Personalized cancer diagnosis

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)

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

Trying the above mentioned using following:

1. PART A: Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. PART B: Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values

3. PART C: Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. PART D: 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

3. Exploratory Data Analysis

In [1]:

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

from mlxtend.classifier import StackingClassifier

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

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"],skipr
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 377.02184900000003 seconds
```

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

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

In [8]:

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

In [9]:

```
result[result['ID']==1109]
```

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

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

In [16]:

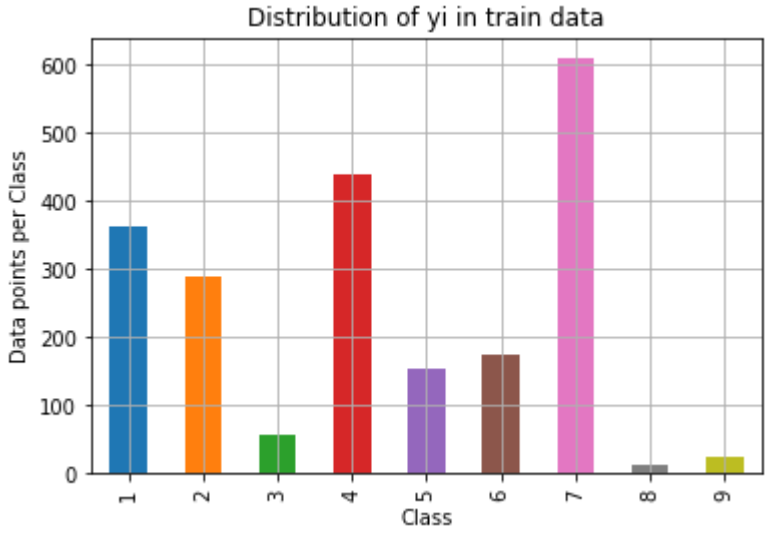
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

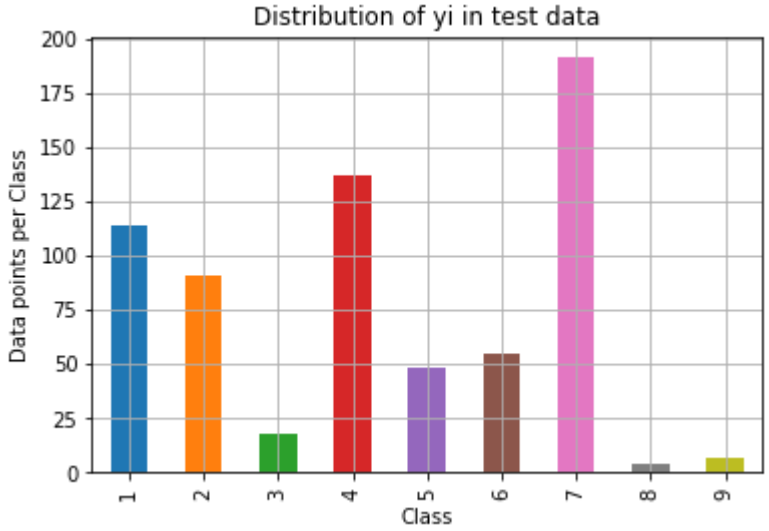
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [0]:

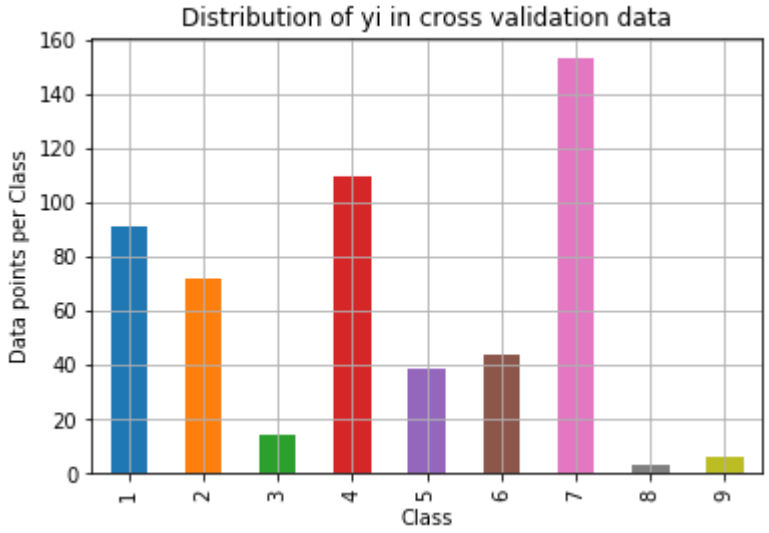
```
# it returns a dict, keys as class labels and values as the number of data points in that c
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors = '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.

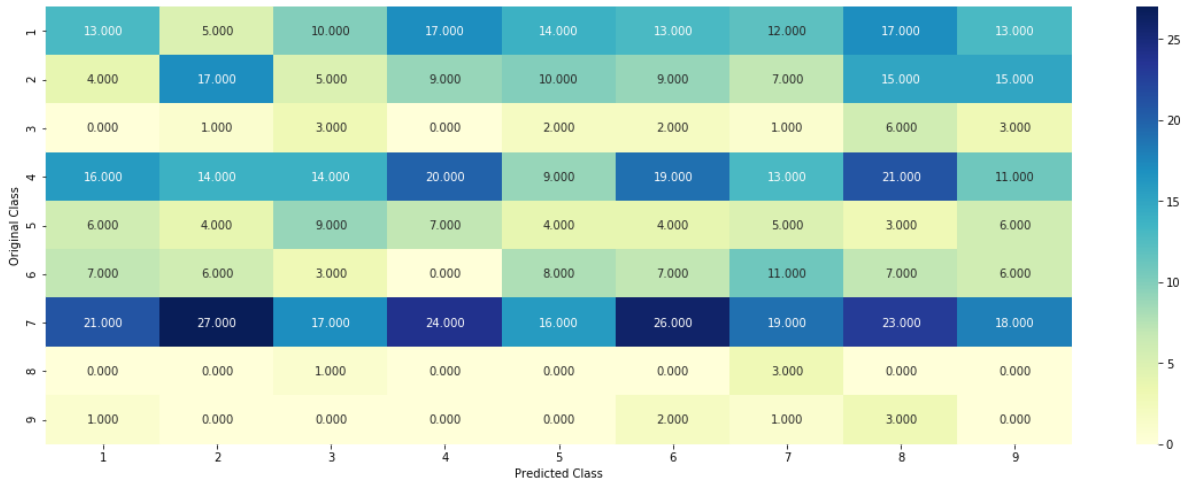
In [11]:

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

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

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

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

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

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

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

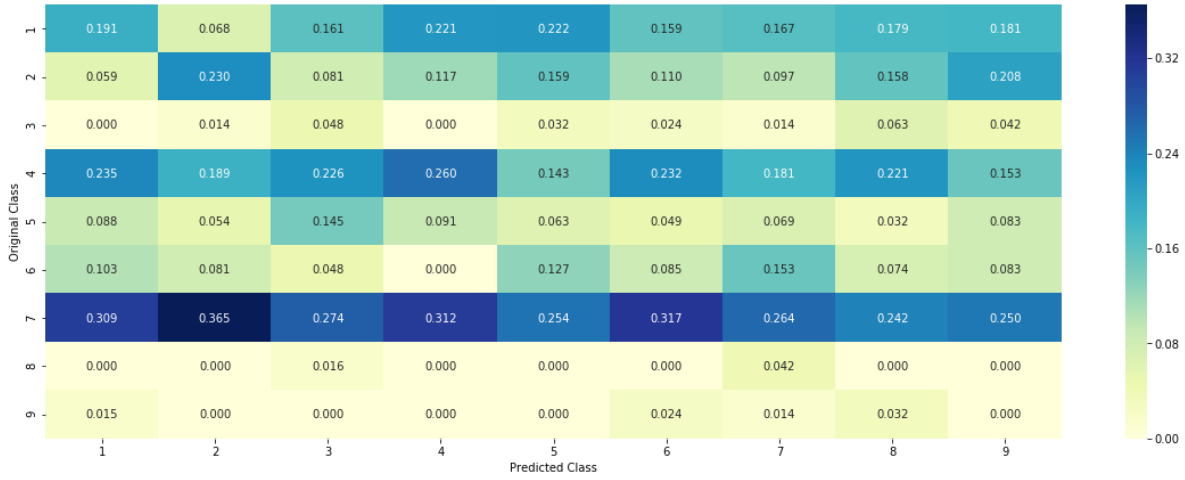
In [0]:

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

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


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-1

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

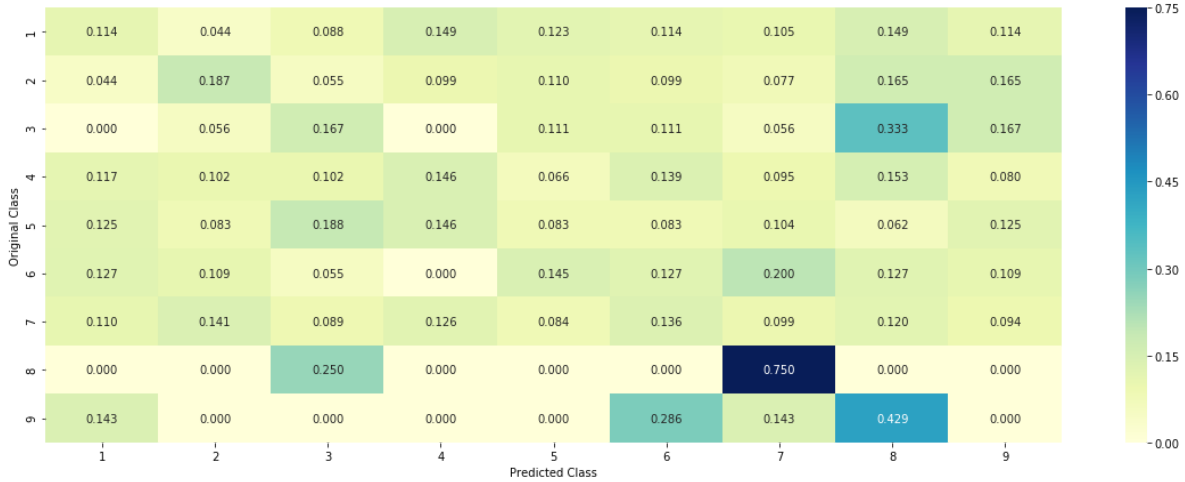
```
Log loss on Cross Validation Data using Random Model 2.536598785706848
Log loss on Test Data using Random Model 2.501572555849742
-------------------- Confusion matrix --------------------
```




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




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



3.3 Univariate Analysis

In [12]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data da
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alp
# 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/var
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
```

```
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular f
            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, 0.068181818181818177, 0.1363
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the
    # 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 [0]:

```
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     164
TP53       90
EGFR       88
PTEN       82
BRCA2      78
KIT        67
BRAF       51
ALK        48
ERBB2      46
PIK3CA     37
Name: Gene, dtype: int64
```

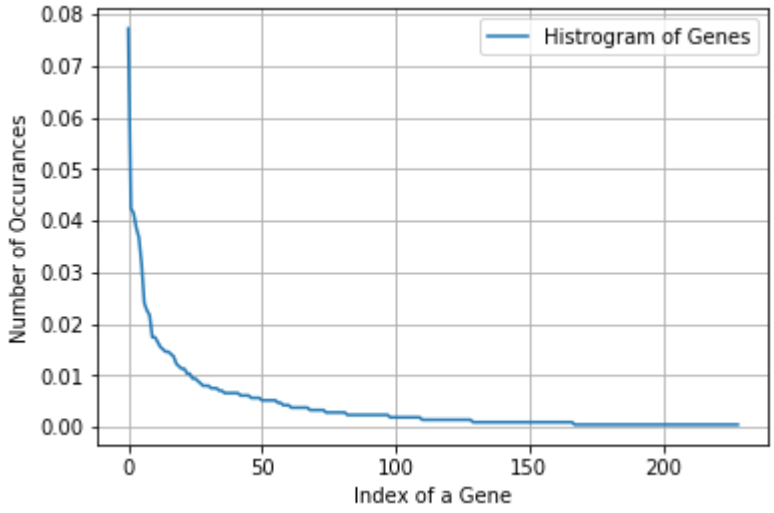
In [0]:

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

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

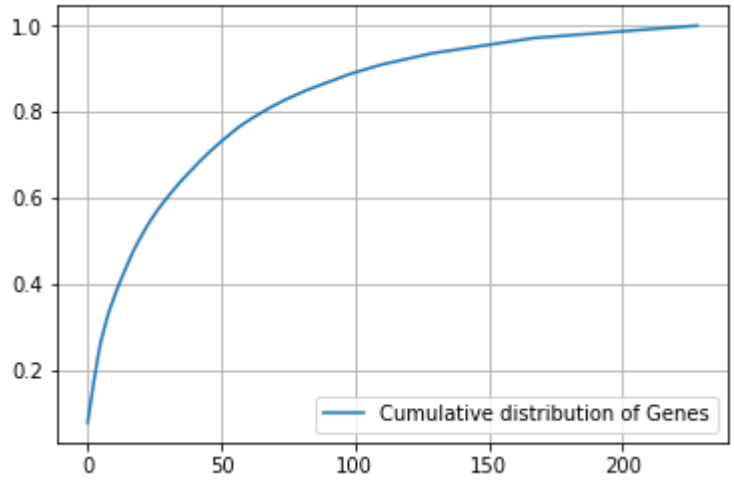
In [0]:

```
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 [0]:

```
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/)

1. One hot Encoding
2. Response coding

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

In [13]:

```
#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 [15]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method.
```

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

In [14]:

```
# 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 [17]:

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

Out[17]:

```
30       TERT
624     FBXW7
2085     AGO2
773     ERBB3
2452    BRCA1
Name: Gene, dtype: object
```

In [ ]:

```
gene_vectorizer.get_feature_names()
```

In [19]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 240)
```

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

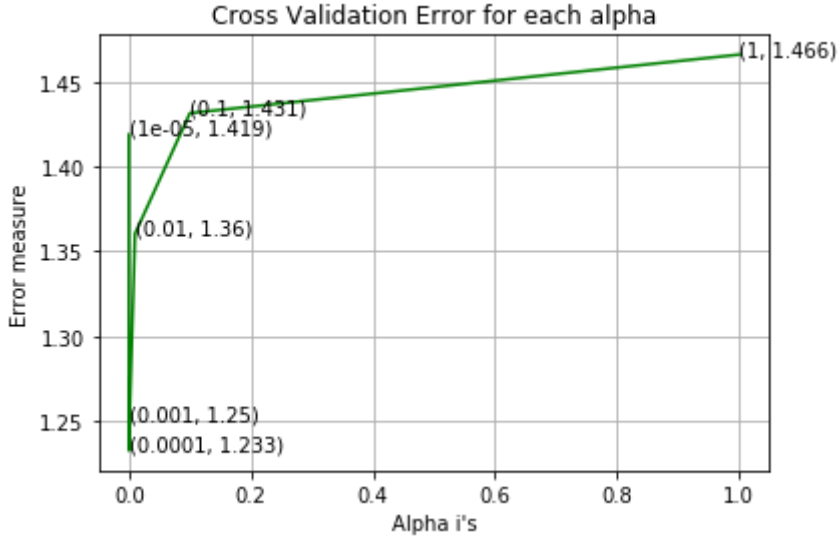
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [0]:

```
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/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# 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 Desce
# predict(X)    Predict class labels for samples in X.

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


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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.418841767162939
For values of alpha =  0.0001 The log loss is: 1.2325868001617826
For values of alpha =  0.001 The log loss is: 1.2503129272158073
For values of alpha =  0.01 The log loss is: 1.360379976757511
```

```
For values of alpha =  0.1 The log loss is: 1.4314392521126913
For values of alpha =  1 The log loss is: 1.4659143358159061
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0425604300119806
For values of best alpha =  0.0001 The cross validation log loss is: 1.23258
68001617826
For values of best alpha =  0.0001 The test log loss is: 1.200905436534172
```

