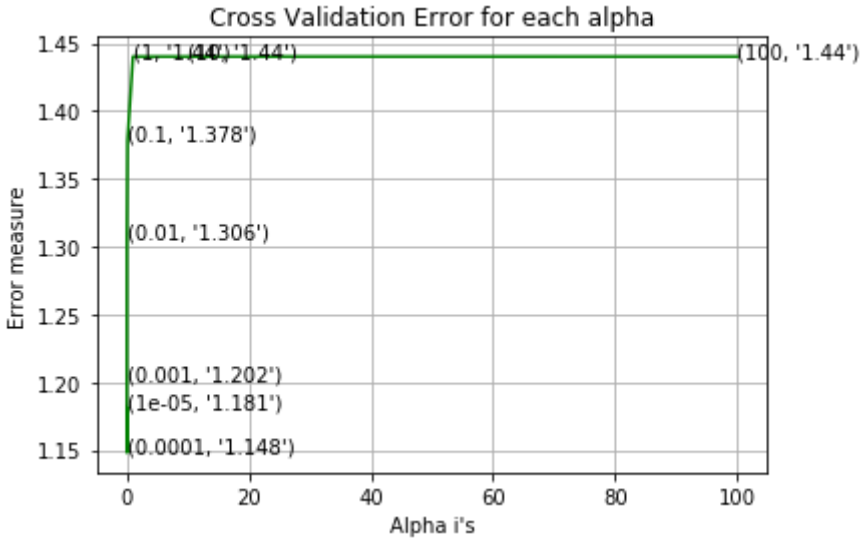
2.5.1 Hyperparameter Tuning

In [135]:

```
# read more about support vector machines with linear kernals here http://scikit-learn.

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

# 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/m
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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',
    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
    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', los
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_lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for C = 1e-05
Log Loss : 1.1809016925059925
for C = 0.0001
Log Loss : 1.1479321905591495
for C = 0.001
Log Loss : 1.202119655391704
for C = 0.01
Log Loss : 1.3064140807506655
for C = 0.1
Log Loss : 1.3783390808690765
for C = 1
Log Loss : 1.4401695252780462
for C = 10
Log Loss : 1.4401694767615567
for C = 100
Log Loss : 1.4401554374050787
```



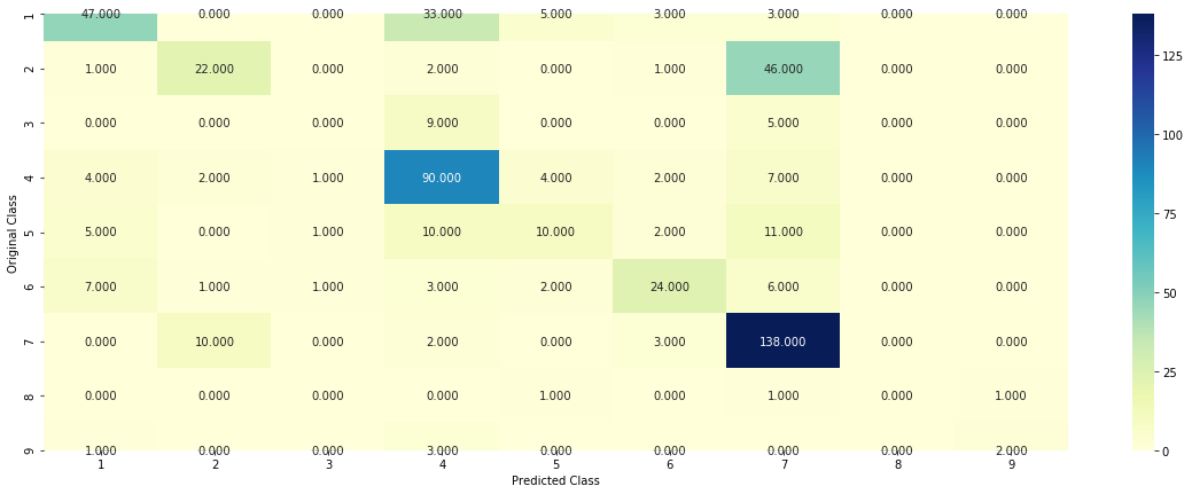

```
For values of best alpha =  0.0001 The train log loss is: 0.44070377379476
32
For values of best alpha =  0.0001 The cross validation log loss is: 1.147
9321905591495
For values of best alpha =  0.0001 The test log loss is: 1.148198755010361
4
```

2.5.2 Testing the model with best hyperparameter

In [136]:

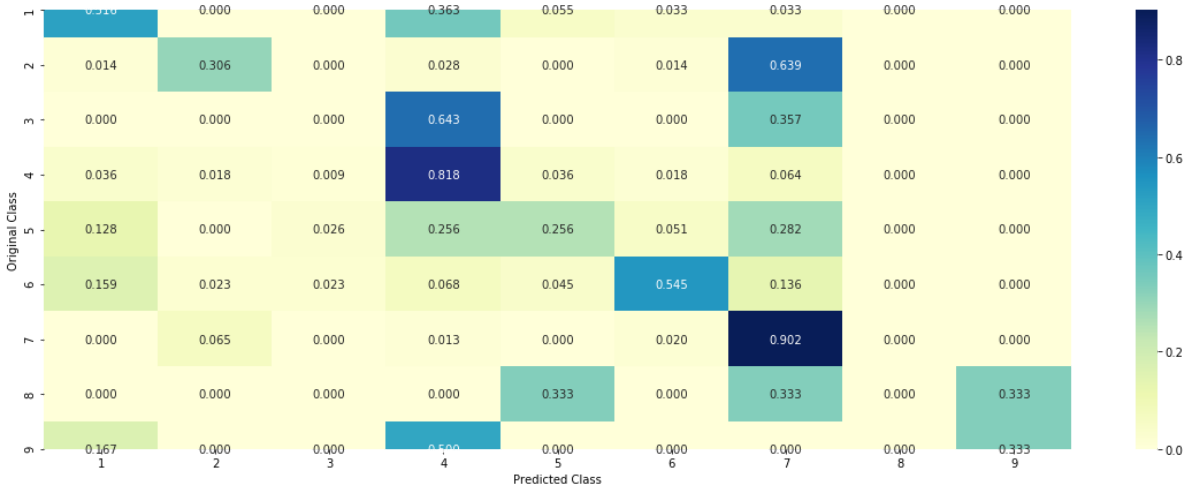
```
# read more about support vector machines with linear kernals here http://scikit-learn.

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

# 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/m
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balance
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y,
```

```
Log loss : 1.1479321905591495
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```




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

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



2.5.3 feature importance correctly classified points

In [137]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0332 0.1119 0.0092 0.0183 0.0245 0.0165
0.7794 0.0031 0.0039]]
Actual Class : 7
--------------------------------------------------
28 Text feature [1c] present in test data point [True]
35 Text feature [138] present in test data point [True]
47 Text feature [12] present in test data point [True]
66 Text feature [1b] present in test data point [True]
70 Text feature [2000] present in test data point [True]
97 Text feature [169] present in test data point [True]
114 Text feature [25th] present in test data point [True]
129 Text feature [49] present in test data point [True]
148 Text feature [14] present in test data point [True]
153 Text feature [100] present in test data point [True]
176 Text feature [259] present in test data point [True]
183 Text feature [02] present in test data point [True]
188 Text feature [16] present in test data point [True]
201 Text feature [27] present in test data point [True]
214 Text feature [149] present in test data point [True]
224 Text feature [15] present in test data point [True]
230 Text feature [13] present in test data point [True]
247 Text feature [5mm] present in test data point [True]
265 Text feature [29] present in test data point [True]
368 Text feature [4c] present in test data point [True]
387 Text feature [1039] present in test data point [True]
421 Text feature [32] present in test data point [True]
422 Text feature [80] present in test data point [True]
431 Text feature [0005] present in test data point [True]
441 Text feature [1d] present in test data point [True]
469 Text feature [24] present in test data point [True]
478 Text feature [57] present in test data point [True]
493 Text feature [569] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

2.5.4 feature importance incorrecly classified points

In [138]:

```
test_point_index = 550
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_onehotCod
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4332 0.0507 0.0109 0.3672 0.0662 0.0182
0.0429 0.006  0.0047]]
Actual Class : 1
--------------------------------------------------
0 Text feature [23] present in test data point [True]
6 Text feature [49] present in test data point [True]
11 Text feature [10] present in test data point [True]
29 Text feature [1a] present in test data point [True]
30 Text feature [222] present in test data point [True]
38 Text feature [28] present in test data point [True]
52 Text feature [18] present in test data point [True]
80 Text feature [22] present in test data point [True]
127 Text feature [1met] present in test data point [True]
146 Text feature [1640] present in test data point [True]
170 Text feature [1450] present in test data point [True]
179 Text feature [2g] present in test data point [True]
220 Text feature [200] present in test data point [True]
245 Text feature [31] present in test data point [True]
259 Gene feature [TP53] present in test data point [True]
274 Text feature [53] present in test data point [True]
283 Text feature [101] present in test data point [True]
302 Text feature [11] present in test data point [True]
312 Text feature [62] present in test data point [True]
333 Text feature [2b] present in test data point [True]
385 Text feature [393] present in test data point [True]
389 Text feature [65] present in test data point [True]
392 Text feature [41] present in test data point [True]
394 Text feature [326] present in test data point [True]
427 Text feature [446] present in test data point [True]
442 Text feature [60] present in test data point [True]
444 Text feature [55] present in test data point [True]
447 Text feature [77] present in test data point [True]
451 Text feature [44] present in test data point [True]
485 Text feature [26] present in test data point [True]
Out of the top  500  features  30 are present in query point
```

2.6 random forest - with one hot encoded TFIDF

2.6.1 Hyperparameter tuning

In [139]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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/r
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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, ran
        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_, ep
        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_err
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', ma
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 i
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validatio
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss i
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.184667562012646
for n_estimators = 100 and max depth =  10
Log Loss : 1.089586501931268
for n_estimators = 200 and max depth =  5
Log Loss : 1.1798418694003792
for n_estimators = 200 and max depth =  10
Log Loss : 1.0869922270269479
for n_estimators = 500 and max depth =  5
Log Loss : 1.181593968376849
for n_estimators = 500 and max depth =  10
Log Loss : 1.0881310436087546
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1854672515365592
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0872662597507492
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1879171024259012
for n_estimators = 2000 and max depth =  10
Log Loss : 1.0864094166809553
For values of best estimator =  2000 The train log loss is: 0.655907713463
1894
For values of best estimator =  2000 The cross validation log loss is: 1.0
864094166809553
For values of best estimator =  2000 The test log loss is: 1.1244432502909
112
```

2.6.2 Testing the model with best hyperparamter

In [140]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None
# 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/r
# -------------------------------

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

```
Log loss : 1.0864094166809553
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```


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

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

2.6.3 Feature importance correctly classified points

In [141]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', m
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 56
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_onehotCod
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_
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3315 0.0227 0.0158 0.5237 0.0389 0.0336
0.0229 0.0048 0.006 ]]
Actual Class : 4
--------------------------------------------------
70 Text feature [58] present in test data point [True]
85 Text feature [39] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

2.6.4 Feature importance incorrectly classified points

In [142]:

```
test_point_index = 550
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_onehotCod
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_
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5164 0.0228 0.0137 0.3422 0.0364 0.0319
0.0258 0.0048 0.0059]]
Actuall Class : 1
--------------------------------------------------
0 Text feature [505] present in test data point [True]
4 Text feature [77] present in test data point [True]
5 Text feature [4d] present in test data point [True]
10 Text feature [4c] present in test data point [True]
11 Text feature [590] present in test data point [True]
24 Text feature [32] present in test data point [True]
30 Text feature [550] present in test data point [True]
70 Text feature [58] present in test data point [True]
84 Text feature [354] present in test data point [True]
85 Text feature [39] present in test data point [True]
Out of the top  100  features  10 are present in query point
```

2.7 Random Forest - with response coding

2.7.1 Hyperparameter Tuning

In [144]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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/r
# --------------------------------


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, ran
        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_, ep
        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_err
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', ma
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 1.4536655224674255
for n_estimators = 10 and max depth =  3
Log Loss : 1.2887691291095762
for n_estimators = 10 and max depth =  5
Log Loss : 1.110909429028989
for n_estimators = 10 and max depth =  10
Log Loss : 0.9781654084565967
for n_estimators = 50 and max depth =  2
Log Loss : 1.2868851124262537
for n_estimators = 50 and max depth =  3
Log Loss : 1.1834040023555004
for n_estimators = 50 and max depth =  5
Log Loss : 1.0649462752313883
for n_estimators = 50 and max depth =  10
Log Loss : 0.9604947063799902
for n_estimators = 100 and max depth =  2
Log Loss : 1.32290738303653
for n_estimators = 100 and max depth =  3
Log Loss : 1.2142465086285095
for n_estimators = 100 and max depth =  5
Log Loss : 1.0699986634552285
for n_estimators = 100 and max depth =  10
Log Loss : 0.9595854883862991
for n_estimators = 200 and max depth =  2
Log Loss : 1.299813951236349
for n_estimators = 200 and max depth =  3
Log Loss : 1.2079727731587535
for n_estimators = 200 and max depth =  5
Log Loss : 1.0803927409822072
for n_estimators = 200 and max depth =  10
Log Loss : 0.9582320049795594
for n_estimators = 500 and max depth =  2
Log Loss : 1.2762926321104902
for n_estimators = 500 and max depth =  3
Log Loss : 1.1899990442975985
for n_estimators = 500 and max depth =  5
Log Loss : 1.0718358239188777
for n_estimators = 500 and max depth =  10
Log Loss : 0.9593978863422857
for n_estimators = 1000 and max depth =  2
Log Loss : 1.274946191178022
for n_estimators = 1000 and max depth =  3
Log Loss : 1.1855966445766681
for n_estimators = 1000 and max depth =  5
```

```
Log Loss : 1.0738276785120329
for n_estimators = 1000 and max depth =  10
Log Loss : 0.9640383193252493
For values of best alpha =  200 The train log loss is: 0.09384624124233819
For values of best alpha =  200 The cross validation log loss is: 0.958232
0049795595
For values of best alpha =  200 The test log loss is: 0.9697113579304354
```

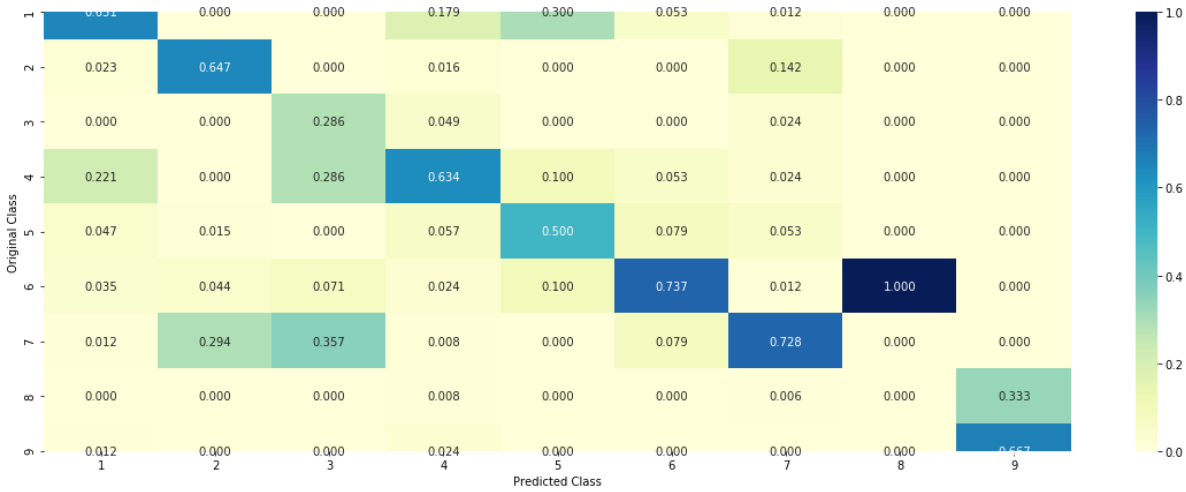
2.7.2 Testing the model with best hyperparameter

In [145]:

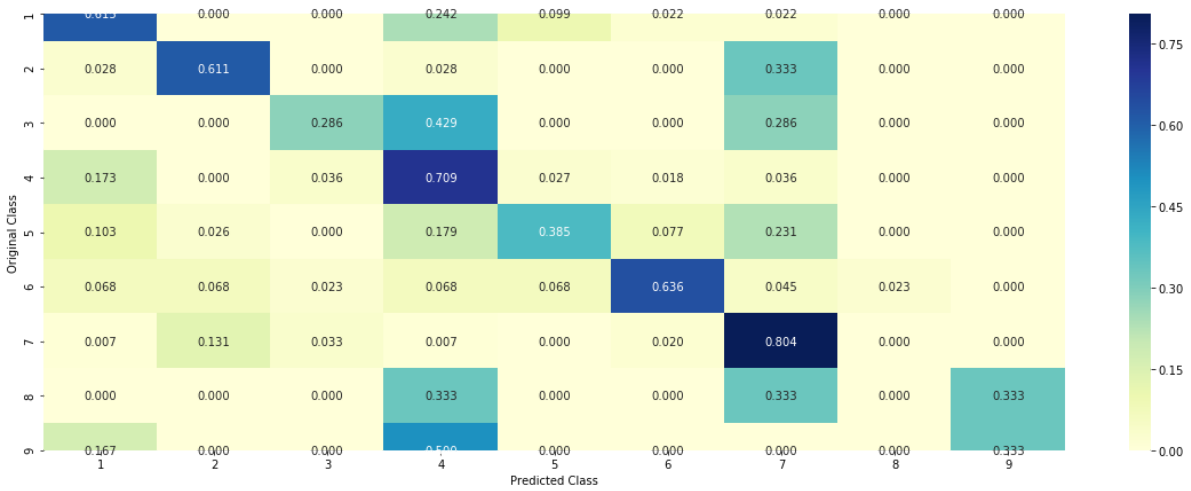
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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/r
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,c
```

```
Log loss : 0.9582320049795594
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```


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



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



2.7.3 Feature importance correctly classified points

In [146]:

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


test_point_index = 50
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_responseC
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0378 0.0779 0.0361 0.0366 0.0636 0.16
92 0.5313 0.0226 0.025 ]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
```

2.7.4 Feature importance incorrectly classified points

In [147]:

```
test_point_index = 550
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_responseC
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[6.530e-01 2.000e-03 9.700e-03 2.854e-01
1.730e-02 2.080e-02 3.000e-04
  4.400e-03 7.100e-03]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
```

2.8 Stack the models

2.8.1 Hyperparameter Tuning

In [148]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# read more about support vector machines with linear kernals here http://scikit-learn.
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabili
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

# 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/m
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit-learn.
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None
# 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/r
# --------------------------------


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', rand
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
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_pr
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_onel
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_classifi
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_l
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

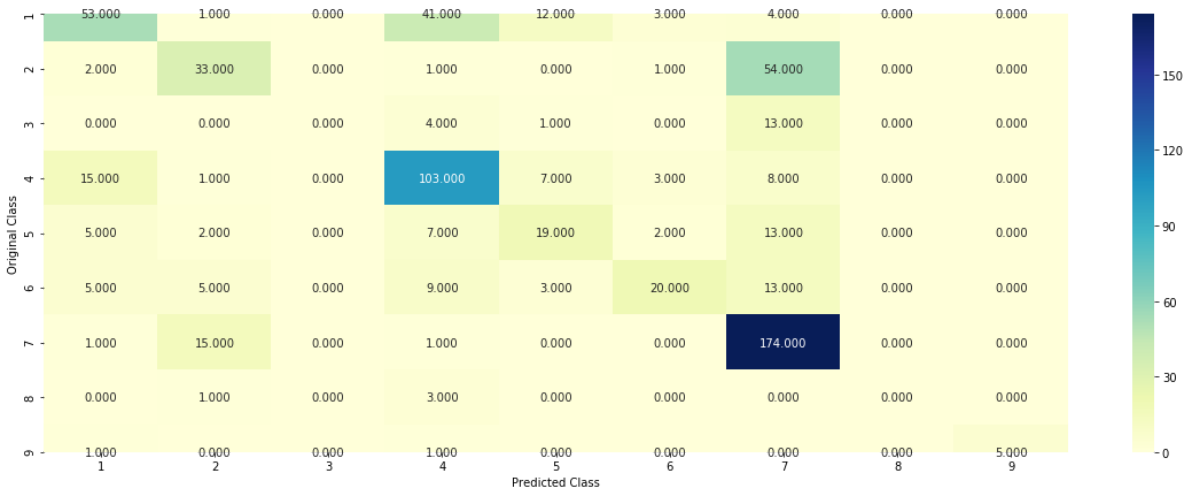
```
Logistic Regression :  Log Loss: 1.04
Support vector machines : Log Loss: 1.44
Naive Bayes : Log Loss: 1.21
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.717
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.340
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.238
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.502
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.771
```

2.8. 2 testing the model with best hyperparameter

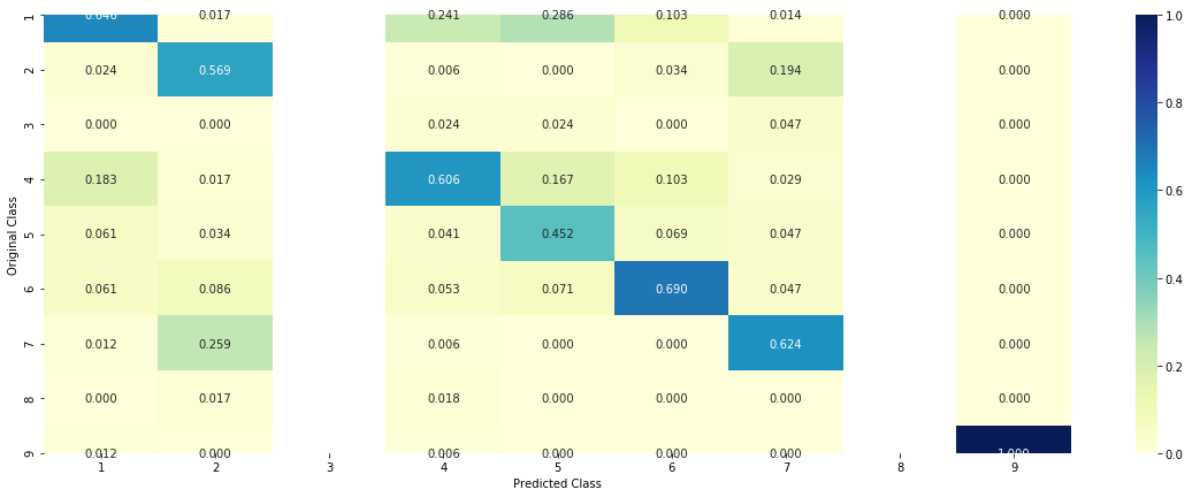
In [149]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
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_onehotC
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

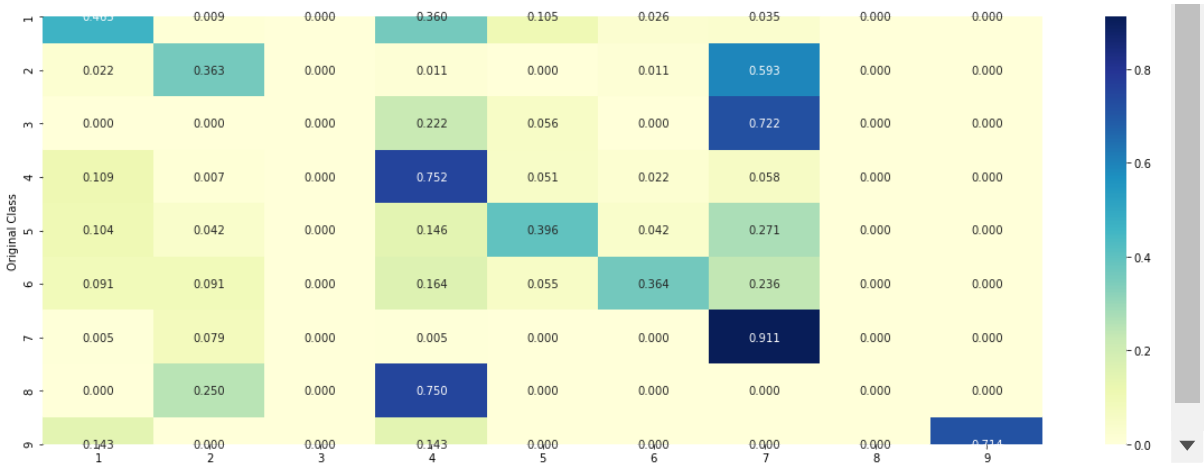
```
Log loss (train) on the stacking classifier : 0.46189019891464417
Log loss (CV) on the stacking classifier : 1.2376050864007015
Log loss (test) on the stacking classifier : 1.2584262545322125
Number of missclassified point : 0.3879699248120301
-------------------- Confusion matrix --------------------
```




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




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

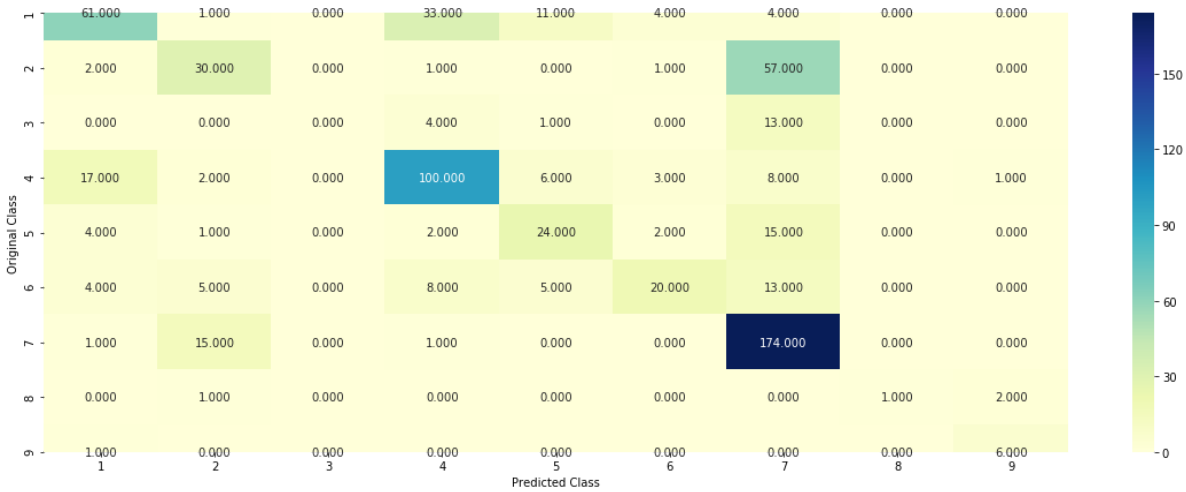


2.8.3 Maximum Voting Classifier

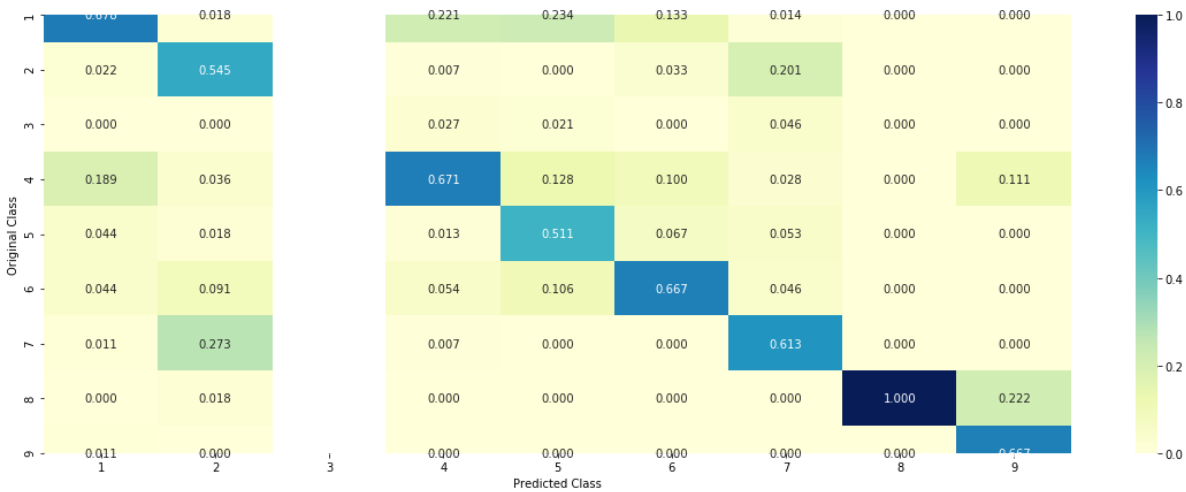
In [150]:

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

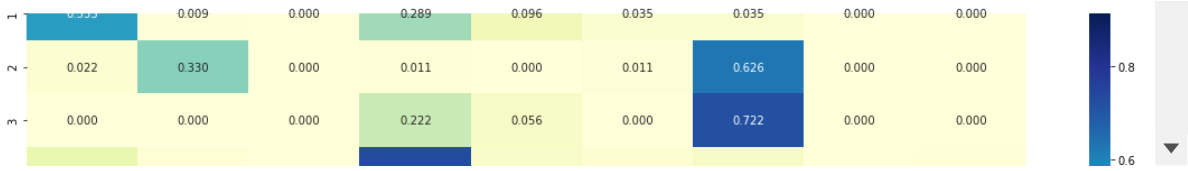
```
Log loss (train) on the VotingClassifier : 0.7764604310544412
Log loss (CV) on the VotingClassifier : 1.1634384626802798
Log loss (test) on the VotingClassifier : 1.173164304724636
Number of missclassified point : 0.3744360902255639
-------------------- Confusion matrix --------------------
```




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




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



A3. Logistic Regression with CountVectorizer(unigrams, bigrams)

1.1 Univariate Analysis on Gene Feature - CountVectorizer(unigrams , bi-grams)

In [180]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_gene_feature_ngram_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_ngram_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_ngram_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

print("Shape of train_gene_feature_ngram_onehotCoding:", train_gene_feature_ngram_onehot
print("Shape of test_gene_feature_ngram_onehotCoding :", test_gene_feature_ngram_onehot
print("Shape of cv_gene_feature_ngram_onehotCoding   :", cv_gene_feature_ngram_onehotCo
print("Gene vectorizer feature names :", gene_vectorizer.get_feature_names()[:10])
```

```
Shape of train_gene_feature_ngram_onehotCoding: (2124, 229)
Shape of test_gene_feature_ngram_onehotCoding : (665, 229)
Shape of cv_gene_feature_ngram_onehotCoding   : (532, 229)
Gene vectorizer feature names : ['abl1', 'acvr1', 'ago2', 'akt1', 'akt2',
'akt3', 'alk', 'apc', 'ar', 'araf']
```