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 [0]:

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

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/t
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage
```

```
Q6. How many data points in Test and CV datasets are covered by the  229  ge
nes in train dataset?
Ans
1. In test data 643 out of 665 : 96.69172932330827
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 [0]:

```
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 : 1924
Truncating_Mutations         59
Deletion                     49
Amplification                47
Fusions                      23
E17K                          3
Overexpression                3
Q22K                          2
Promoter_Hypermethylation     2
G13D                          2
T73I                          2
Name: Variation, dtype: int64
```

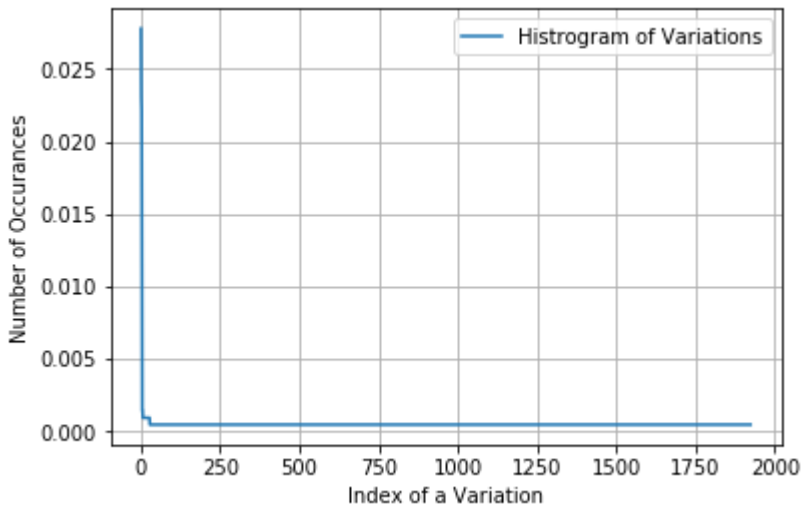
In [0]:

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

```
Ans: There are 1924 different categories of variations in the train data, an
d they are distibuted as follows
```

In [0]:

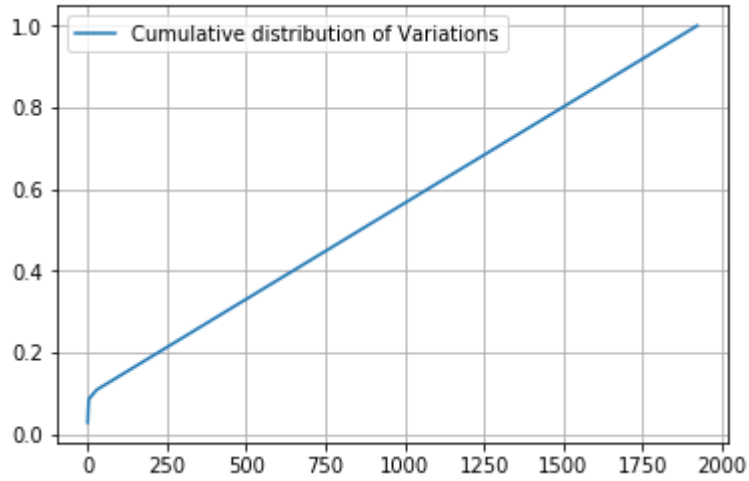
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [0]:

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

```
[0.02777778 0.05084746 0.07297552 ... 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/)

1. One hot Encoding
2. Response coding

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

In [15]:

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

In [17]:

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

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

In [16]:

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

In [19]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-ho
t encoding method. The shape of Variation feature: (2124, 1969)
```

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

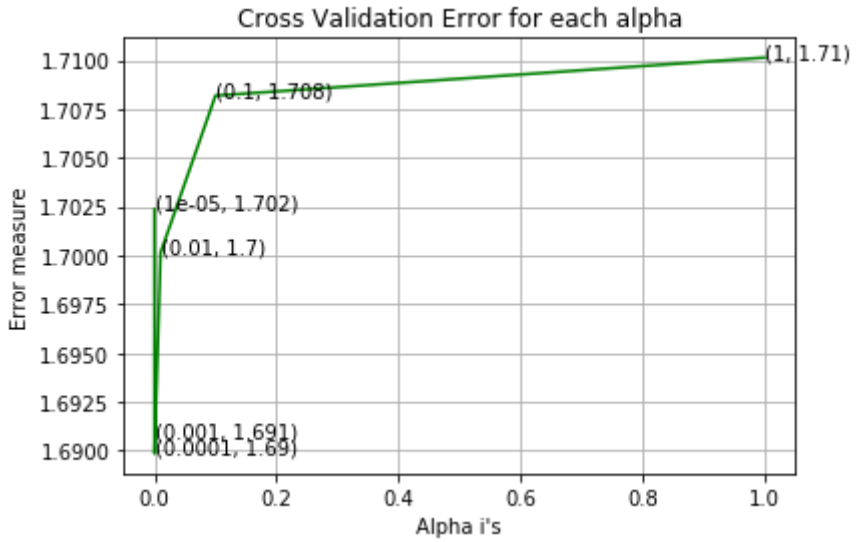
Let's build a model just like the earlier!

In [0]:

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

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

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


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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7023621747765858
For values of alpha =  0.0001 The log loss is: 1.689842322110077
For values of alpha =  0.001 The log loss is: 1.6907130717518253
```

```
For values of alpha =  0.01 The log loss is: 1.7001345396142153
For values of alpha =  0.1 The log loss is: 1.7081826775989355
For values of alpha =  1 The log loss is: 1.710131025593559
```




```
For values of best alpha =  0.0001 The train log loss is: 0.8255455900343496
For values of best alpha =  0.0001 The cross validation log loss is: 1.68984
2322110077
For values of best alpha =  0.0001 The test log loss is: 1.7362385768757977
```

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 [0]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " gene
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/t
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage
```

```
Q12. How many data points are covered by total  1924  genes in test and cros
s validation data sets?
Ans
1. In test data 64 out of 665 : 9.624060150375941
2. In cross validation data 56 out of  532 : 10.526315789473683
```

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 [17]:

```
# 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 [18]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].s
            row_index += 1
    return text_feature_responseCoding
```

In [22]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(ngram_range = (1,2), max_features = 1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number o
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 occu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

In [23]:

```
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 [24]:

```
#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 [25]:

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_re
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_response
```

In [26]:

```
# 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 [27]:

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

In [ ]:

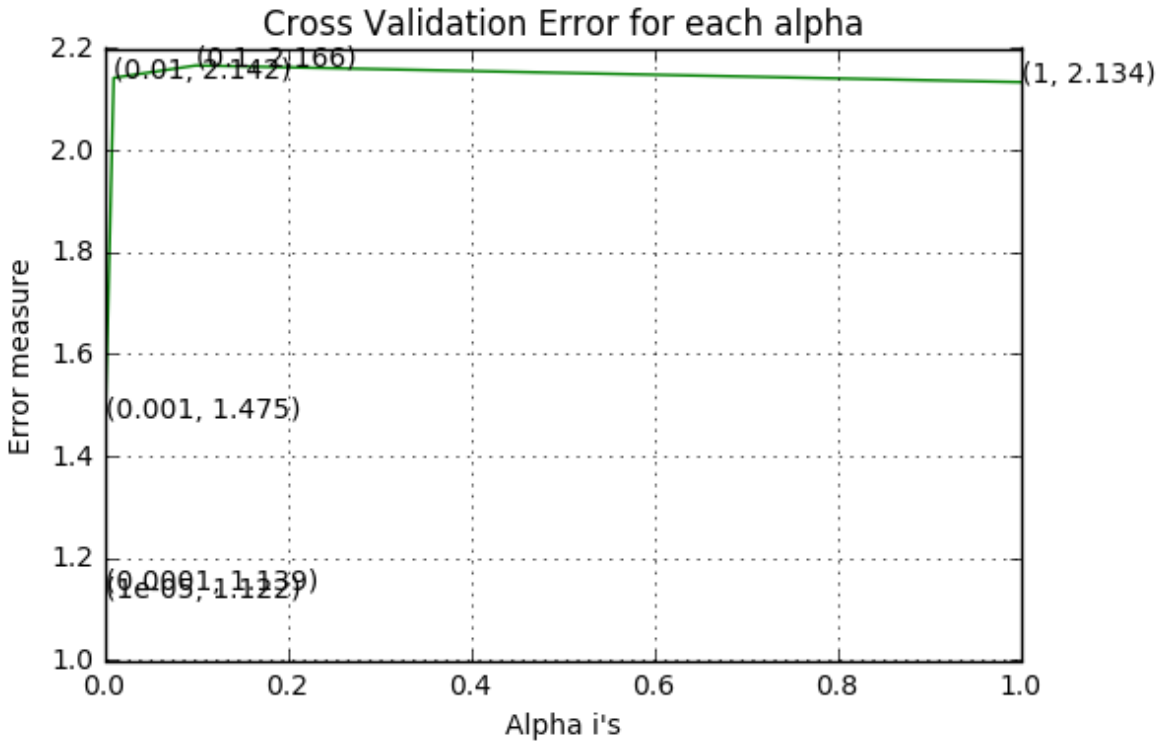
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

In [29]:

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

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

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

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, n_j
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.1223631352768793
For values of alpha =  0.0001 The log loss is: 1.1388416513572448
For values of alpha =  0.001 The log loss is: 1.4747792850710966
```

```
For values of alpha =  0.01 The log loss is: 2.1418942425780827
For values of alpha =  0.1 The log loss is: 2.1662038994011676
For values of alpha =  1 The log loss is: 2.1335012689159973
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7993394393775066
For values of best alpha =  1e-05 The cross validation log loss is: 1.122363
1352768793
For values of best alpha =  1e-05 The test log loss is: 1.171112288364703
```

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

Ans. Yes, it seems like!

In [0]:

```
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 [0]:

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

```
97.125 % of word of test data appeared in train data
98.056 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

PART A: Applying all the models with tf-idf features

PART B: Instead of using all the words in the dataset, using only the top 1000 words based of tf-idf values

In [64]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to each
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.sh
    plot_confusion_matrix(test_y, pred_y)
```

In [65]:

```
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 [66]:

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

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

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

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

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

Stacking the three types of features

In [33]:

```
# 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_featu
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehot

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variatio
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_f
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_respo
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_response
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding
```

In [34]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3200)
(number of data points * number of features) in test data =  (665, 3200)
(number of data points * number of features) in cross validation data = (53
2, 3200)
```

In [35]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCodi
print("(number of data points * number of features) in test data = ", test_x_responseCoding
print("(number of data points * number of features) in cross validation data =", cv_x_respo
```

```
 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 = (53
2, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

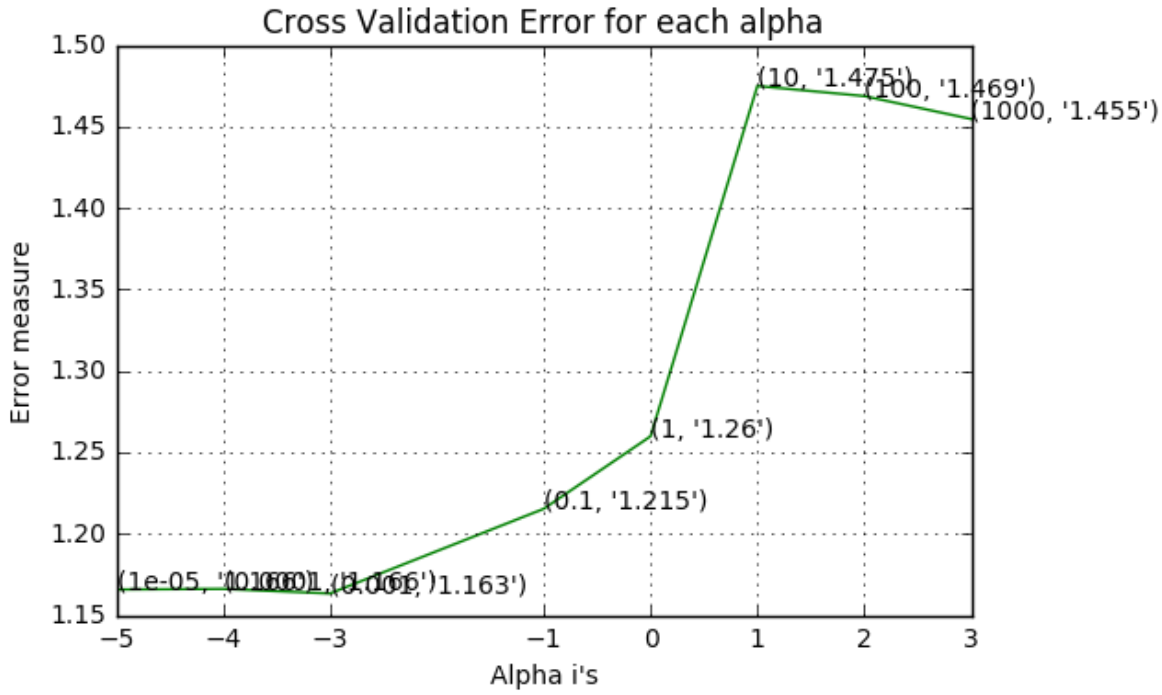
4.1.1.1. Hyper parameter tuning

In [36]:

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

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


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


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

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


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

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

```
for alpha = 1e-05
Log Loss : 1.1657321749849234
for alpha = 0.0001
Log Loss : 1.1661242512541043
for alpha = 0.001
Log Loss : 1.1632927921722251
for alpha = 0.1
Log Loss : 1.2152460145121646
for alpha = 1
Log Loss : 1.259778194615194
for alpha = 10
Log Loss : 1.4748989529867178
for alpha = 100
Log Loss : 1.4686958392805554
for alpha = 1000
Log Loss : 1.4546063654716792
```



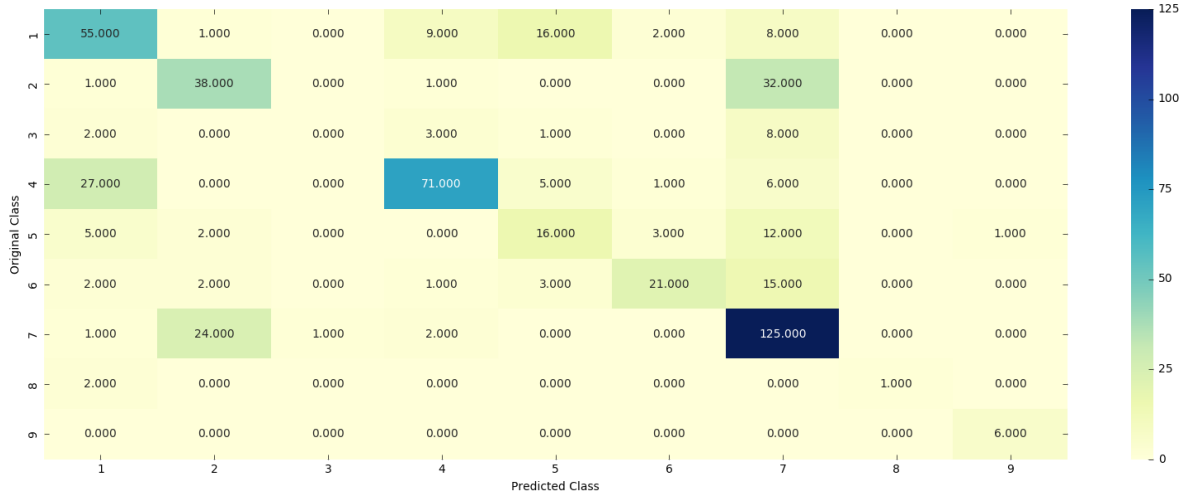

```
For values of best alpha =  0.001 The train log loss is: 0.5150086990710969
For values of best alpha =  0.001 The cross validation log loss is: 1.163292
7921722251
For values of best alpha =  0.001 The test log loss is: 1.191919676927422
```