1.2 Univariate Analysis on Variation Feature - CountVectorizer(unigrams , bi-grams)

In [181]:

```
# one-hot encoding of Gene feature.
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_ngram_onehotCoding = variation_vectorizer.fit_transform(train_d
test_variation_feature_ngram_onehotCoding = variation_vectorizer.transform(test_df['Var
cv_variation_feature_ngram_onehotCoding = variation_vectorizer.transform(cv_df['Variati

print("Shape of train_variation_feature_ngram_onehotCoding:", train_variation_feature_n
print("Shape of test_variation_feature_ngram_onehotCoding :", test_variation_feature_ng
print("Shape of cv_variation_feature_ngram_onehotCoding   :", cv_variation_feature_ngra
print("Variation vectorizer feature names :", variation_vectorizer.get_feature_names()[
```

```
Shape of train_variation_feature_ngram_onehotCoding: (2124, 2060)
Shape of test_variation_feature_ngram_onehotCoding : (665, 2060)
Shape of cv_variation_feature_ngram_onehotCoding   : (532, 2060)
Variation vectorizer feature names : ['17', '1_fusion', '2010_2471trunc',
'256_286trunc', '385_418del', '534_536del', '550_592del', '560_561inser',
'596_619splice', '963_d1010splice']
```

1.3 Univariate Analysis on Text Feature - CountVectorizer(unigrams , bi-

grams)

In [182]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train d
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_ngram_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it
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 : 799305
```

In [183]:

```
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 [184]:

```
# don't forget to normalize every feature
train_text_feature_ngram_onehotCoding = normalize.fit_transform(train_text_feature_ngra

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

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

In [185]:

```
print("Total number of unique words in train data :", len(text_vectorizer.get_feature_n
```

```
Total number of unique words in train data : 799305
```

In [186]:

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

In [187]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur[:10]))
```

```
Counter({152709: 1, 118652: 1, 82393: 1, 67828: 1, 67689: 1, 67112: 1, 670
72: 1, 65406: 1, 64159: 1, 63156: 1})
```

In [188]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3, ngram_range=(1,2))
    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 [189]:

```
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 da
```

```
93.804 % of word of test data appeared in train data
92.2 % of word of Cross Validation appeared in train data
```

In [190]:

```
#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 e
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_
    plot_confusion_matrix(test_y, pred_y)
```

In [191]:

```
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 [192]:

```
# 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(ngram_range=(1,2))
    var_count_vec = CountVectorizer(ngram_range=(1,2))
    text_count_vec = CountVectorizer(ngram_range=(1,2))

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

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

Stacking the three types of features

In [193]:

```
# 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_ngram_onehotCoding = hstack((train_gene_feature_ngram_onehotCoding,train_
test_gene_var_ngram_onehotCoding = hstack((test_gene_feature_ngram_onehotCoding,test_va
cv_gene_var_ngram_onehotCoding = hstack((cv_gene_feature_ngram_onehotCoding,cv_variatio

train_x_ngram_onehotCoding = hstack((train_gene_var_ngram_onehotCoding, train_text_feat


test_x_ngram_onehotCoding = hstack((test_gene_var_ngram_onehotCoding, test_text_feature


cv_x_ngram_onehotCoding = hstack((cv_gene_var_ngram_onehotCoding, cv_text_feature_ngram



train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_vari
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variati
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fea

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_r
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_resp
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCo

train_y = np.array(list(train_df['Class']))
test_y = np.array(list(test_df['Class']))
cv_y = np.array(list(cv_df['Class']))
```

In [194]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_ngram_on
print("(number of data points * number of features) in test data = ", test_x_ngram_oneh
print("(number of data points * number of features) in cross validation data =", cv_x_n
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 80159
4)
(number of data points * number of features) in test data =  (665, 801594)
(number of data points * number of features) in cross validation data = (5
32, 801594)
```

2. Logistic Regression - with Class balancing

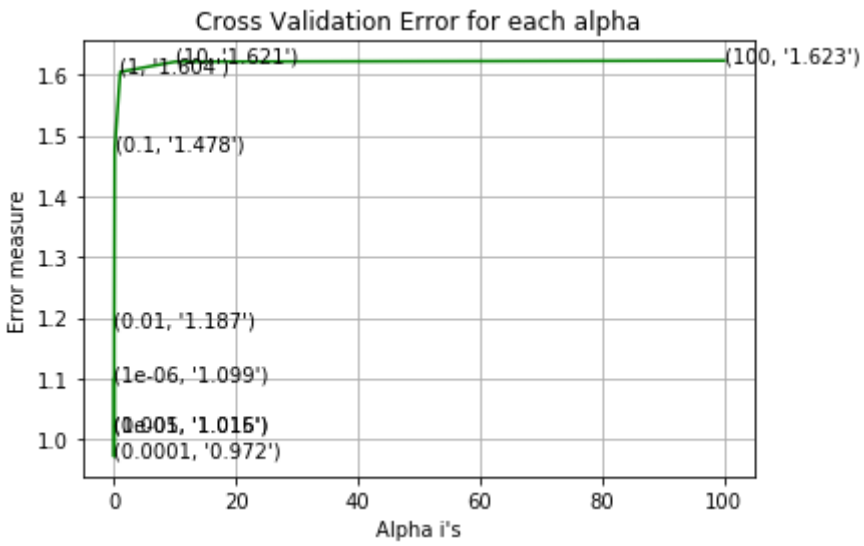
2.1.1 Hyperparameter Tuning

In [237]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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', ran
    clf.fit(train_x_ngram_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngram_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngram_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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', los
clf.fit(train_x_ngram_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_ngram_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_ngram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_ngram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_ngram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.0986871323881453
for alpha = 1e-05
Log Loss : 1.016488763089549
for alpha = 0.0001
Log Loss : 0.9717603022242269
for alpha = 0.001
Log Loss : 1.0148932769674794
for alpha = 0.01
Log Loss : 1.1865427593935765
for alpha = 0.1
Log Loss : 1.477915439077639
for alpha = 1
Log Loss : 1.6043405965681714
for alpha = 10
Log Loss : 1.6210726191483362
for alpha = 100
Log Loss : 1.6229161917597845
```




```
For values of best alpha =  0.0001 The train log loss is: 0.37556199712578
21
For values of best alpha =  0.0001 The cross validation log loss is: 0.971
7603022242269
For values of best alpha =  0.0001 The test log loss is: 0.995165116730416
2
```

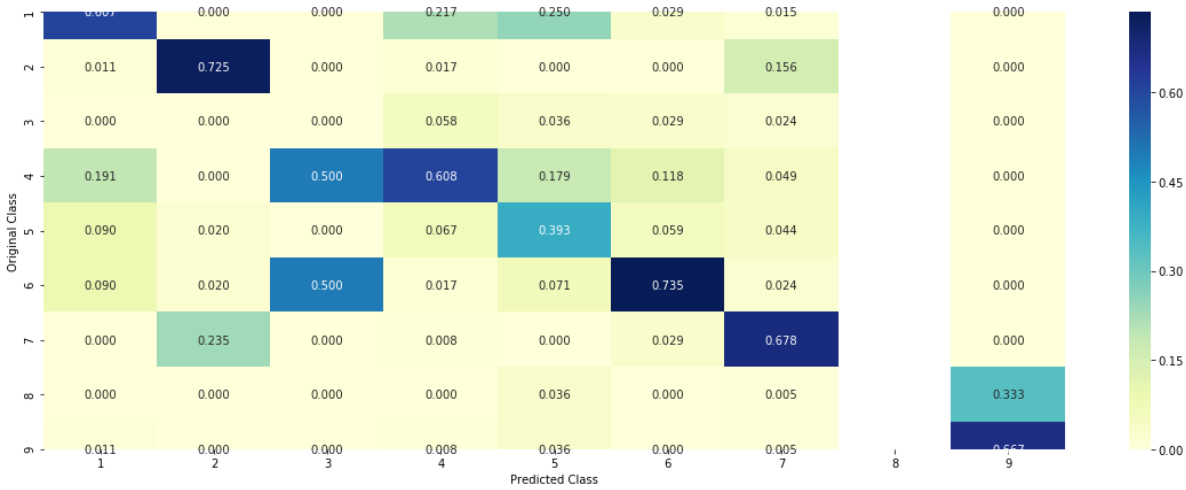
2.1.2 testing the model with best hyperparameter

In [196]:

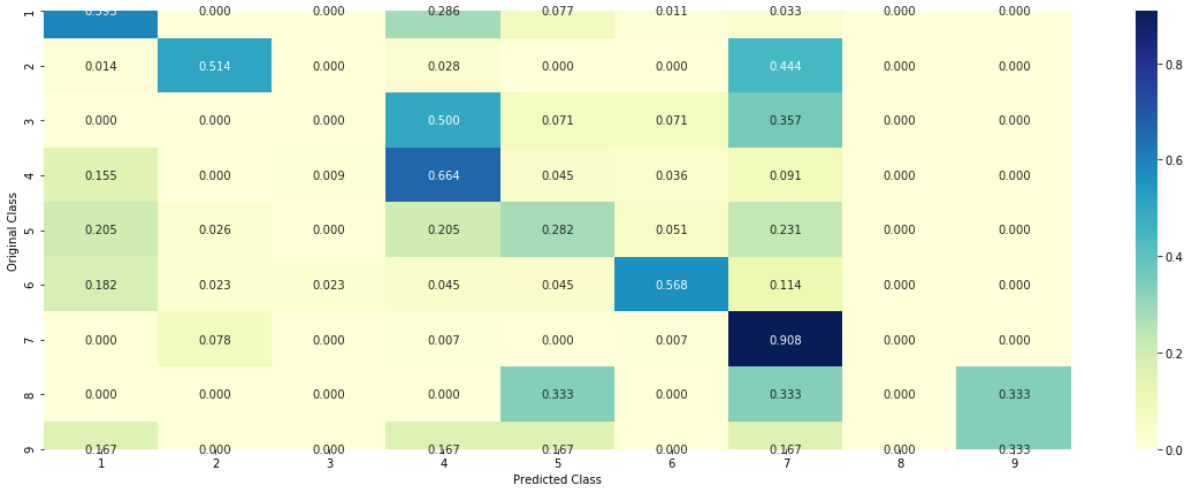
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
predict_and_plot_confusion_matrix(train_x_ngram_onehotCoding, train_y, cv_x_ngram_onehot
```

```
Log loss : 0.9717603022242269
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```


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



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



2.1.3 Feature importance correctly classified

In [197]:

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

In [198]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
clf.fit(train_x_ngram_onehotCoding,train_y)
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_ngram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngram_onel
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8599 0.0263 0.0068 0.0296 0.0149 0.0137
0.0418 0.0037 0.0032]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

2.1.4 Feature importance incorrectly classified

In [199]:

```
test_point_index = 361
no_feature = 100
predicted_cls = sig_clf.predict(test_x_ngram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngram_onel
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.305  0.4368 0.0068 0.1459 0.0152 0.0081
0.0787 0.0021 0.0015]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

2.2 Logistic Regression witout class balancing

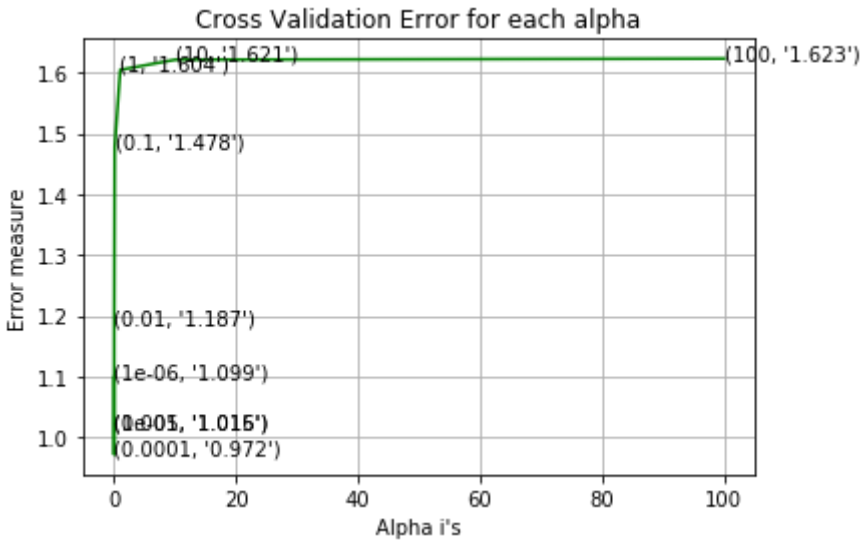
2.2.1 Hyperparameter Tuning

In [200]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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', ran
    clf.fit(train_x_ngram_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngram_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngram_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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_ngram_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_ngram_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_ngram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_ngram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_ngram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.0986871323881453
for alpha = 1e-05
Log Loss : 1.016488763089549
for alpha = 0.0001
Log Loss : 0.9717603022242269
for alpha = 0.001
Log Loss : 1.0148932769674794
for alpha = 0.01
Log Loss : 1.1865427593935765
for alpha = 0.1
Log Loss : 1.477915439077639
for alpha = 1
Log Loss : 1.6043405965681714
for alpha = 10
Log Loss : 1.6210726191483362
for alpha = 100
Log Loss : 1.6229161917597845
```



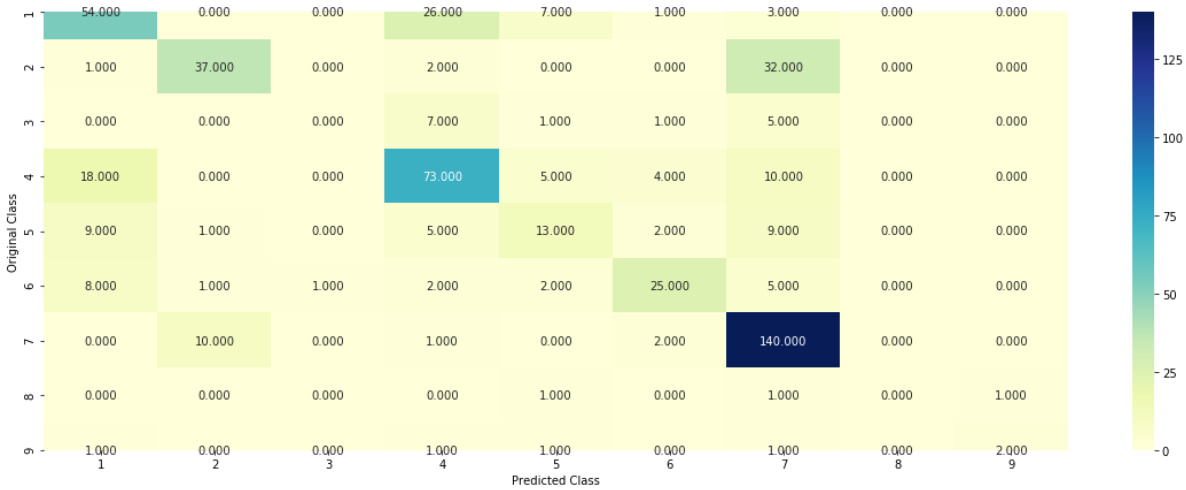

```
For values of best alpha =  0.0001 The train log loss is: 0.36290406339328
274
For values of best alpha =  0.0001 The cross validation log loss is: 0.968
8029563373155
For values of best alpha =  0.0001 The test log loss is: 0.992959236097336
4
```

2.2.2 Testing the model with best hyperparameter

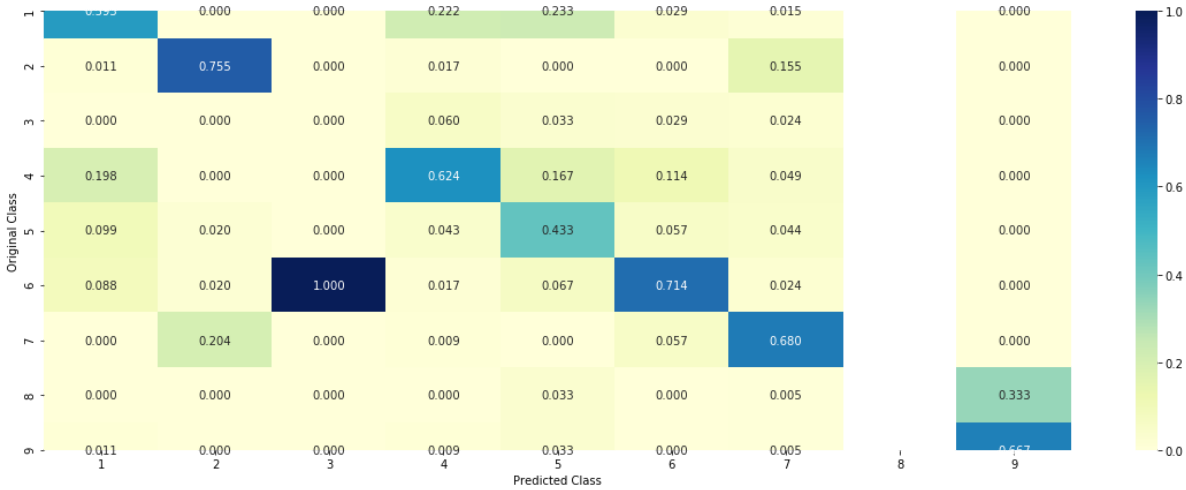
In [201]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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

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

```
Log loss : 0.9688029563373155
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




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




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

2.2.3 Feature importance correctly classified

In [202]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
clf.fit(train_x_ngram_onehotCoding,train_y)
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_ngram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngram_onel
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8616 0.026  0.0062 0.0272 0.015  0.0134
0.0449 0.0027 0.0028]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

2.2.4 Feature importance incorrectly classified

In [203]:

```
test_point_index = 361
no_feature = 100
predicted_cls = sig_clf.predict(test_x_ngram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngram_onel
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.305  0.4281 0.0061 0.1512 0.0151 0.0078
0.0837 0.0017 0.0013]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Conclusion ::

In [212]:

```
from prettytable import PrettyTable

p = PrettyTable()

p.field_names=["Model Name ", "Vectorizer", "Train Loss", "CV Loss", "Test loss", " Mis
p.add_row(["Naive Bayes","CountVectorizer", 0.827, 1.262, 1.291, 0.398])
p.add_row(["KNN", "CountVectorizer", 0.655, 1.084, 1.087, 0.390])
p.add_row(["LR with class-bal", "CountVectorizer", 0.516, 1.082, 1.115, 0.383])
p.add_row(["LR without class-bal", "CountVectorizer", 0.513, 1.138, 1.112, 0.377])
p.add_row(["Linear SVM", "CountVectorizer", 0.714, 1.163, 1.181, 0.394])
p.add_row(["RF - One-Hot ","CountVectorizer", 0.674, 1.184, 1.223, 0.394])
p.add_row(["RF - response code", "Response Coding", 0.070, 1.327, 1.309, 0.409])
p.add_row(["stack classifier", "CountVectorizer", 0.484, 1.198, 1.191, 0.372])
p.add_row(["Max-Voting classifier ", "CountVectorizer", 0.841, 1.193, 1.224, 0.369])
p.add_row(["\n", "\n",  "\n","\n","\n","\n" ])
p.add_row(["Naive Bayes", "TfidfVectorizer", 0.861, 1.194, 1.191, 0.407])
p.add_row(['KNN', "TfidfVectorizer", 0.769, 1.137, 1.139, 0.411])
p.add_row(["LR with class-balance", "TfidfVectorizer", 0.464, 1.000, 1.035, 0.360])
p.add_row(["LR without Class_balance", "TfidfVectorizer", 0.444, 0.995, 1.033, 0.357])
p.add_row(["Liner SVM", "TfidfVectorizer", 0.440, 1.147, 1.148,  0.374 ])
p.add_row(["RF one-hot", "TfidfVectorizer", 0.655, 1.086, 1.124, 0.362])
p.add_row(["RF - Response code", "Response Coding", 0.093, 0.095, 0.969, 0.342])
p.add_row(["Stack Calssifier", "TfidfVectorizer", 0.461, 1.237, 1.258, 0.387])
p.add_row(["Max voting clf", "TfidfVectorizer", 0.776, 1.163, 1.173, 0.374])
p.add_row(["\n", "\n", "\n","\n","\n","\n" ])
p.add_row(["LR with class_balance", "CountVec_unigram_bigram", 0.374, 0.971, 0.995, 0.3
p.add_row(["LR without class_balance", "CountVec_unigram_bigram", 0.362, 0.968, 0.992, (
print(p)
```

```
+-------------------------+-------------------------+------------+-------
--+-----------+-------------------------+
|       Model Name        |        Vectorizer       | Train Loss | CV Los
s | Test loss |  Mis Classified points  |
+-------------------------+-------------------------+------------+-------
--+-----------+-------------------------+
|       Naive Bayes       |     CountVectorizer     |   0.827    |  1.262
|   1.291   |          0.398          |
|           KNN           |     CountVectorizer     |   0.655    |  1.084
|   1.087   |           0.39          |
|    LR with class-bal    |     CountVectorizer     |   0.516    |  1.082
|   1.115   |          0.383          |
|   LR without class-bal  |     CountVectorizer     |   0.513    |  1.138
|   1.112   |          0.377          |
|        Linear SVM       |     CountVectorizer     |   0.714    |  1.163
|   1.181   |          0.394          |
|      RF - One-Hot       |     CountVectorizer     |   0.674    |  1.184
|   1.223   |          0.394          |
|    RF - response code   |     Response Coding     |    0.07    |  1.327
|   1.309   |          0.409          |
|     stack classifier    |     CountVectorizer     |   0.484    |  1.198
|   1.191   |          0.372          |
|  Max-Voting classifier  |     CountVectorizer     |   0.841    |  1.193
|   1.224   |          0.369          |
|                         |                         |            |
|           |                         |
|                         |                         |            |
|           |                         |
|       Naive Bayes       |     TfidfVectorizer     |   0.861    |  1.194
|   1.191   |          0.407          |
```

```
|           KNN            |     TfidfVectorizer     |   0.769    |  1.137
|   1.139   |          0.411          |
|  LR with class-balance   |     TfidfVectorizer     |   0.464    |   1.0
|   1.035   |           0.36          |
| LR without Class_balance |     TfidfVectorizer     |   0.444    |  0.995
|   1.033   |          0.357          |
|        Liner SVM         |     TfidfVectorizer     |    0.44    |  1.147
|   1.148   |          0.374          |
|        RF one-hot        |     TfidfVectorizer     |   0.655    |  1.086
|   1.124   |          0.362          |
|    RF - Response code    |     Response Coding     |   0.093    |  0.095
|   0.969   |          0.342          |
|     Stack Calssifier     |     TfidfVectorizer     |   0.461    |  1.237
|   1.258   |          0.387          |
|      Max voting clf      |     TfidfVectorizer     |   0.776    |  1.163
|   1.173   |          0.374          |
|                          |                         |            |
|           |                         |
|                          |                         |            |
|           |                         |
|  LR with class_balance   | CountVec_unigram_bigram |   0.374    |  0.971
|   0.995   |          0.359          |
| LR without class_balance | CountVec_unigram_bigram |   0.362    |  0.968
|   0.992   |          0.353          |
+--------------------------+-------------------------+------------+-------
--+-----------+-------------------------+
```

- Our main objective is to minimize the loss to lees than 1. But by replacing Bag of Words representation with tfidf, we almost achieved with some simple model and we could not achieved with some other modles.

- By looking at the above mentioned table, we can say that though the model Logistic Regression without class balancing performs well and we reduced the log loss less than 1.

- Took Top 1000 words based on Tf-Idf values of Text Feature. Out of 9 models, two models performed equally well i.e. LR Without Class Balancing & RF Response code model to reduced the log loss less than 1

- Defined ngram range while representing Text Features by Bag of Words. Both the model performance are not helping us to achieve our objective of reducing log loss beyond 1.

- Tried steps like combining response coding and one hot coding of gene, variation and text feature , some model performed well but some models could not achived the log loass less than 1 but reached the log loss around 1

A4.Trying the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

1.1 Univariate Analysis on `Gene Feature - tfidfVectorizer(unigrams, bi-grams)`

In [221]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(ngram_range=(1, 2))
train_gene_feature_bigram_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene']
test_gene_feature_bigram_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_bigram_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

print("Shape of train_gene_feature_bigram_onehotCoding:", train_gene_feature_bigram_onel
print("Shape of test_gene_feature_bigram_onehotCoding :", test_gene_feature_bigram_oneh
print("Shape of cv_gene_feature_bigram_onehotCoding   :", cv_gene_feature_bigram_onehot
print("Gene vectorizer feature names :", gene_vectorizer.get_feature_names()[:10])
```

```
Shape of train_gene_feature_bigram_onehotCoding: (2124, 229)
Shape of test_gene_feature_bigram_onehotCoding : (665, 229)
Shape of cv_gene_feature_bigram_onehotCoding   : (532, 229)
Gene vectorizer feature names : ['abl1', 'acvr1', 'ago2', 'akt1', 'akt2',
'akt3', 'alk', 'apc', 'ar', 'araf']
```

1.2 Univariate Analysis on `Text Feature - CountVectorizer( bi-grams)`

In [222]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train d
text_vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2))
train_text_feature_bigram_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT']
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it
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 : 206620
```

In [223]:

```
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 [238]:

```
# don't forget to normalize every feature
train_text_feature_bigram_onehotCoding = normalize.fit_transform(train_text_feature_big

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

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

In [239]:

```
print("Total number of unique words in train data :", len(text_vectorizer.get_feature_n
```

```
Total number of unique words in train data : 206620
```

In [240]:

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

In [241]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur[:10]))
```

```
Counter({245.74059129288608: 1, 171.49202881803927: 1, 141.74669804071084:
1, 129.1955630366883: 1, 112.07263050435022: 1, 90.60738667273849: 1, 89.2
675160798364: 1, 87.43513579019121: 1, 87.09878394271173: 1, 78.0292266997
6549: 1})
```

In [242]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=10, ngram_range=(2,2))
    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 [243]:

```
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 da
```

```
99.895 % of word of test data appeared in train data
99.968 % of word of Cross Validation appeared in train data
```

In [244]:

```
#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 e
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_
    plot_confusion_matrix(test_y, pred_y)
```

In [245]:

```
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 [246]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer(ngram_range=(2,2))
    var_count_vec  = TfidfVectorizer(ngram_range=(2,2))
    text_count_vec = TfidfVectorizer(ngram_range=(2,2))

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

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

Stacking the two types of features (gene and Text)

In [ ]:

```
# 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_x_bigram_onehotCoding = hstack((train_gene_feature_bigram_onehotCoding,train_text_
test_x_bigram_onehotCoding = hstack((test_gene_feature_bigram_onehotCoding,test_text_fe
cv_x_bigram_onehotCoding = hstack((cv_gene_feature_bigram_onehotCoding,cv_text_feature_

train_y = np.array(list(train_df['Class']))
test_y = np.array(list(test_df['Class']))
cv_y = np.array(list(cv_df['Class']))
```