4.1.1.2. Testing the model with best hyper paramters

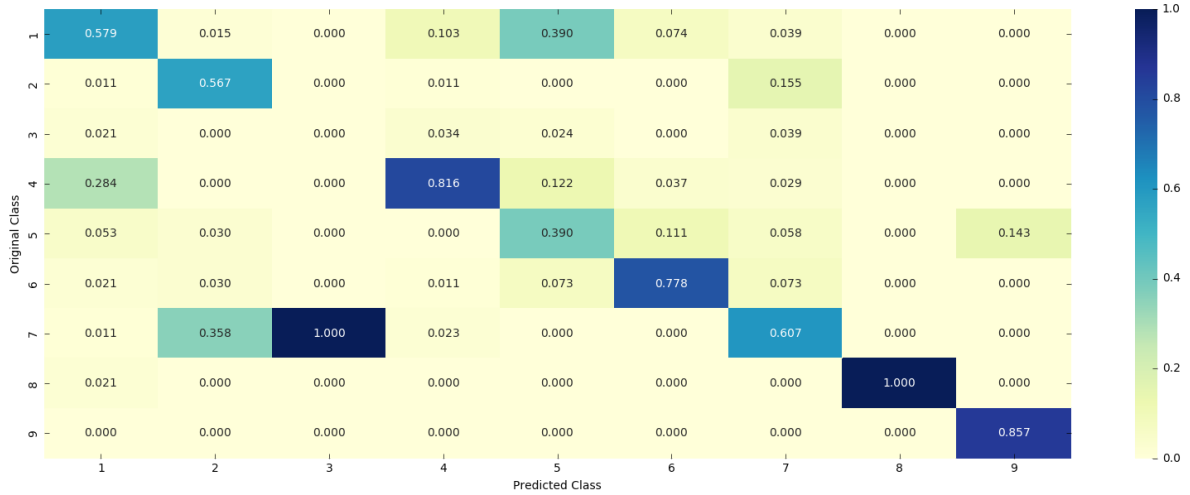
In [39]:

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

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


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

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

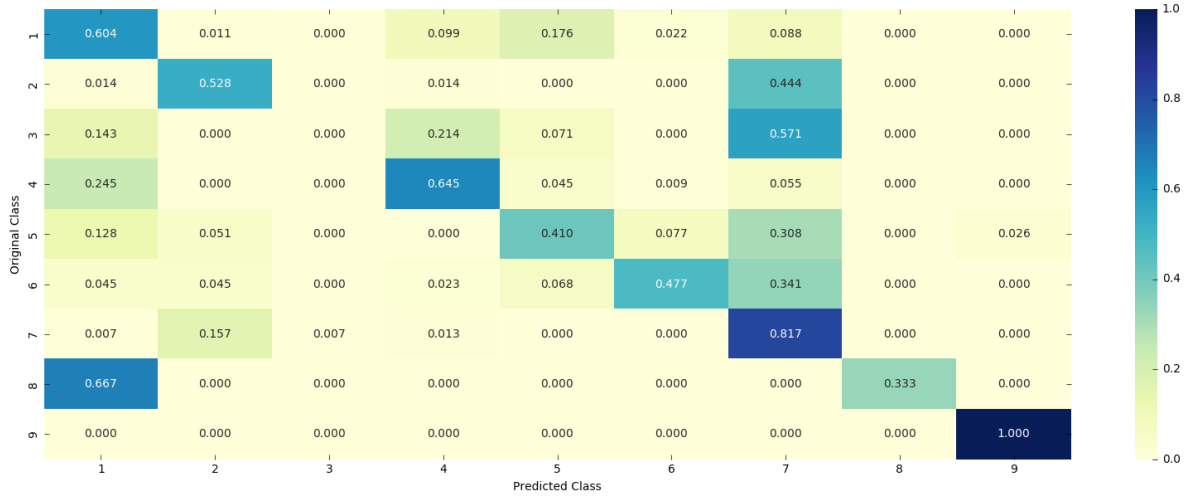
```
Log Loss : 1.1632927921722251
Number of missclassified point : 0.37406015037593987
-------------------- Confusion matrix --------------------
```




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



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



4.1.1.3. Query point 1

In [40]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0564 0.0417 0.0098 0.0667 0.0319 0.0266
0.7613 0.0035 0.0022]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Query point 2

In [41]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0563 0.0406 0.0098 0.0666 0.0319 0.0266
0.7627 0.0035 0.0022]]
Actual Class : 7
--------------------------------------------------
18 Text feature [079] present in test data point [True]
45 Text feature [104] present in test data point [True]
69 Text feature [025] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.2. K Nearest Neighbour Classification

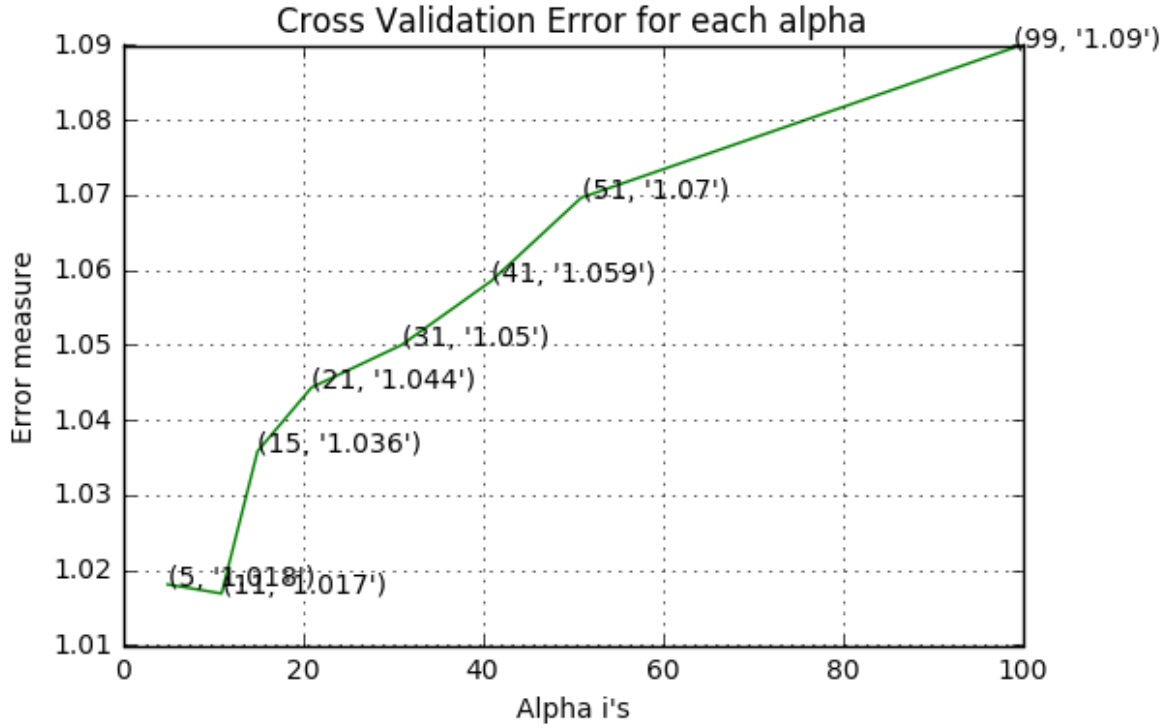
4.2.1. Hyper parameter tuning

In [42]:

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


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


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

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

```
for alpha = 5
Log Loss : 1.0181056698844584
for alpha = 11
Log Loss : 1.016889296386972
for alpha = 15
Log Loss : 1.0358531391905432
for alpha = 21
Log Loss : 1.0443489743595016
for alpha = 31
Log Loss : 1.0500128413668648
for alpha = 41
Log Loss : 1.0585663223028579
for alpha = 51
Log Loss : 1.0696624793230598
for alpha = 99
Log Loss : 1.0896046025825497
```



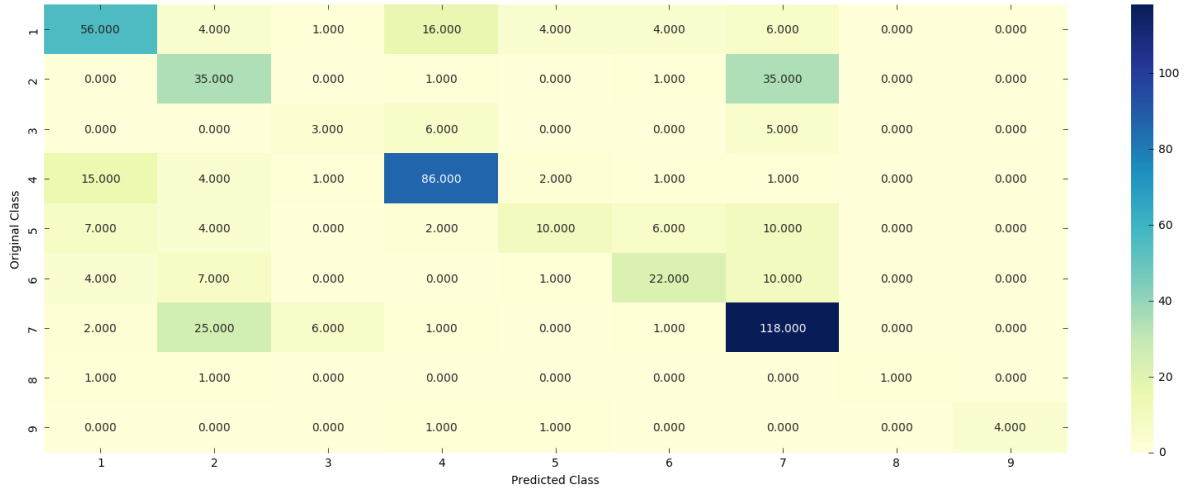

```
For values of best alpha =  11 The train log loss is: 0.6380683951739716
For values of best alpha =  11 The cross validation log loss is: 1.016889296
386972
For values of best alpha =  11 The test log loss is: 1.0594131806710874
```

4.2.2. Testing the model with best hyper paramters

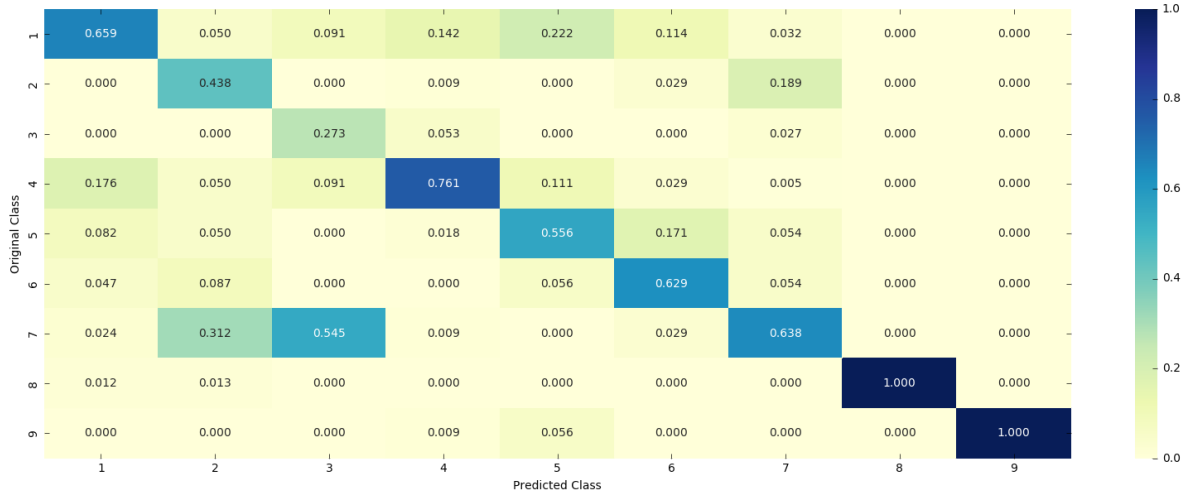
In [43]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/genera
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=
# 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-nea
#-----------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha], n_jobs = -1)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_
```

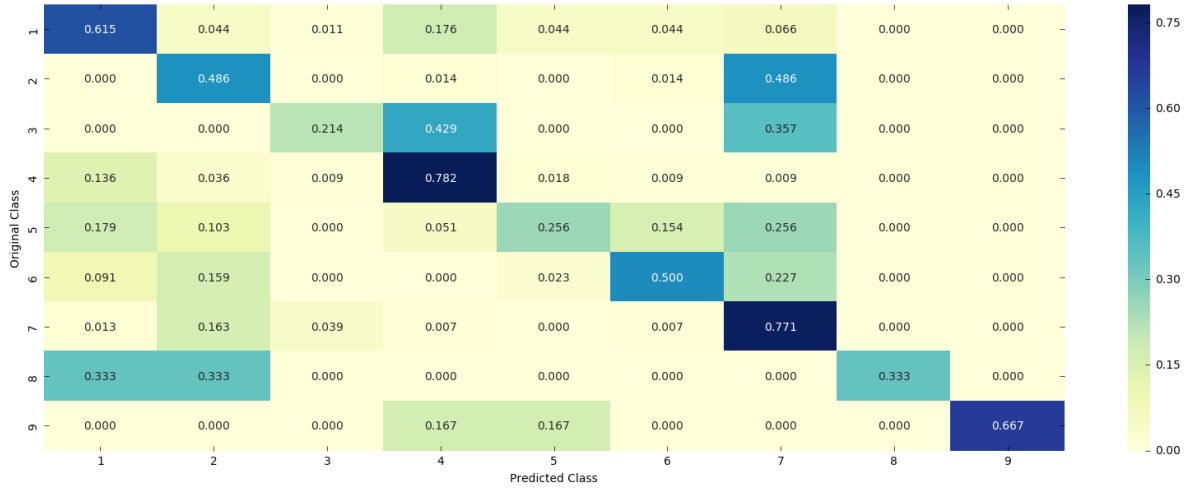
```
Log loss : 1.016889296386972
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [44]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha], n_jobs = -1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 7
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 7 4
3 7 1 7 7 7]
Fequency of nearest points : Counter({7: 8, 1: 1, 3: 1, 4: 1})
```

4.2.4. Sample Query Point-2

In [45]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha], n_jobs = -1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belo
ngs to classes [7 7 7 2 7 7 7 7 7 7 7]
Fequency of nearest points : Counter({7: 10, 2: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

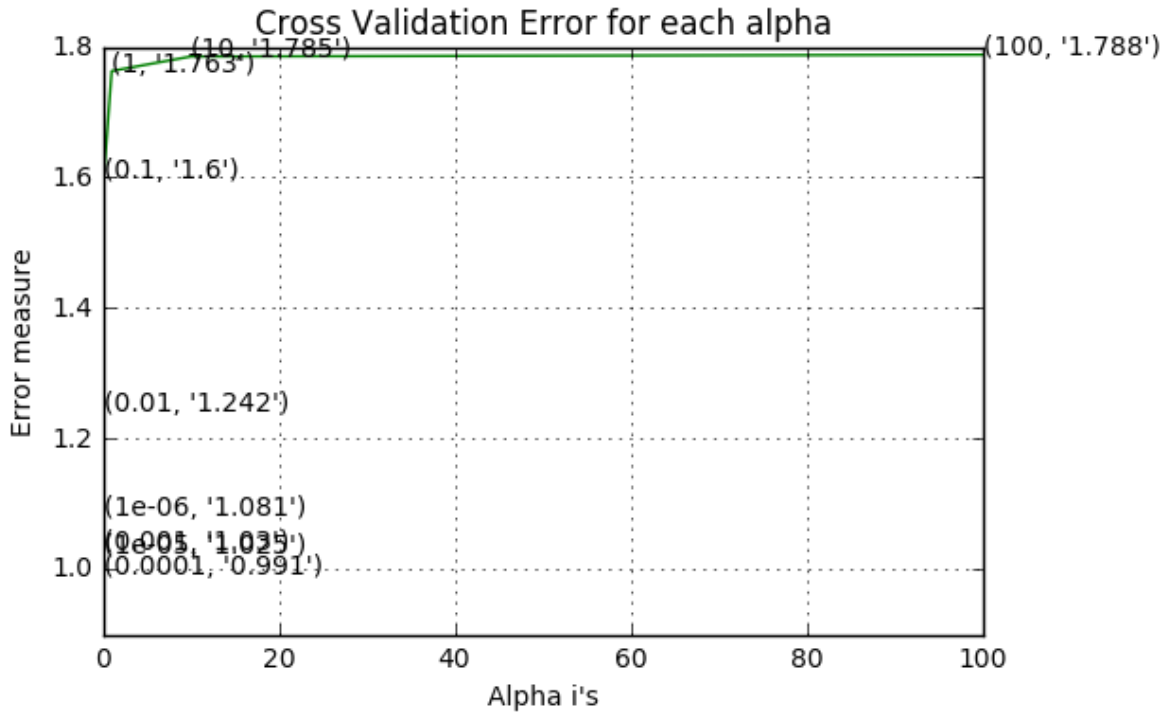
4.3.1.1. Hyper paramter tuning

In [46]:

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

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


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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.0814977263639225
for alpha = 1e-05
Log Loss : 1.0246901307530114
for alpha = 0.0001
Log Loss : 0.9913869905132415
for alpha = 0.001
Log Loss : 1.0296317141555387
for alpha = 0.01
Log Loss : 1.2416686083995605
for alpha = 0.1
Log Loss : 1.5998994095245929
for alpha = 1
Log Loss : 1.7629017446280562
for alpha = 10
Log Loss : 1.7851454956871151
for alpha = 100
Log Loss : 1.7878501081508689
```



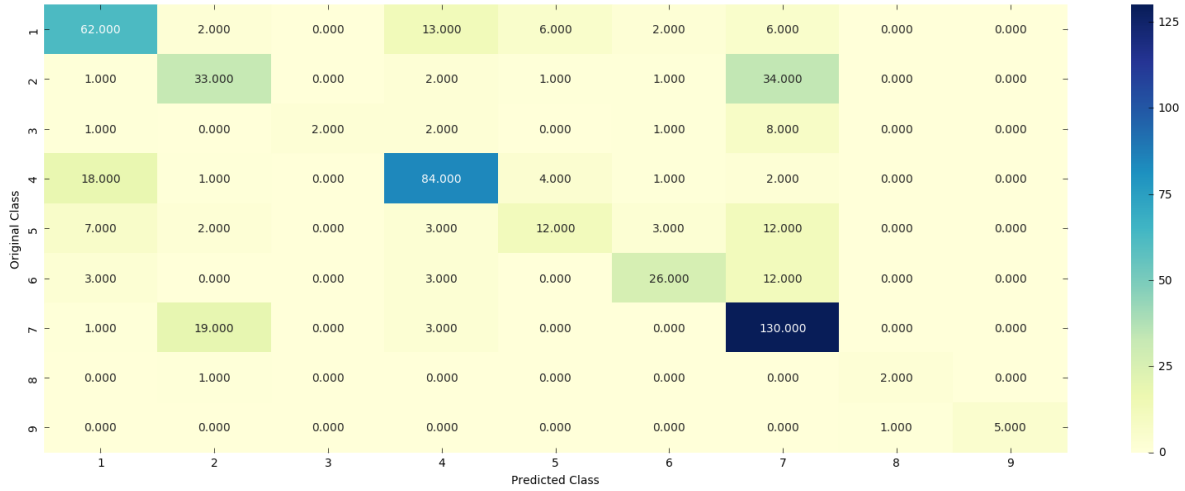

```
For values of best alpha =  0.0001 The train log loss is: 0.4433209963563271
For values of best alpha =  0.0001 The cross validation log loss is: 0.99138
69905132415
For values of best alpha =  0.0001 The test log loss is: 1.0441205356056804
```

4.3.1.2. Testing the model with best hyper paramters

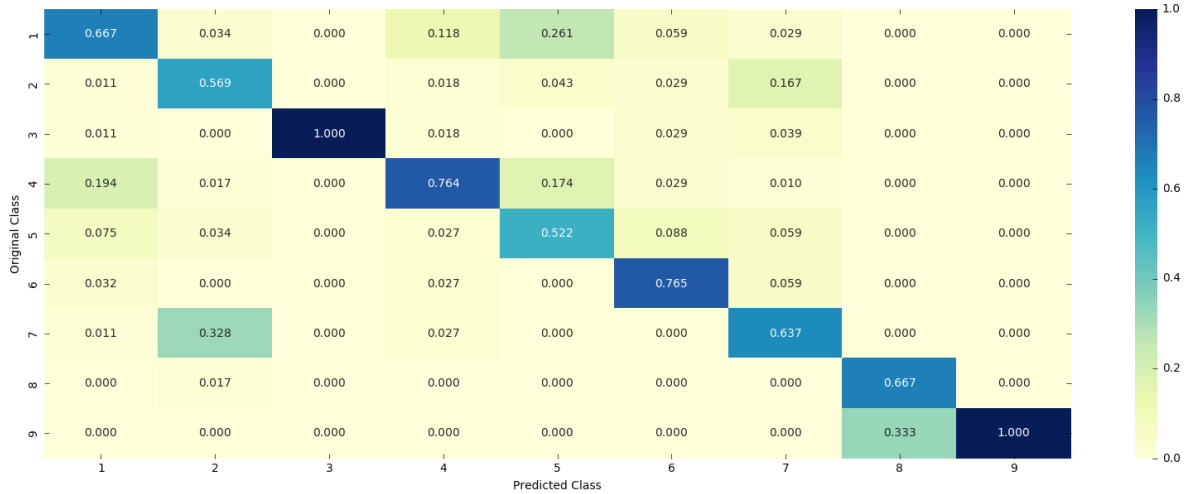
In [47]:

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

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

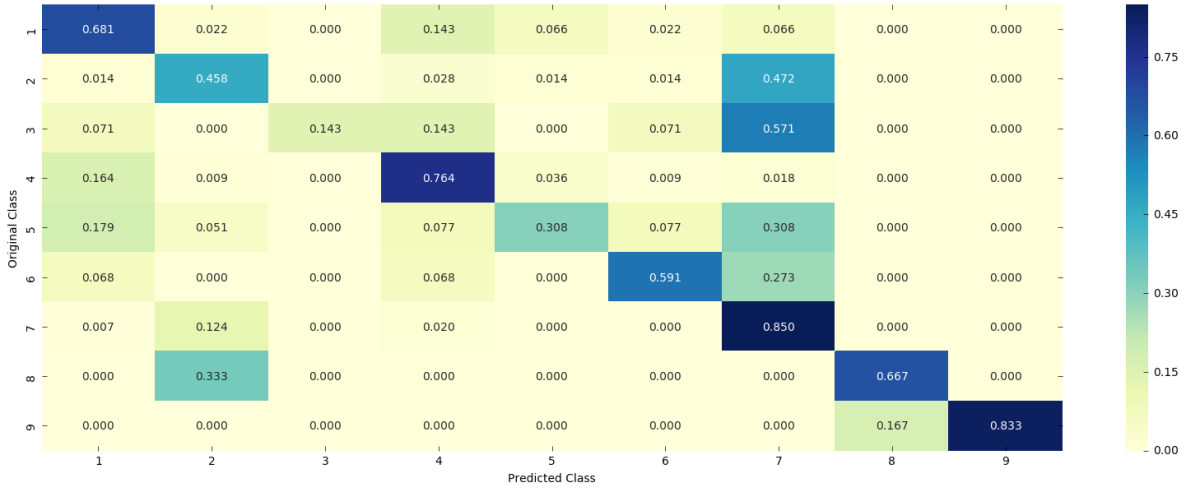
```
Log loss : 0.9913869905132415
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Feature Importance

In [48]:

```
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. Query point 1

In [49]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[
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'].ilo
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.800e-03 2.800e-03 7.000e-04 7.400e-03 3.1
00e-03 5.000e-04 9.685e-01
  1.410e-02 1.000e-04]]
Actual Class : 7
--------------------------------------------------
85 Text feature [12] present in test data point [True]
199 Text feature [0012] present in test data point [True]
486 Text feature [023] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.1.3.2. Query point 2

In [50]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[
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'].ilo
```

```
Predicted Class : 7
Predicted Class Probabilities: [[9.600e-03 5.020e-02 4.000e-04 1.740e-02 4.9
00e-03 3.600e-03 8.587e-01
  5.430e-02 9.000e-04]]
Actual Class : 7
--------------------------------------------------
85 Text feature [12] present in test data point [True]
102 Text feature [079] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.2. Without Class balancing

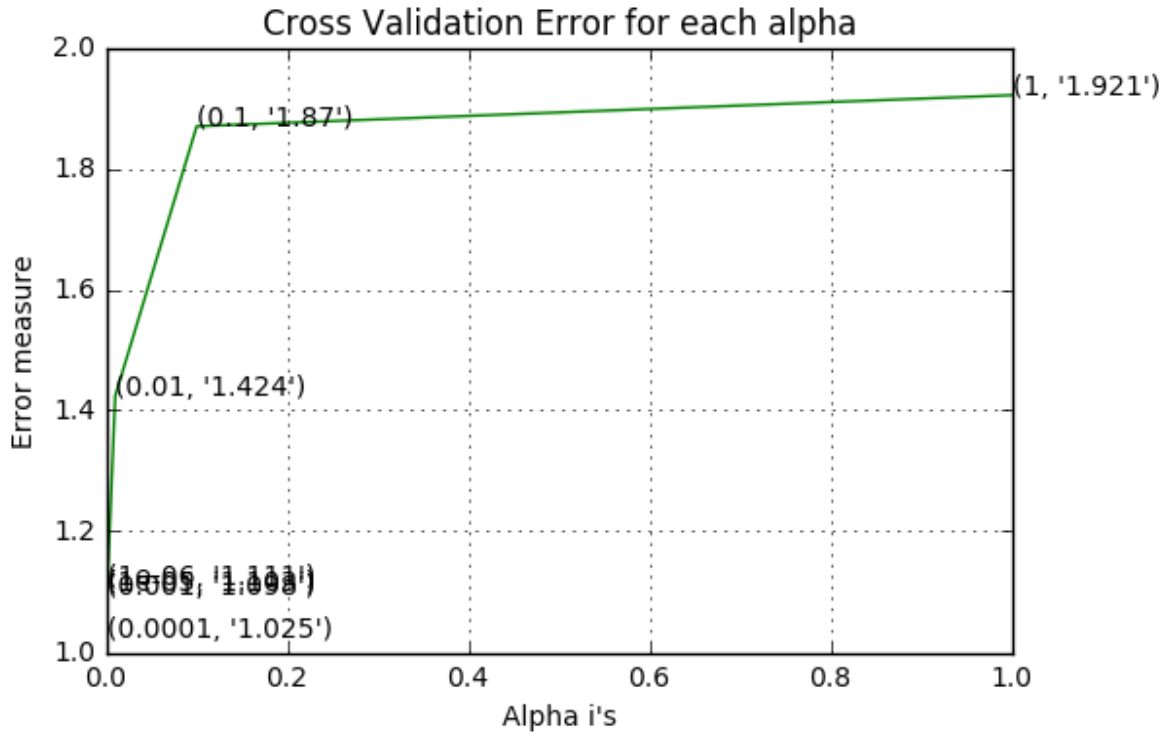
4.3.2.1. Hyper paramter tuning

In [51]:

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.111234706002402
for alpha = 1e-05
Log Loss : 1.1031364520593092
for alpha = 0.0001
Log Loss : 1.0249479270278747
for alpha = 0.001
Log Loss : 1.0975338812583246
for alpha = 0.01
Log Loss : 1.424015261566076
for alpha = 0.1
Log Loss : 1.8698360264558889
for alpha = 1
Log Loss : 1.9212341065855523
```



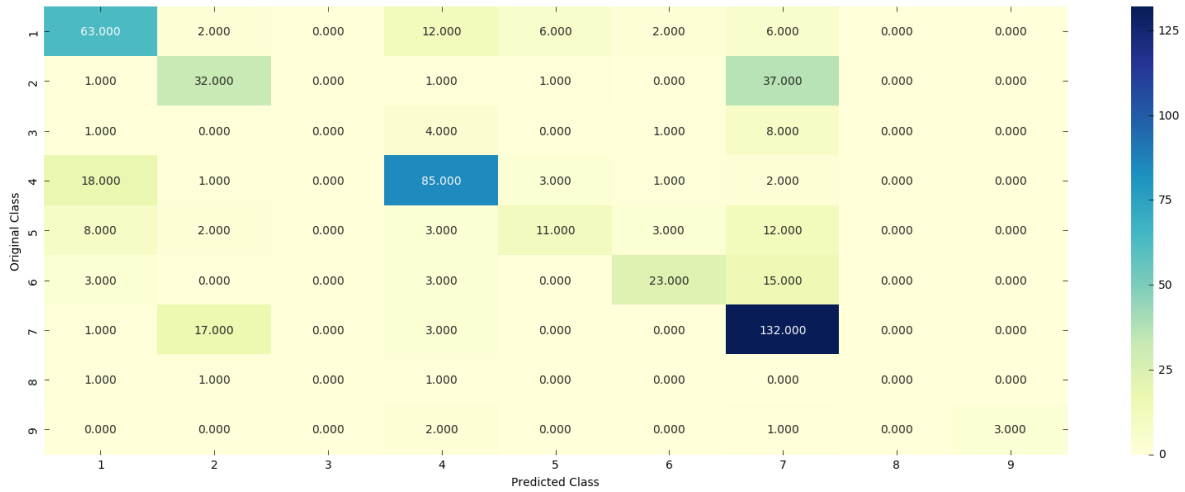