In [248]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_bigram_o
print("(number of data points * number of features) in test data = ", test_x_bigram_onel
print("(number of data points * number of features) in cross validation data =", cv_x_b
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 20684
9)
(number of data points * number of features) in test data =  (665, 206849)
(number of data points * number of features) in cross validation data = (5
32, 206849)
```

2. Logistic Regression - With Class balancing Tfidf(bigrams) - Gene with Text

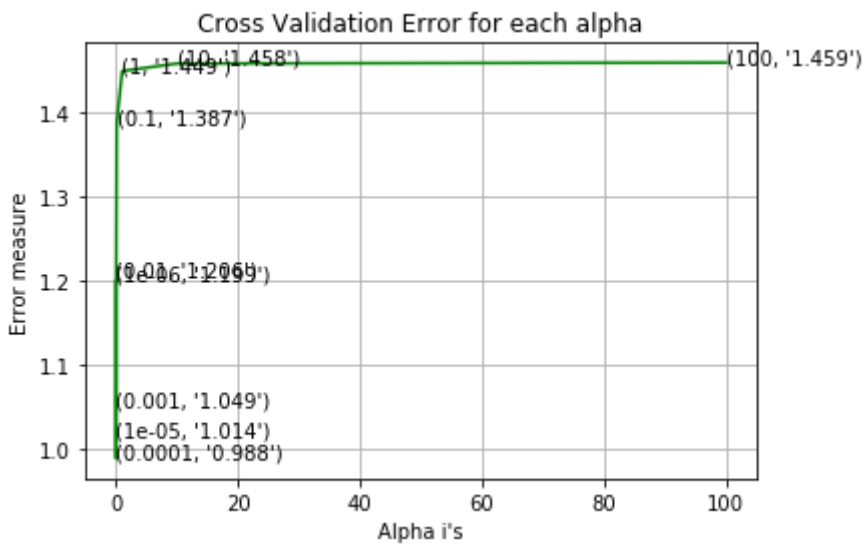
2.1.1 Hyperparameter Tuning

In [249]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# -----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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', ran
    clf.fit(train_x_bigram_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_bigram_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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', los
clf.fit(train_x_bigram_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_bigram_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.1994273476071848
for alpha = 1e-05
Log Loss : 1.0144787954965129
for alpha = 0.0001
Log Loss : 0.9879423220232212
for alpha = 0.001
Log Loss : 1.0493907334912265
for alpha = 0.01
Log Loss : 1.2063751446367699
for alpha = 0.1
Log Loss : 1.3873734584529869
for alpha = 1
Log Loss : 1.4487291159095523
for alpha = 10
Log Loss : 1.457671190139001
for alpha = 100
Log Loss : 1.458755775236865
```



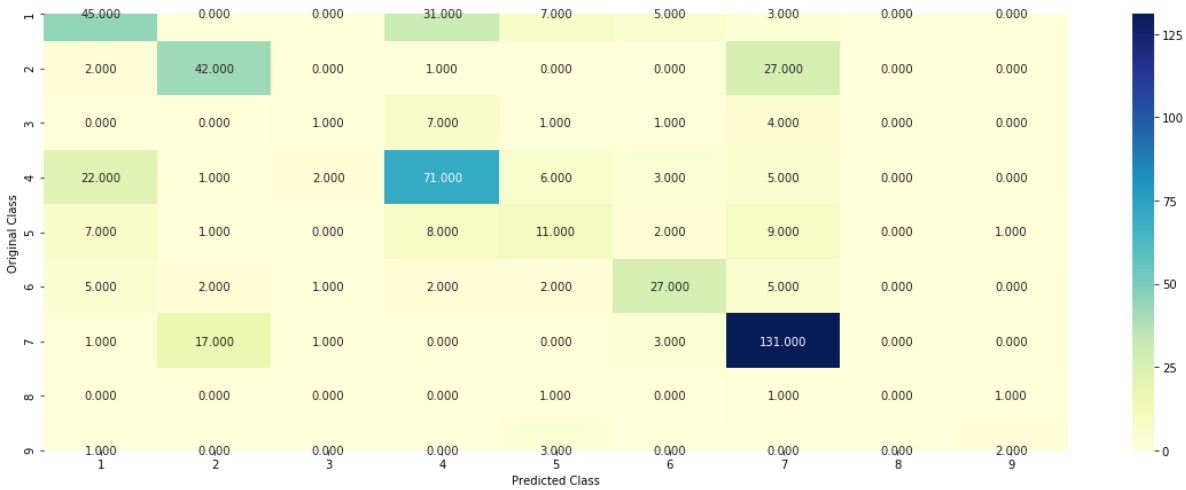

```
For values of best alpha =  0.0001 The train log loss is: 0.57579011149084
69
For values of best alpha =  0.0001 The cross validation log loss is: 0.987
9423220232212
For values of best alpha =  0.0001 The test log loss is: 1.044229980397962
4
```

2.1.2 testing the model with best hyperparameter

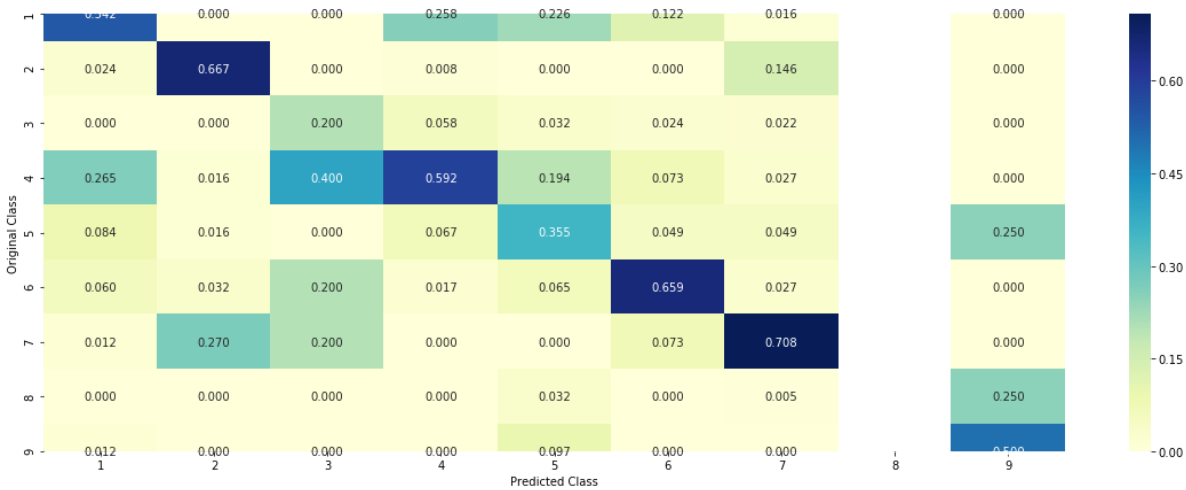
In [253]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
predict_and_plot_confusion_matrix(train_x_bigram_onehotCoding, train_y, cv_x_bigram_one
```

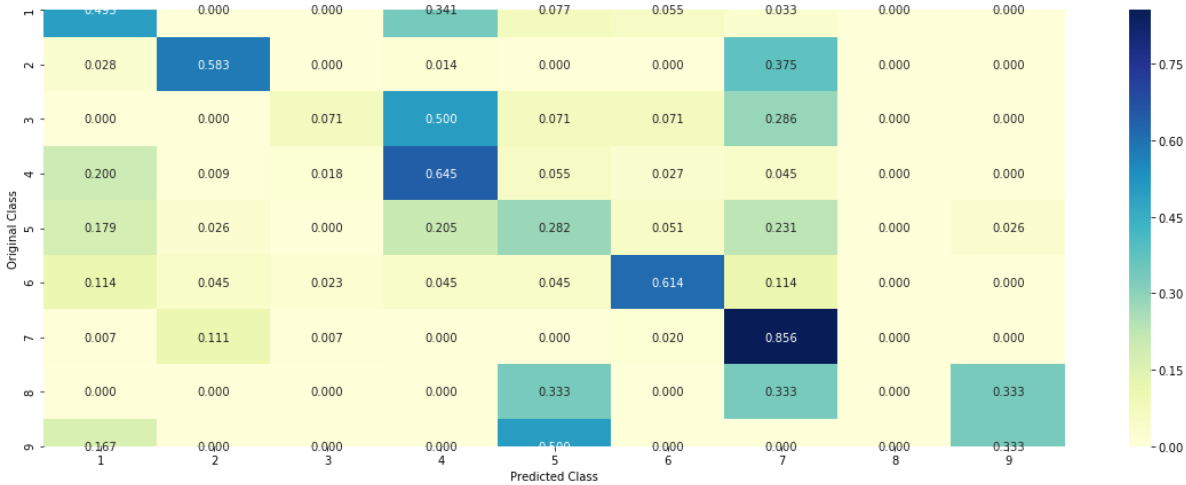
```
Log loss : 0.9879423220232212
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```




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



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



2.1.3 Feature importance correctly classified

In [254]:

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

In [255]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', los
clf.fit(train_x_bigram_onehotCoding,train_y)
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_bigram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_bigram_on
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.868  0.0286 0.0081 0.0233 0.0139 0.0141
0.0366 0.0036 0.0038]]
Actual Class : 7
--------------------------------------------------
```

2.1.4 Feature importance incorrectly classified

In [256]:

```
test_point_index = 361
no_feature = 100
predicted_cls = sig_clf.predict(test_x_bigram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_bigram_on
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.3875 0.484  0.0049 0.0713 0.0077 0.0075
0.0324 0.0026 0.0021]]
Actual Class : 2
--------------------------------------------------
```

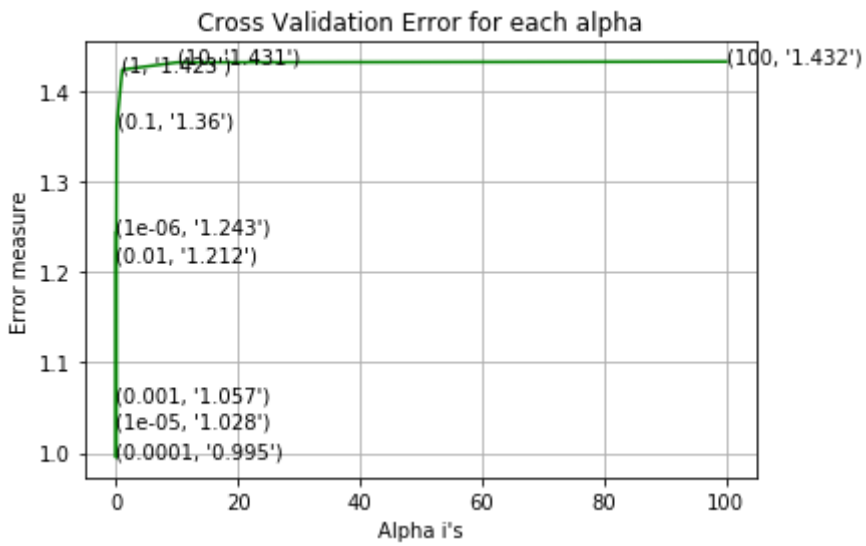
2.2 Logistic Regression without class balancing - Tfidf- Gene, Text

In [258]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# -----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
#
# 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(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_bigram_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_bigram_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
    # to avoid rounding error while multiplying probabilites we use log-probability est
    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_bigram_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_bigram_onehotCoding, train_y)
```

```
predict_y = sig_clf.predict_proba(train_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.2433759797598878
for alpha = 1e-05
Log Loss : 1.0281667027597325
for alpha = 0.0001
Log Loss : 0.9949629404316109
for alpha = 0.001
Log Loss : 1.05673352539388
for alpha = 0.01
Log Loss : 1.212334924499428
for alpha = 0.1
Log Loss : 1.3601482915842733
for alpha = 1
Log Loss : 1.4230829676425123
for alpha = 10
Log Loss : 1.4310942225706862
for alpha = 100
Log Loss : 1.4319991149766391
```



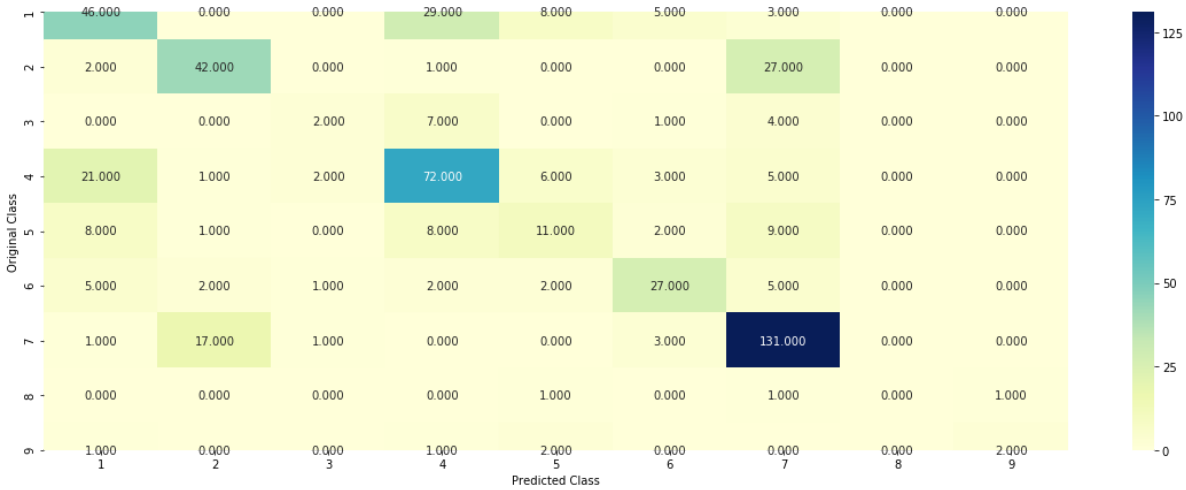
```
For values of best alpha =  0.0001 The train log loss is: 0.567313359174
8932
For values of best alpha =  0.0001 The cross validation log loss is: 0.9
949629404316109
For values of best alpha =  0.0001 The test log loss is: 1.0450092774084
95
```

2.2.2 testing the model with best hyperparameter

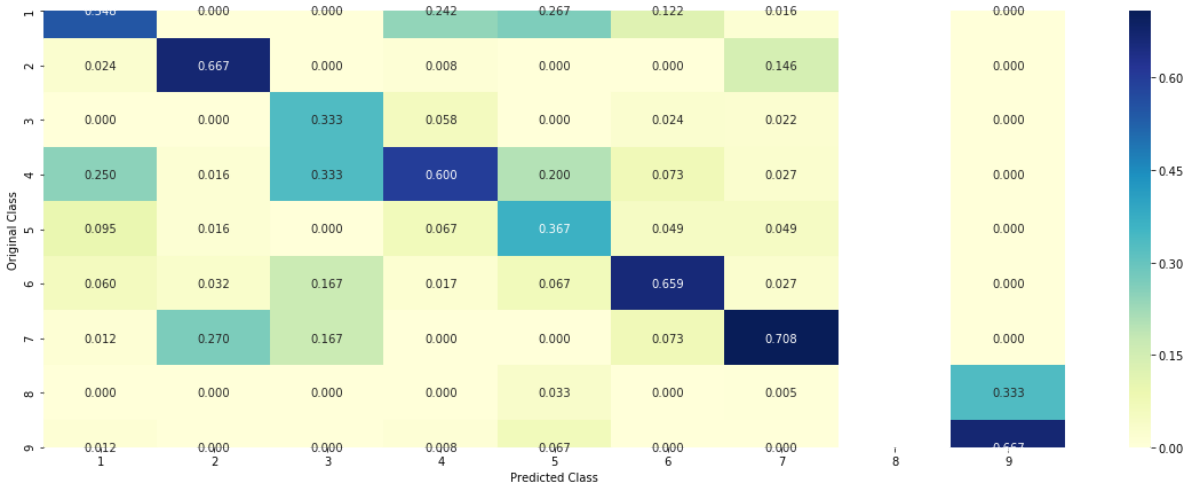
In [259]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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 D
# predict(X)    Predict class labels for samples in X.

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

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

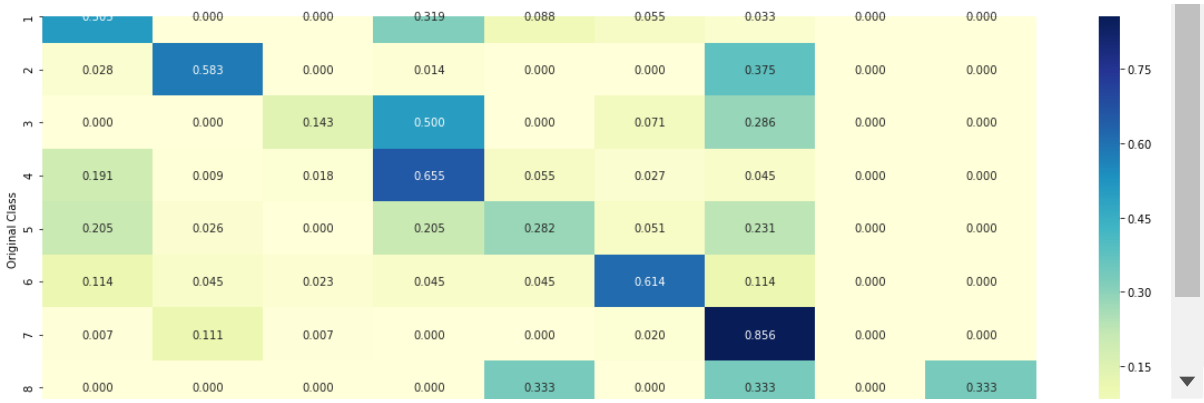
```
Log loss : 0.9949629404316109
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```




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




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



2.2.3 Feature importance correctly classified

In [260]:

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

In [262]:

```
# from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_bigram_onehotCoding,train_y)
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_bigram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_bigram_on
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8685 0.0292 0.0075 0.0222 0.0146 0.0148
0.0364 0.0034 0.0035]]
Actual Class : 7
--------------------------------------------------
```

2.2.4 Feature importance incorrectly classified

In [263]:

```
test_point_index = 361
no_feature = 100
predicted_cls = sig_clf.predict(test_x_bigram_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_bigram_on
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.3947 0.4812 0.0046 0.0707 0.0083 0.0078
0.0283 0.0025 0.0019]]
Actual Class : 2
--------------------------------------------------
```

Conclusion ::

In [265]:

```
p = PrettyTable()

p.field_names=["Model Name ", "Vectorizer", "Train Loss", "CV Loss", "Test loss", " Mis
p.add_row(["LR with class balance", "TfidfVectorizer-bigrams", 0.575, 0.987, 1.044, 0.3
p.add_row(["LR without class balance", "TfidfVectorizer-bigrmas", 0.567, 0.994, 1.045, 
print(p)
```

```
+-------------------------+-------------------------+------------+-------
--+-----------+------------------------+
|       Model Name        |        Vectorizer       | Train Loss | CV Los
s | Test loss |  Mis Classified points  |
+-------------------------+-------------------------+------------+-------
--+-----------+------------------------+
|  LR with class balance  | TfidfVectorizer-bigrams |   0.575    |  0.987
|   1.044   |          0.379          |
| LR without class balance | TfidfVectorizer-bigrmas |   0.567    |  0.994
|   1.045   |          0.374          |
+-------------------------+-------------------------+------------+-------
--+-----------+------------------------+
```

- **Our main objective is to minimize the loss to lees than 1, I achieved that by trying feature engineering with tfidfvectorizer(unigrams, bigrams) and combining the two stable features of Gene and text**

Sign Off RAMESH BATTU (https://www.linkedin.com/in/rameshbattuai/)