```
For values of best alpha =  0.0001 The train log loss is: 0.4340695471462118
For values of best alpha =  0.0001 The cross validation log loss is: 1.02494
79270278747
For values of best alpha =  0.0001 The test log loss is: 1.0631383305330515
```

4.3.2.2. Testing model with best hyper parameters

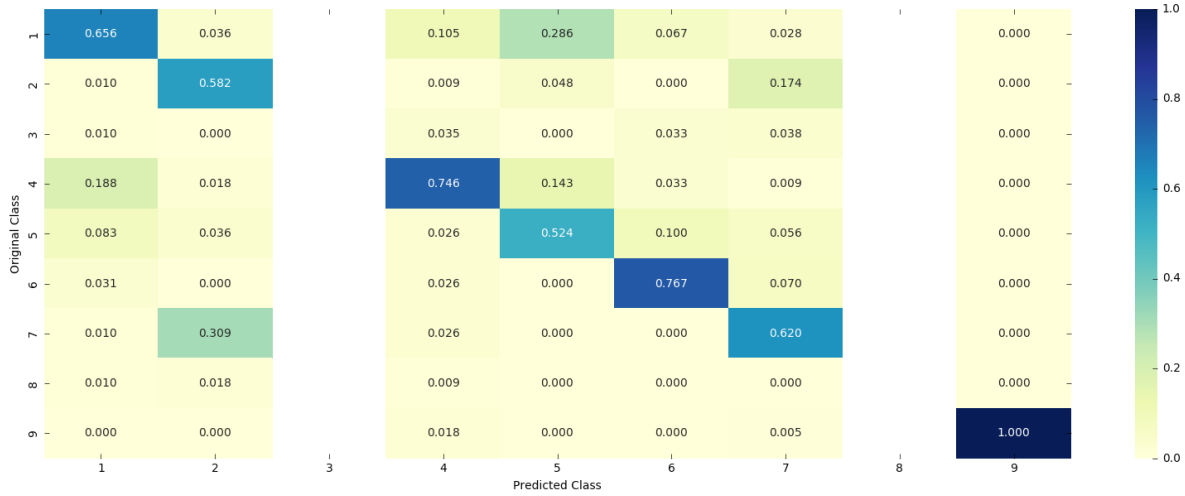
In [52]:

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

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

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

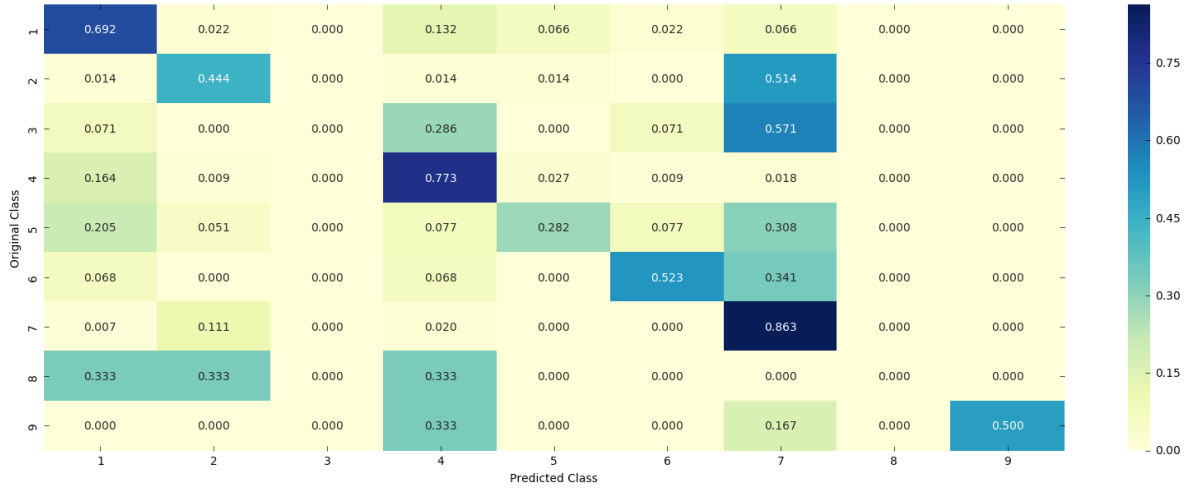
```
Log loss : 1.0249479270278747
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Query point 1

In [53]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[3.200e-03 2.700e-03 9.000e-04 9.100e-03 2.6
00e-03 4.000e-04 9.603e-01
  2.080e-02 0.000e+00]]
Actual Class : 7
--------------------------------------------------
97 Text feature [12] present in test data point [True]
317 Text feature [0012] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.2.4. Query point 2

In [54]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[
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'].ilo
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.150e-02 5.190e-02 4.000e-04 2.070e-02 4.2
00e-03 3.300e-03 8.832e-01
  2.470e-02 1.000e-04]]
Actual Class : 7
--------------------------------------------------
97 Text feature [12] present in test data point [True]
186 Text feature [079] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.4. Linear Support Vector Machines

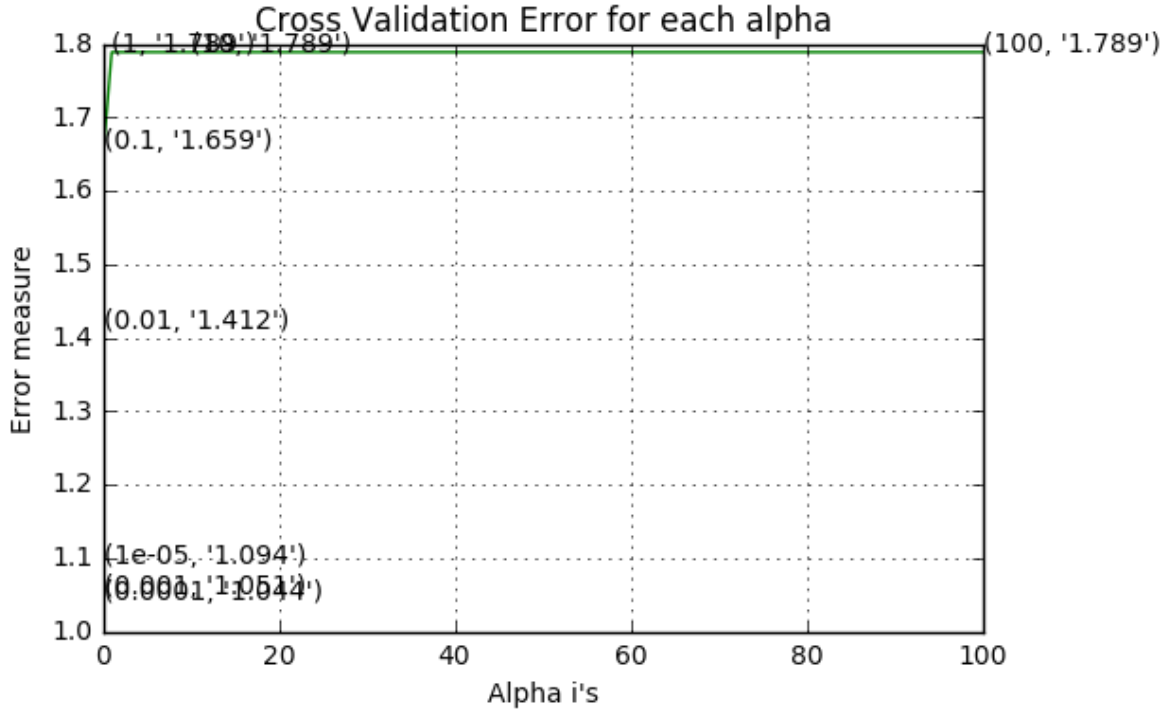
4.4.1. Hyper paramter tuning

In [55]:

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

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

# 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/mathe
# --------------------------------



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

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

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


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

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for C = 1e-05
Log Loss : 1.093581255054222
for C = 0.0001
Log Loss : 1.0442176947021733
for C = 0.001
Log Loss : 1.0513345289152918
for C = 0.01
Log Loss : 1.4117286403509524
for C = 0.1
Log Loss : 1.6590472113100139
for C = 1
Log Loss : 1.788630108233841
for C = 10
Log Loss : 1.7886298848323559
for C = 100
Log Loss : 1.788630055438651
```



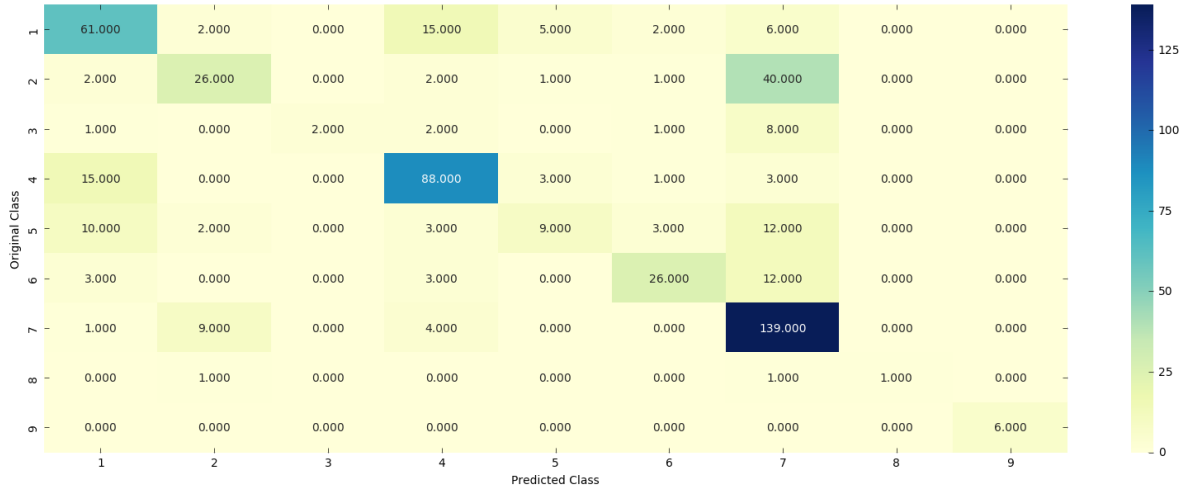

```
For values of best alpha =  0.0001 The train log loss is: 0.4853915524426819
For values of best alpha =  0.0001 The cross validation log loss is: 1.04421
76947021733
For values of best alpha =  0.0001 The test log loss is: 1.0941334950159225
```

4.4.2. Testing model with best hyper parameters

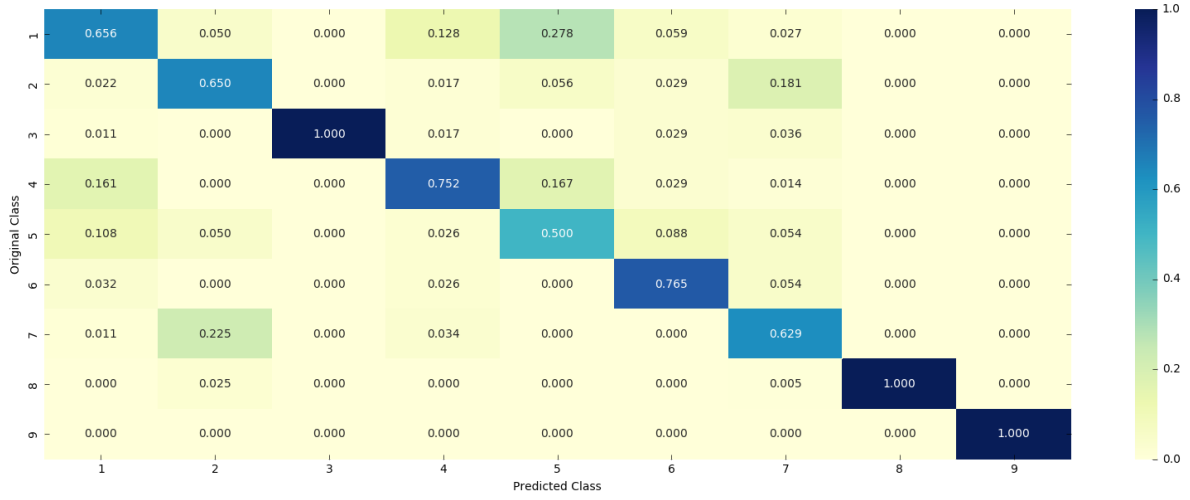
In [56]:

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

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

# 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/mathe
# ------------------------------


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

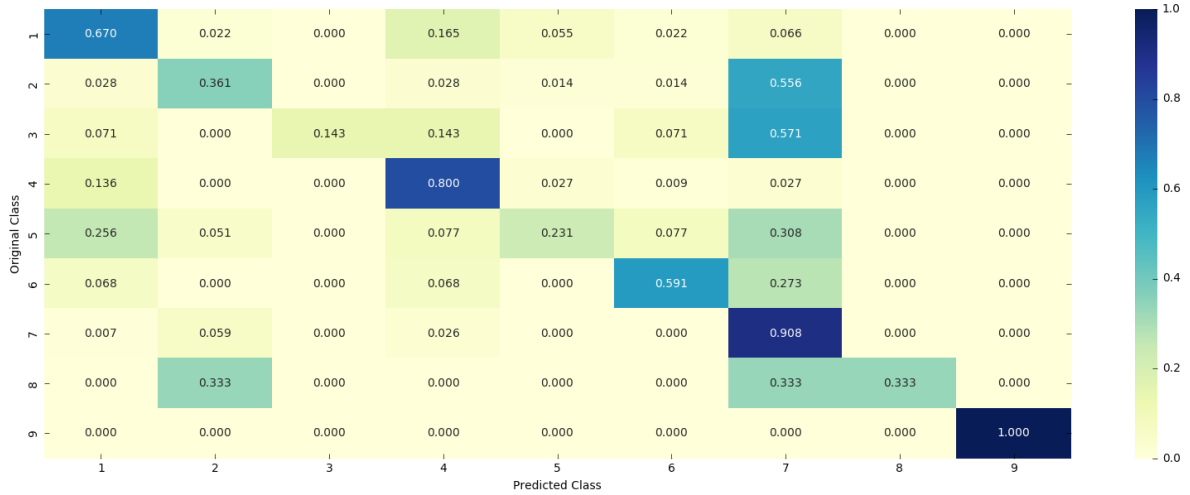
```
Log loss : 1.0442176947021733
Number of mis-classified points : 0.32706766917293234
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

4.3.3.1. Query point 1

In [57]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[
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'].ilc
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.750e-02 2.300e-03 5.000e-04 2.610e-02 2.8
90e-02 1.270e-02 8.911e-01
  1.070e-02 2.000e-04]]
Actual Class : 7
--------------------------------------------------
31 Text feature [12] present in test data point [True]
256 Text feature [023] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.3.2. Query point 2

In [58]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[
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'].ilo
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0605 0.0309 0.0017 0.0577 0.0242 0.0394
0.7553 0.0292 0.0011]]
Actual Class : 7
--------------------------------------------------
31 Text feature [12] present in test data point [True]
291 Text feature [079] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [59]:

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_
        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()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_error_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_d
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:"
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",
```

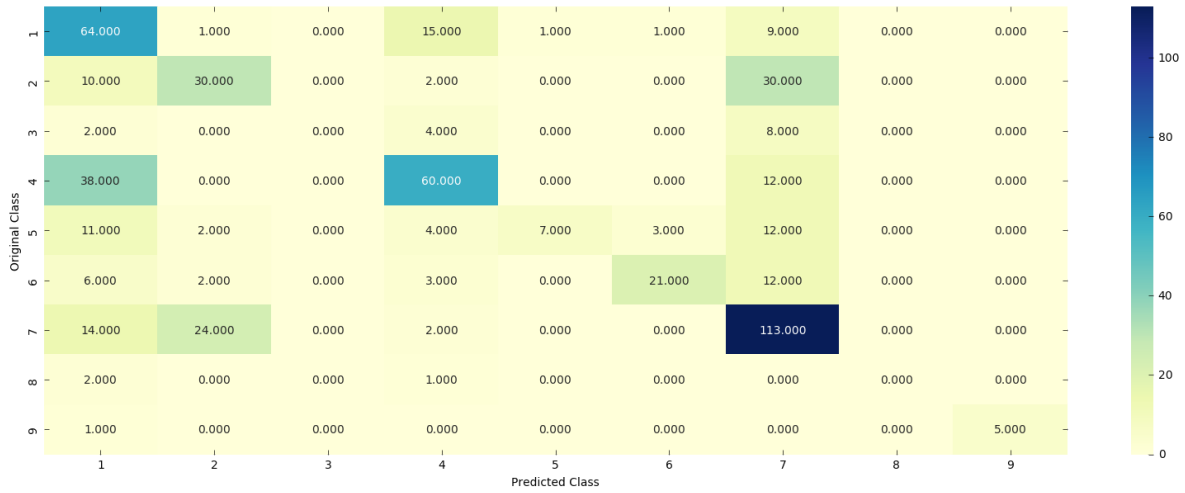
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2173637628822547
for n_estimators = 100 and max depth =  10
Log Loss : 1.2440005605589912
for n_estimators = 200 and max depth =  5
Log Loss : 1.2090815889702158
for n_estimators = 200 and max depth =  10
Log Loss : 1.2247218807024804
for n_estimators = 500 and max depth =  5
Log Loss : 1.203219777496975
for n_estimators = 500 and max depth =  10
Log Loss : 1.2226202359902434
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1968506036396285
for n_estimators = 1000 and max depth =  10
Log Loss : 1.217676533701273
for n_estimators = 2000 and max depth =  5
Log Loss : 1.192980849544485
for n_estimators = 2000 and max depth =  10
Log Loss : 1.218115164253433
For values of best estimator =  2000 The train log loss is: 0.85363123243664
34
For values of best estimator =  2000 The cross validation log loss is: 1.192
980849544485
For values of best estimator =  2000 The test log loss is: 1.173863368108815
```

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

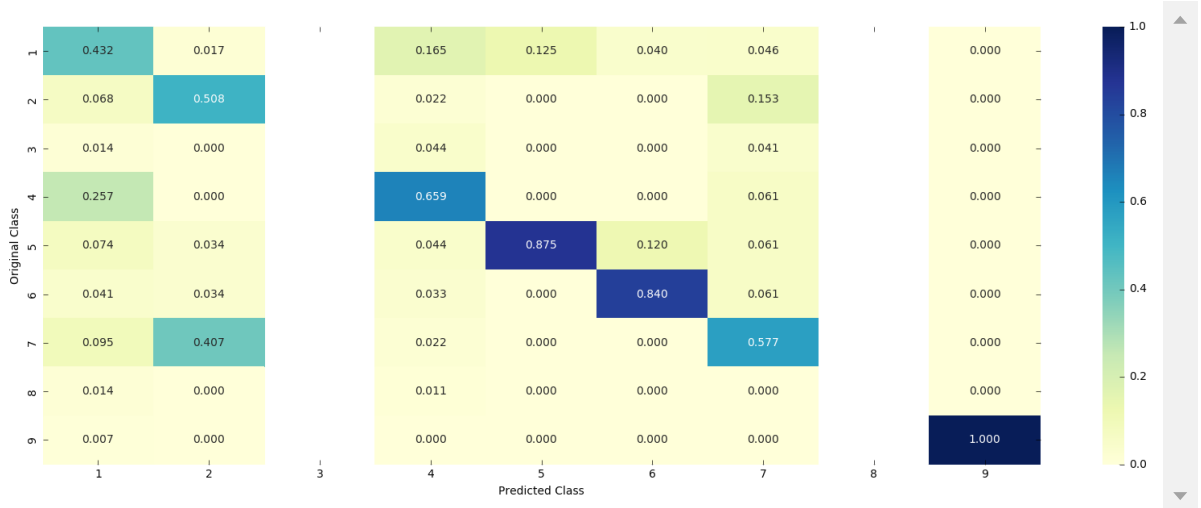
In [60]:

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

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

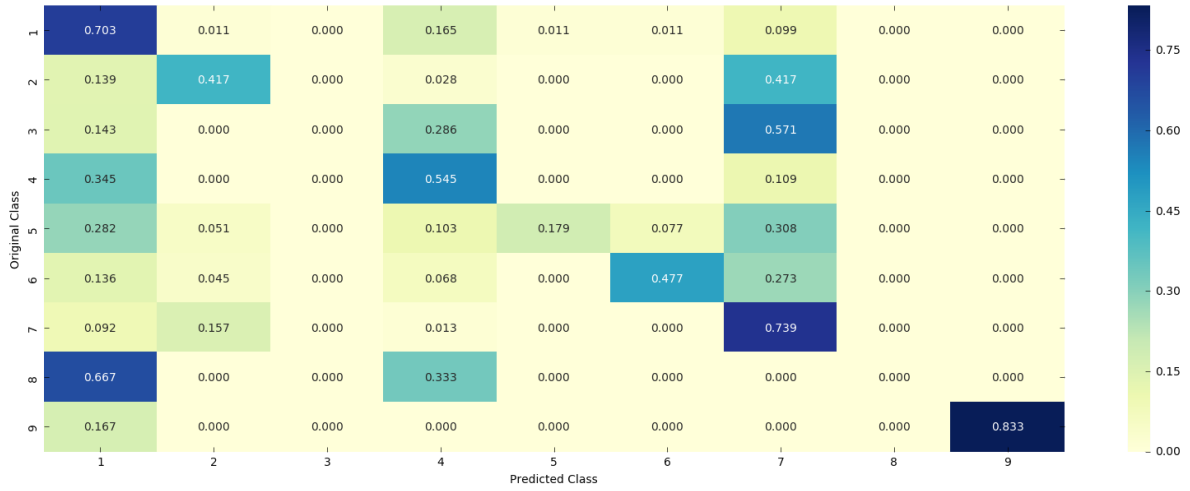
```
Log loss : 1.192980849544485
Number of mis-classified points : 0.43609022556390975
-------------------- Confusion matrix --------------------
```




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



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



4.5.3. Feature Importance

4.5.3.1. Query point 1

In [61]:

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

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0266 0.2261 0.0139 0.0268 0.037  0.034
0.6258 0.0065 0.0032]]
Actual Class : 7
--------------------------------------------------
25 Text feature [04] present in test data point [True]
83 Text feature [0012] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3.2. Query point 2

In [62]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0793 0.2998 0.0163 0.0607 0.0542 0.0438
0.4327 0.0083 0.0048]]
Actuall Class : 7
--------------------------------------------------
7 Text feature [104] present in test data point [True]
72 Text feature [079] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [63]:

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_
        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
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
'''

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2166092060701135
for n_estimators = 10 and max depth =  3
Log Loss : 1.7787197597285047
for n_estimators = 10 and max depth =  5
Log Loss : 1.5368473064904198
for n_estimators = 10 and max depth =  10
Log Loss : 1.6741753645254844
for n_estimators = 50 and max depth =  2
Log Loss : 1.760958976056977
for n_estimators = 50 and max depth =  3
Log Loss : 1.5316794397703808
for n_estimators = 50 and max depth =  5
Log Loss : 1.4747930884172178
for n_estimators = 50 and max depth =  10
Log Loss : 1.8427350580095139
for n_estimators = 100 and max depth =  2
Log Loss : 1.6618847693997008
for n_estimators = 100 and max depth =  3
Log Loss : 1.526823972920177
for n_estimators = 100 and max depth =  5
Log Loss : 1.4394129045937991
for n_estimators = 100 and max depth =  10
Log Loss : 1.7853612172649458
for n_estimators = 200 and max depth =  2
Log Loss : 1.7685227525340483
for n_estimators = 200 and max depth =  3
Log Loss : 1.5495045506309488
for n_estimators = 200 and max depth =  5
Log Loss : 1.4801590061755832
for n_estimators = 200 and max depth =  10
Log Loss : 1.8528880200854634
for n_estimators = 500 and max depth =  2
Log Loss : 1.8394041365436635
for n_estimators = 500 and max depth =  3
Log Loss : 1.6406110799558489
for n_estimators = 500 and max depth =  5
Log Loss : 1.495254658345441
for n_estimators = 500 and max depth =  10
Log Loss : 1.8721886447902703
for n_estimators = 1000 and max depth =  2
Log Loss : 1.8027984587402415
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6398200100412026
for n_estimators = 1000 and max depth =  5
```

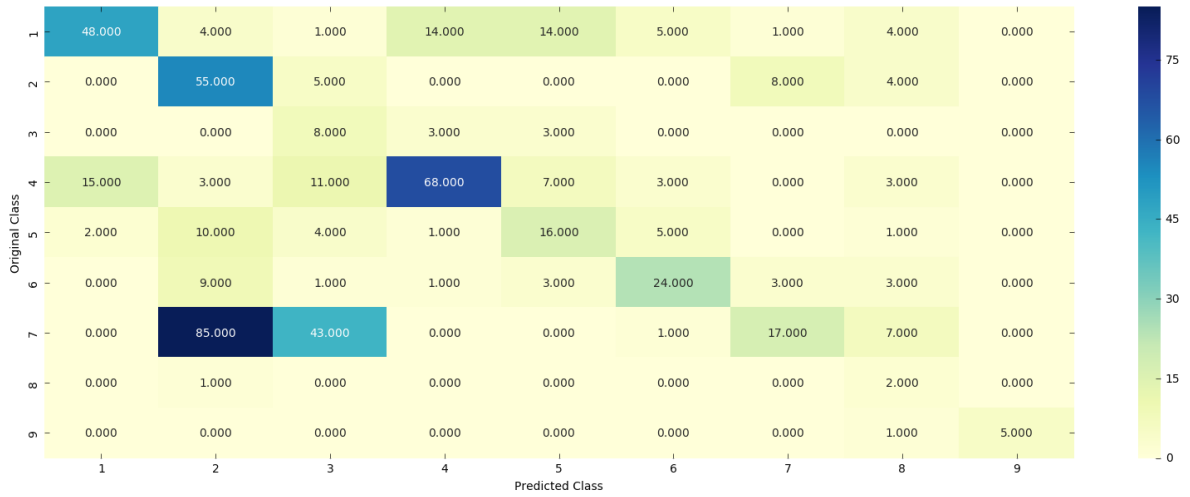
```
Log Loss : 1.4774820622314209
for n_estimators = 1000 and max depth =  10
Log Loss : 1.83608329077426
For values of best alpha =  100 The train log loss is: 0.052908505171030414
For values of best alpha =  100 The cross validation log loss is: 1.43941290
45938011
For values of best alpha =  100 The test log loss is: 1.4391570695023095
```

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

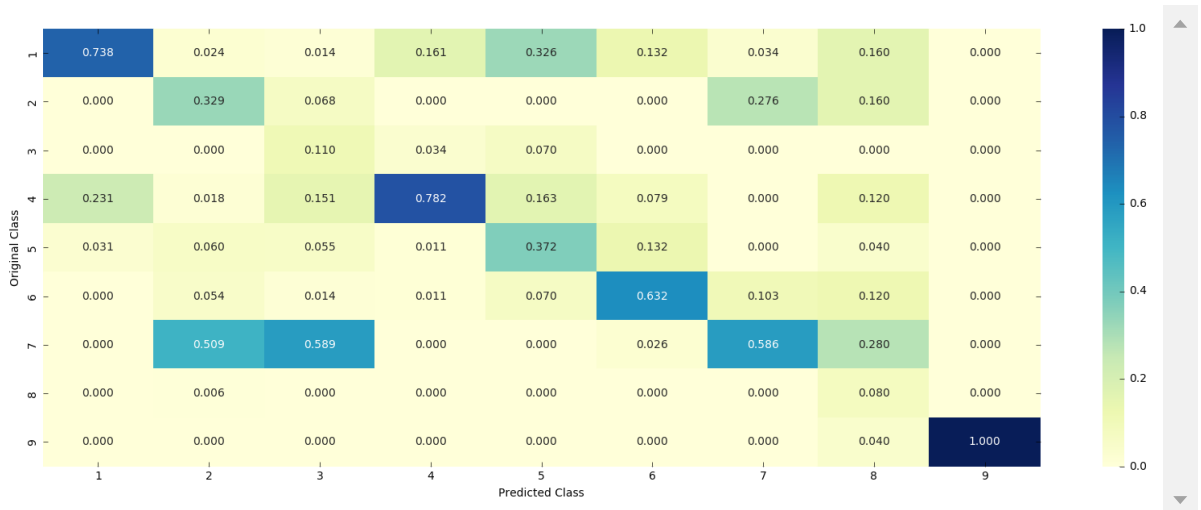
In [64]:

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

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

```
Log loss : 1.4394129045938004
Number of mis-classified points : 0.543233082706767
-------------------- Confusion matrix --------------------
```




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



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

4.5.5. Feature Importance

4.5.5.1. Query point 1

In [65]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_d
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_responseCodir
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 : 3
Predicted Class Probabilities: [[0.0171 0.1364 0.4695 0.0162 0.0307 0.025
0.1945 0.0948 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
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.5.5.2. Query point 2

In [66]:

```
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_responseCodir
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: [[0.0236 0.2925 0.1935 0.0229 0.0376 0.0402
0.2315 0.1389 0.0194]]
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
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [67]:

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

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


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

# 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/mathe
# --------------------------------


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


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

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

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

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

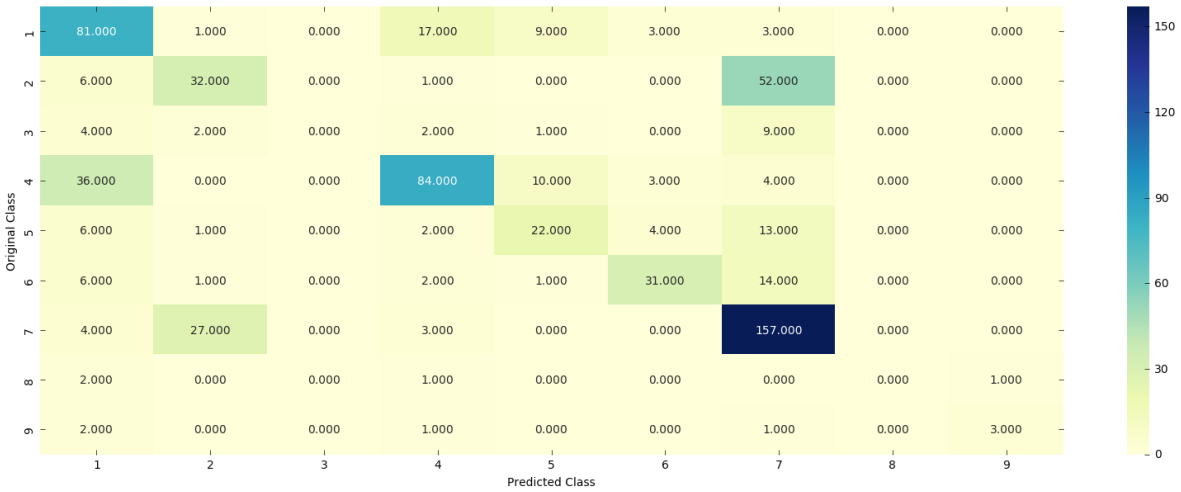
```
Logistic Regression :  Log Loss: 1.02
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.16
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.503
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.177
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.404
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.899
```

4.7.2 testing the model with the best hyper parameters

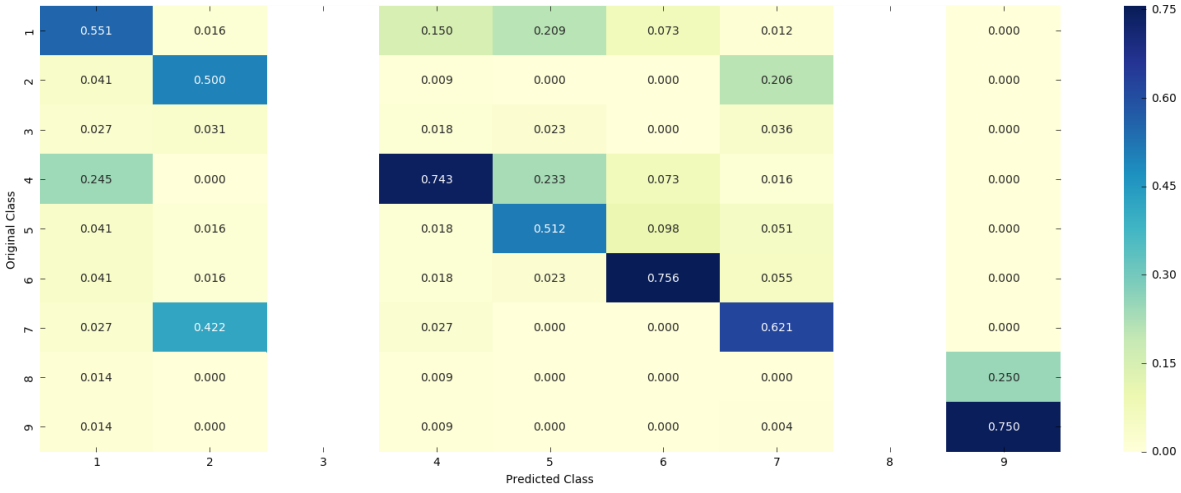
In [68]:

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

```
Log loss (train) on the stacking classifier : 0.5310981132260069
Log loss (CV) on the stacking classifier : 1.1765402806602043
Log loss (test) on the stacking classifier : 1.1751084645809629
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```




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



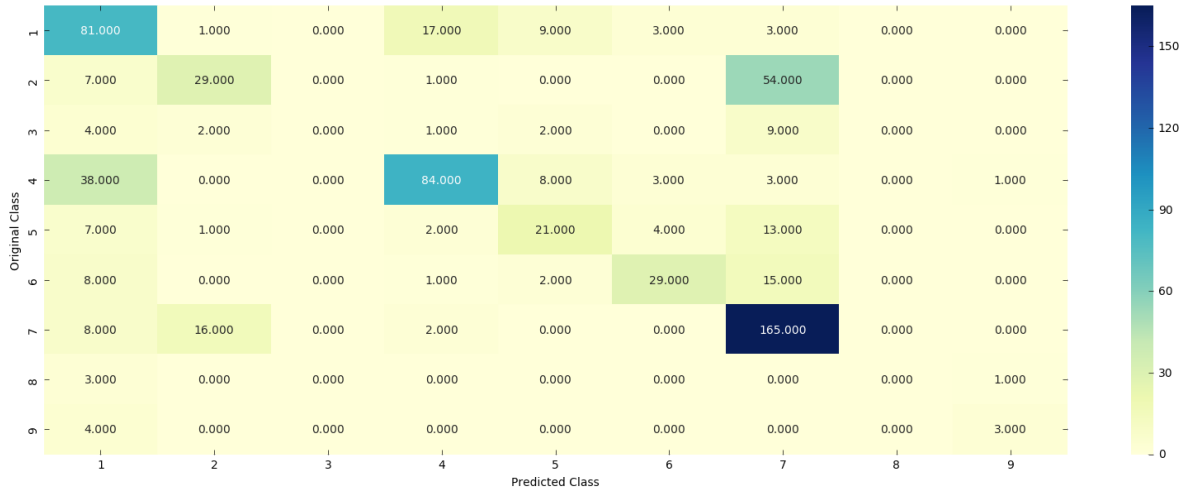

```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.7.3 Maximum Voting classifier

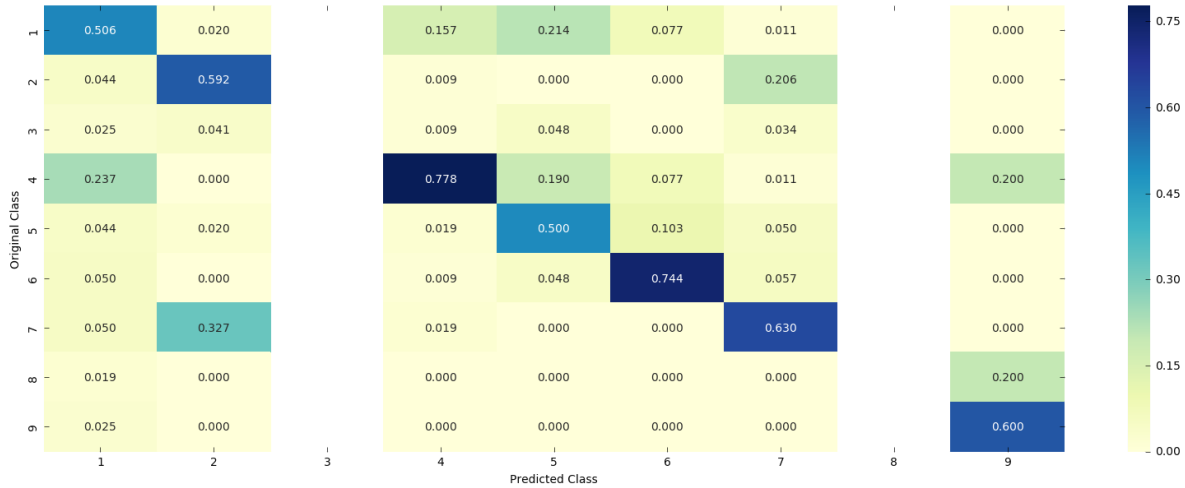
In [69]:

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

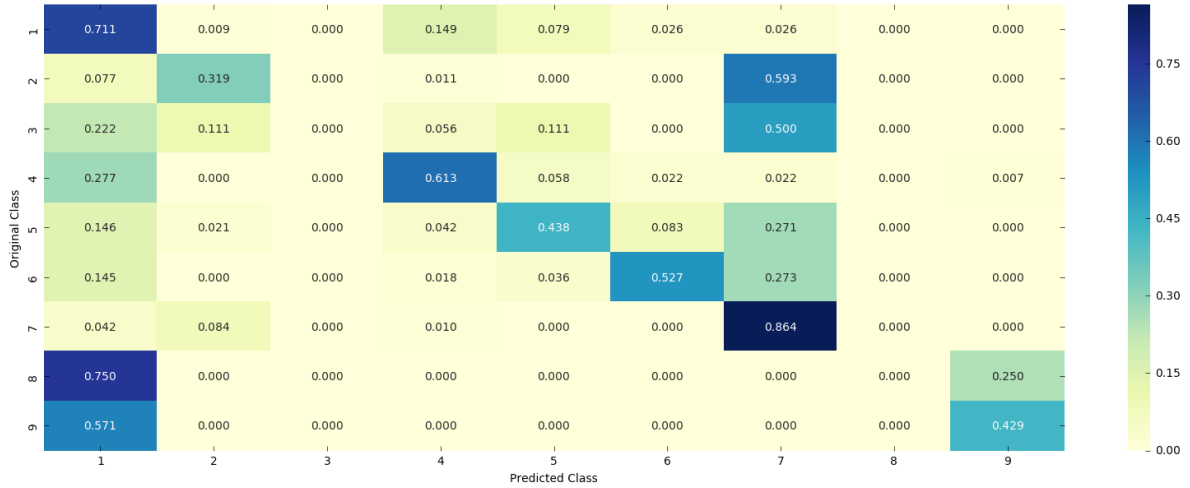
```
Log loss (train) on the VotingClassifier : 0.8254368699415732
Log loss (CV) on the VotingClassifier : 1.1750134972578687
Log loss (test) on the VotingClassifier : 1.2020291798855884
Number of missclassified point : 0.3804511278195489
-------------------- Confusion matrix --------------------
```




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




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



PART C: Applying Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [72]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer_bow = CountVectorizer(ngram_range = (1,2))
train_text_feature_onehotCoding_bow = text_vectorizer_bow.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features_bow = text_vectorizer_bow.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number o
train_text_fea_counts_bow = train_text_feature_onehotCoding_bow.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occu
text_fea_dict_bow = dict(zip(list(train_text_features_bow),train_text_fea_counts_bow))


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

```
Total number of unique words in train data : 2318906
```

In [73]:

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

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

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

In [74]:

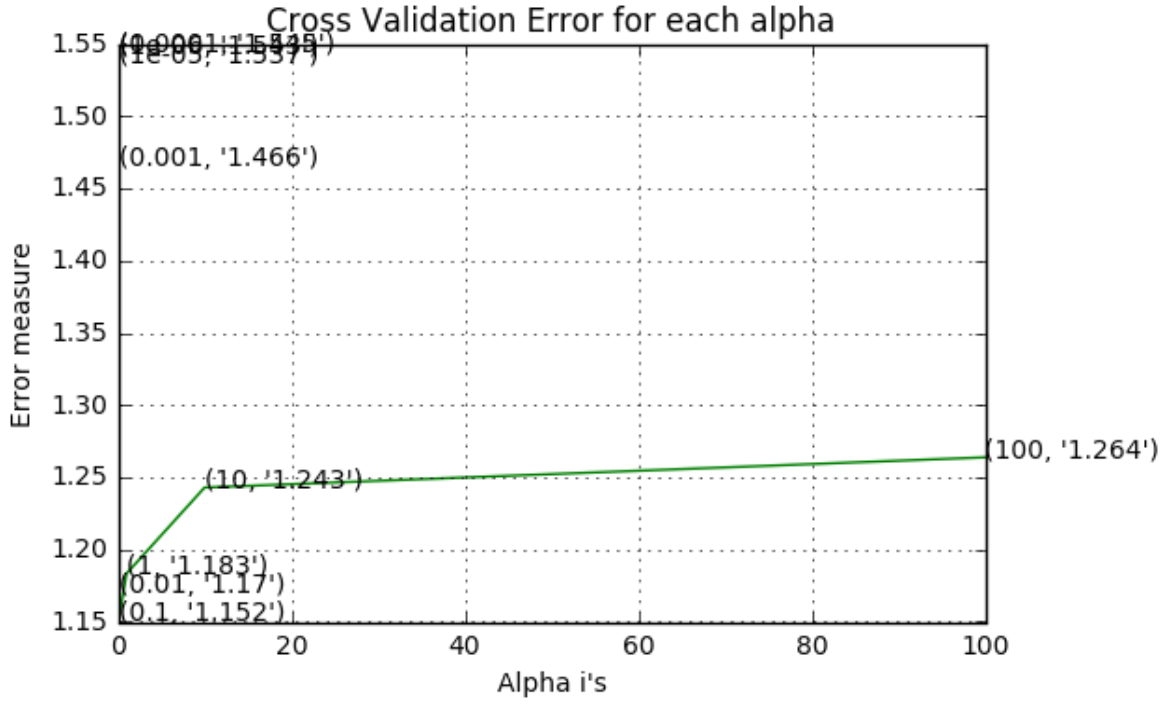
```
# merging gene, variance and text features

# 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_fea
# test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featur
# cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_oneh

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

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

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


# train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variat
# test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation
# cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_featu

# train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_res
# test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_respon
# cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCodi
```

In [75]:

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

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


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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.543166915845367
for alpha = 1e-05
Log Loss : 1.5373556424618713
for alpha = 0.0001
Log Loss : 1.5449934401925902
for alpha = 0.001
Log Loss : 1.4656178132575959
for alpha = 0.01
Log Loss : 1.17033922327687
for alpha = 0.1
Log Loss : 1.1517159073423373
for alpha = 1
Log Loss : 1.1834448675102172
for alpha = 10
Log Loss : 1.2429464627239997
for alpha = 100
Log Loss : 1.2638337831164044
```



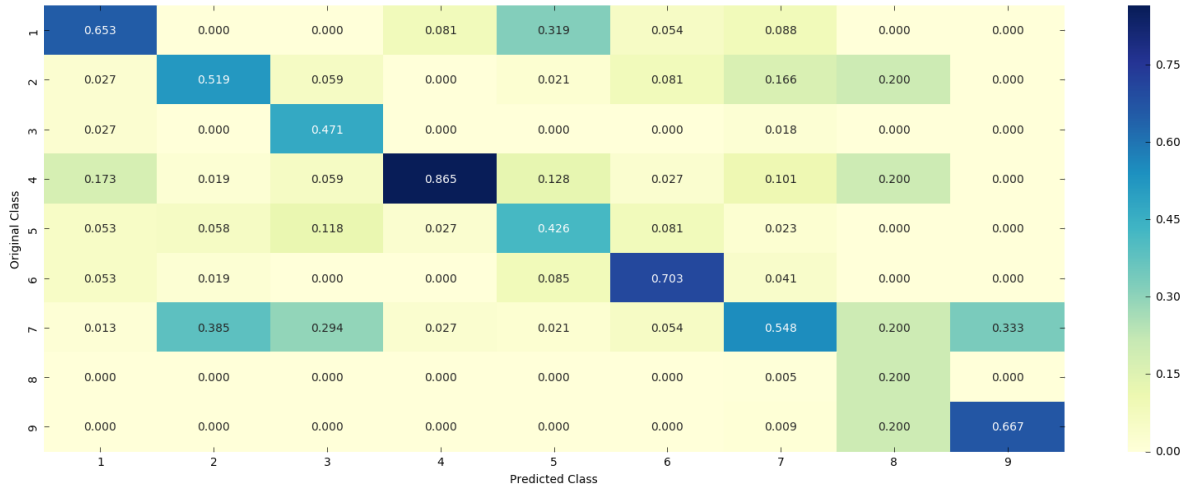

```
For values of best alpha =  0.1 The train log loss is: 0.6570213213783089
For values of best alpha =  0.1 The cross validation log loss is: 1.15171590
73423373
For values of best alpha =  0.1 The test log loss is: 1.2797398282905652
```

In [76]:

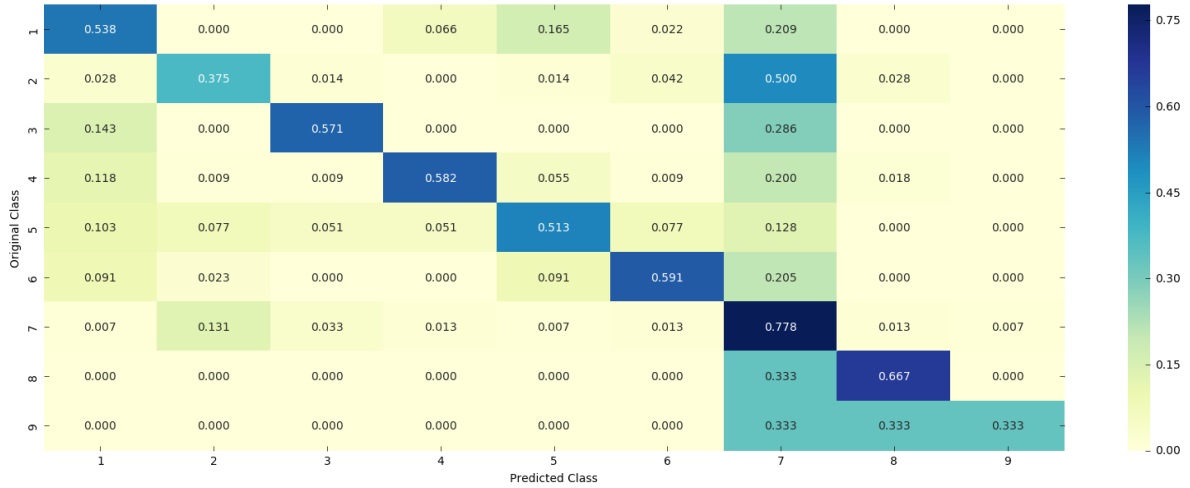
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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

```
Log loss : 1.1517159073423373
Number of mis-classified points : 0.4041353383458647
-------------------- Confusion matrix --------------------
```


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




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



PART C: Trying feature engineering techniques to reduce the CV and test log-loss to a value less than 1.0

In [19]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from tqdm import tqdm
from sklearn.preprocessing import StandardScaler
```

In [20]:

```
train_df['words_in_text'] = train_df.apply(lambda row: len(row.TEXT), axis = 1)
```

In [21]:

```
test_df['words_in_text'] = test_df.apply(lambda row: len(row.TEXT), axis = 1)
```

In [22]:

```
cv_df['words_in_text'] = cv_df.apply(lambda row: len(row.TEXT), axis = 1)
```

In [23]:

```
train_df.head(2)
```

Out[23]:

	ID	Gene	Variation	Class	TEXT	words_in_text
73	73	RAD50	L1273F	4	metastatic solid tumors almost invariably fata...	21005
3110	3110	RAD21	Deletion	1	asxl1 frequently mutated spectrum myeloid mali...	38512

In [24]:

```
test_df.head(2)
```

Out[24]:

	ID	Gene	Variation	Class	TEXT	words_in_text
1570	1570	ALK	H694R	7	oncogenic property anaplastic lymphoma kinase ...	35340
729	729	ERBB2	V842I	7	individuals gallbladder carcinoma gbc aggressi...	55311

In [25]:

```
cv_df.head(2)
```

Out[25]:

	ID	Gene	Variation	Class	TEXT	words_in_text
2218	2218	PTEN	C124R	4	tumour suppressor gene pten maps 10q23 3 encod...	55771
2436	2436	BRCA1	M1689T	1	genetic screening breast ovarian cancer suscep...	27133

In [26]:

```
print(train_df.shape)
print(test_df.shape)
print(cv_df.shape)
```

```
(2124, 6)
(665, 6)
(532, 6)
```

In [27]:

```
words_in_text_scalar = StandardScaler()
words_in_text_scalar.fit(train_df['words_in_text'].values.reshape(-1,1)) # finding the mean

# Now standardize the data with above maen and variance.
X_train_words_in_text_std = words_in_text_scalar.transform(train_df['words_in_text'].values
X_cv_words_in_text_std = words_in_text_scalar.transform(cv_df['words_in_text'].values.resha
X_test_words_in_text_std = words_in_text_scalar.transform(test_df['words_in_text'].values.r

print("Shape of matrix after one hot encodig ",X_train_words_in_text_std.shape)
print("Shape of matrix after one hot encodig ",X_cv_words_in_text_std.shape)
print("Shape of matrix after one hot encodig ",X_test_words_in_text_std.shape)
```

```
Shape of matrix after one hot encodig  (2124, 1)
Shape of matrix after one hot encodig  (532, 1)
Shape of matrix after one hot encodig  (665, 1)
```

In [28]:

```
train_df['freq_gene'] = train_df.groupby('Gene')['Gene'].transform('count')
test_df['freq_gene'] = test_df.groupby('Gene')['Gene'].transform('count')
cv_df['freq_gene'] = cv_df.groupby('Gene')['Gene'].transform('count')
```

In [29]:

```
print(train_df.shape)
print(test_df.shape)
print(cv_df.shape)
```

```
(2124, 7)
(665, 7)
(532, 7)
```

In [30]:

```
freq_gene_scalar = StandardScaler()
freq_gene_scalar.fit(train_df['freq_gene'].values.reshape(-1,1)) # finding the mean and sta

# Now standardize the data with above maen and variance.
X_train_freq_gene_std = freq_gene_scalar.transform(train_df['freq_gene'].values.reshape(-1,
X_cv_freq_gene_std = freq_gene_scalar.transform(cv_df['freq_gene'].values.reshape(-1, 1))#
X_test_freq_gene_std = freq_gene_scalar.transform(test_df['freq_gene'].values.reshape(-1, 1

print("Shape of matrix after one hot encodig ",X_train_freq_gene_std.shape)
print("Shape of matrix after one hot encodig ",X_cv_freq_gene_std.shape)
print("Shape of matrix after one hot encodig ",X_test_freq_gene_std.shape)
```

```
Shape of matrix after one hot encodig  (2124, 1)
Shape of matrix after one hot encodig  (532, 1)
Shape of matrix after one hot encodig  (665, 1)
```

In [31]:

```
train_df['freq_variation'] = train_df.groupby('Variation')['Variation'].transform('count')
test_df['freq_variation'] = test_df.groupby('Variation')['Variation'].transform('count')
cv_df['freq_variation'] = cv_df.groupby('Variation')['Variation'].transform('count')
```

In [32]:

```
train_df.head(2)
```

Out[32]:

	ID	Gene	Variation	Class	TEXT	words_in_text	freq_gene	freq_variation
73	73	RAD50	L1273F	4	metastatic solid tumors almost invariably fata...	21005	3	1
3110	3110	RAD21	Deletion	1	asxl1 frequently mutated spectrum myeloid mali...	38512	2	51

In [33]:

```
freq_variation_scalar = StandardScaler()
freq_variation_scalar.fit(train_df['freq_variation'].values.reshape(-1,1)) # finding the me

# Now standardize the data with above maen and variance.
X_train_freq_variation_std = freq_variation_scalar.transform(train_df['freq_variation'].val
X_cv_freq_variation_std = freq_variation_scalar.transform(cv_df['freq_variation'].values.re
X_test_freq_variation_std = freq_variation_scalar.transform(test_df['freq_variation'].value

print("Shape of matrix after one hot encodig ",X_train_freq_variation_std.shape)
print("Shape of matrix after one hot encodig ",X_cv_freq_variation_std.shape)
print("Shape of matrix after one hot encodig ",X_test_freq_variation_std.shape)
```

```
Shape of matrix after one hot encodig  (2124, 1)
Shape of matrix after one hot encodig  (532, 1)
Shape of matrix after one hot encodig  (665, 1)
```

In [34]:

```
train_df['Gene_Variation'] = train_df['Gene'] + " " + train_df['Variation']
test_df['Gene_Variation'] = test_df['Gene'] + " " + test_df['Variation']
cv_df['Gene_Variation'] = cv_df['Gene'] + " " + cv_df['Variation']
```

In [35]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene_Variatic
# test gene feature
test_gene_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene_Variation
# cross validation gene feature
cv_gene_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene_Variation",
```

In [36]:

```
# one-hot encoding of Gene_Variation feature.
gene_variation_vectorizer = CountVectorizer()
train_gene_variation_feature_onehotCoding = gene_variation_vectorizer.fit_transform(train_d
test_gene_variation_feature_onehotCoding = gene_variation_vectorizer.transform(test_df['Ger
cv_gene_variation_feature_onehotCoding = gene_variation_vectorizer.transform(cv_df['Gene_Va
```

In [37]:

```
train_df["words_in_text_50000"] = train_df["words_in_text"].apply(lambda x: 1 if x > 50000
test_df["words_in_text_50000"] = test_df["words_in_text"].apply(lambda x: 1 if x > 50000 el
cv_df["words_in_text_50000"] = cv_df["words_in_text"].apply(lambda x: 1 if x > 50000 else 0
```

In [38]:

```
words_50000_scalar = StandardScaler()
words_50000_scalar.fit(train_df['words_in_text_50000'].values.reshape(-1,1)) # finding the

# Now standardize the data with above maen and variance.
X_train_words_50000_std = words_50000_scalar.transform(train_df['words_in_text_50000'].valu
X_cv_words_50000_std = words_50000_scalar.transform(cv_df['words_in_text_50000'].values.res
X_test_words_50000_std = words_50000_scalar.transform(test_df['words_in_text_50000'].values

print("Shape of matrix after one hot encodig ",X_train_words_50000_std.shape)
print("Shape of matrix after one hot encodig ",X_cv_words_50000_std.shape)
print("Shape of matrix after one hot encodig ",X_test_words_50000_std.shape)
```

```
Shape of matrix after one hot encodig  (2124, 1)
Shape of matrix after one hot encodig  (532, 1)
Shape of matrix after one hot encodig  (665, 1)
```

In [39]:

```
#https://stackoverflow.com/a/12761576
def avg_word_length(sentence):
    word_len = []
    for i in tqdm(sentence):
#         print(i)
        words = i.split()
        average = sum(len(word) for word in words) / len(words)
        word_len.append(average)
    return word_len
```

In [40]:

```
train_text = train_df['TEXT'].tolist()
test_text = test_df['TEXT'].tolist()
cv_text = cv_df['TEXT'].tolist()
train_avg_word_len = avg_word_length(train_text)
test_avg_word_len = avg_word_length(test_text)
cv_avg_word_len = avg_word_length(cv_text)
```

```
100%|██████████| 2124/2124 [00:02<00:00, 850.31it/s]
100%|██████████| 665/665 [00:00<00:00, 797.84it/s]
100%|██████████| 532/532 [00:00<00:00, 858.31it/s]
```

In [ ]:

```
train_avg_word_len
```

In [42]:

```
train_df["Average_Text_length"] = train_avg_word_len
test_df["Average_Text_length"] = test_avg_word_len
cv_df["Average_Text_length"] = cv_avg_word_len
```

In [43]:

```
avg_text_len_scalar = StandardScaler()
avg_text_len_scalar.fit(train_df['Average_Text_length'].values.reshape(-1,1)) # finding the

# Now standardize the data with above maen and variance.
X_train_avg_text_len_std = avg_text_len_scalar.transform(train_df['Average_Text_length'].va
X_cv_avg_text_len_std = avg_text_len_scalar.transform(cv_df['Average_Text_length'].values.r
X_test_avg_text_len_std = avg_text_len_scalar.transform(test_df['Average_Text_length'].valu

print("Shape of matrix after one hot encodig ",X_train_avg_text_len_std.shape)
print("Shape of matrix after one hot encodig ",X_cv_avg_text_len_std.shape)
print("Shape of matrix after one hot encodig ",X_test_avg_text_len_std.shape)
```

```
Shape of matrix after one hot encodig  (2124, 1)
Shape of matrix after one hot encodig  (532, 1)
Shape of matrix after one hot encodig  (665, 1)
```

In [44]:

```
train_df.head(2)
```

Out[44]:

	ID	Gene	Variation	Class	TEXT	words_in_text	freq_gene	freq_variation
73	73	RAD50	L1273F	4	metastatic solid tumors almost invariably fata...	21005	3	1
3110	3110	RAD21	Deletion	1	asxl1 frequently mutated spectrum myeloid mali...	38512	2	51

In [59]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer_tfidf = TfidfVectorizer(ngram_range = (1,4), max_features = 5000)
train_text_feature_onehotCoding_tfidf = text_vectorizer_tfidf.fit_transform(train_df['TEXT'
# getting all the feature names (words)
train_text_features_tfidf = text_vectorizer_tfidf.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number o
train_text_fea_counts_tfidf = train_text_feature_onehotCoding_tfidf.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occu
text_fea_dict_tfidf = dict(zip(list(train_text_features_tfidf),train_text_fea_counts_tfidf)


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

```
Total number of unique words in train data : 5000
```

In [60]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding_tfidf = normalize(train_text_feature_onehotCoding_tfidf, ax

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

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

In [80]:

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

In [118]:

```
best_select_k = SelectKBest(chi2, k=2000)
train_text_feature_onehotCoding_tfidf = best_select_k.fit_transform(train_text_feature_oneh
cv_text_feature_onehotCoding_tfidf = best_select_k.transform(cv_text_feature_onehotCoding_t
test_text_feature_onehotCoding_tfidf = best_select_k.transform(test_text_feature_onehotCodi
print(train_text_feature_onehotCoding_tfidf.shape)
print(cv_text_feature_onehotCoding_tfidf.shape)
print(test_text_feature_onehotCoding_tfidf.shape)
```

```
(2124, 2000)
(532, 2000)
(665, 2000)
```

In [61]:

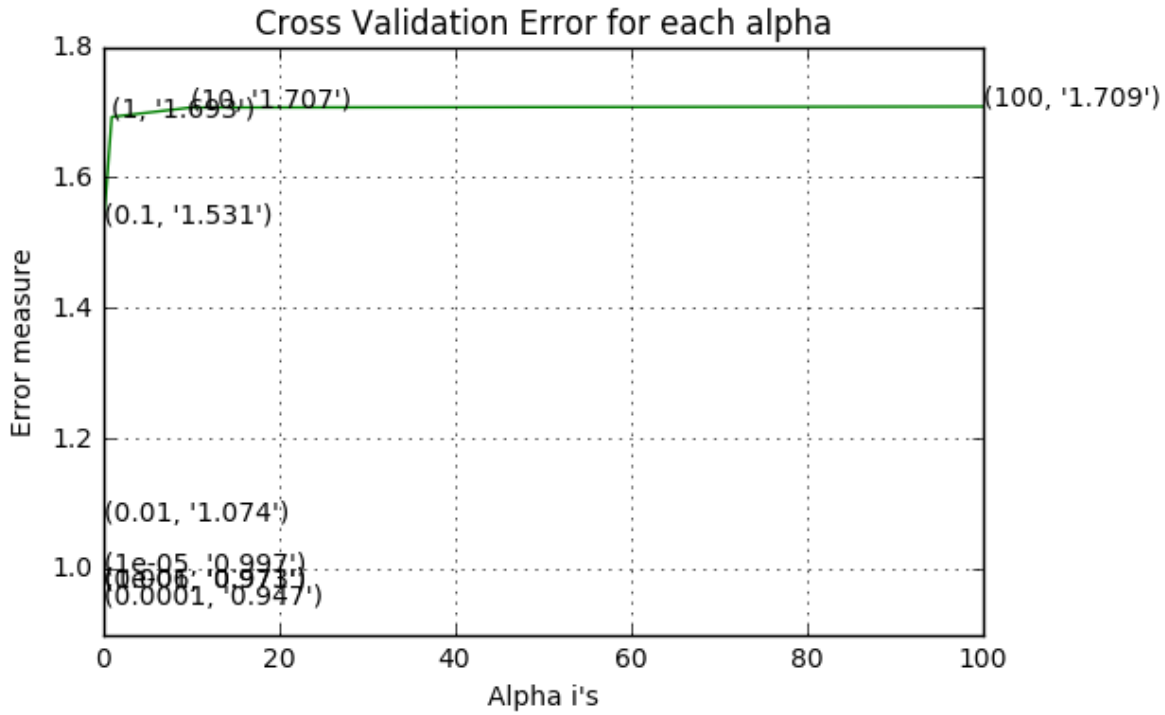
```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_featu
                                      X_train_words_in_text_std, X_train_freq_gene_std, X_tr
                                       train_gene_variation_feature_onehotCoding, X_train_wo
                                      X_train_avg_text_len_std))

test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_
                                     X_test_words_in_text_std, X_test_freq_gene_std, X_test_
                                     test_gene_variation_feature_onehotCoding, X_test_words_
                                     X_test_avg_text_len_std))

cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehot
                                   X_cv_words_in_text_std, X_cv_freq_gene_std, X_cv_freq_var
                                   cv_gene_variation_feature_onehotCoding, X_cv_words_50000_
                                   X_cv_avg_text_len_std))


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

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

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


# train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variat
# test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation
# cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_featu

# train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_res
# test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_respon
# cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCodi
```

In [62]:

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

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


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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 0.9726474240624396
for alpha = 1e-05
Log Loss : 0.9966290778609723
for alpha = 0.0001
Log Loss : 0.9465802918431186
for alpha = 0.001
Log Loss : 0.9710837712192022
for alpha = 0.01
Log Loss : 1.0735021422793032
for alpha = 0.1
Log Loss : 1.5305081177970328
for alpha = 1
Log Loss : 1.6925576456248228
for alpha = 10
Log Loss : 1.7069853780658244
for alpha = 100
Log Loss : 1.7085130810340872
```



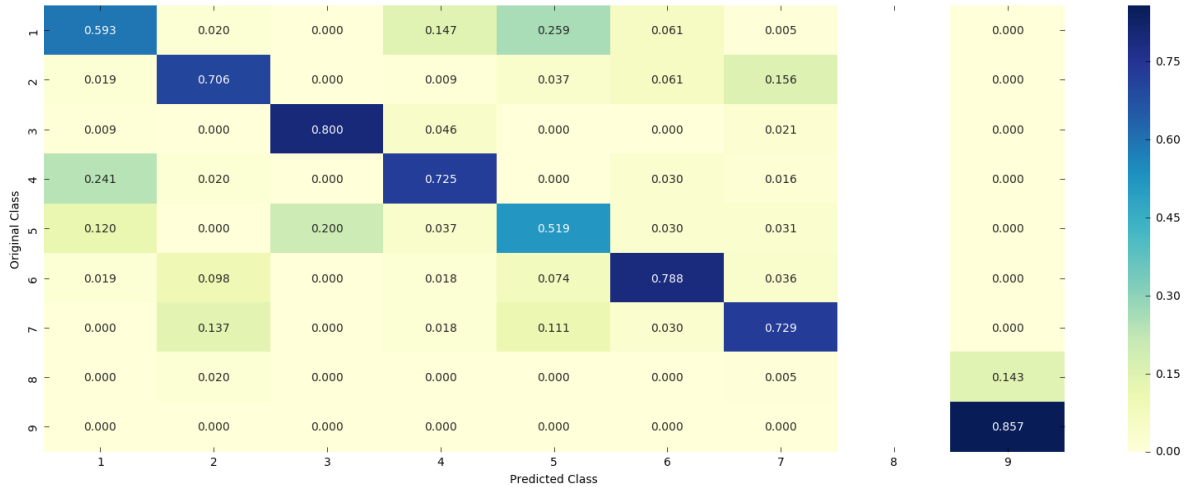

```
For values of best alpha =  0.0001 The train log loss is: 0.4671615539103785
For values of best alpha =  0.0001 The cross validation log loss is: 0.94658
02918431186
For values of best alpha =  0.0001 The test log loss is: 0.9887541060934563
```

In [67]:

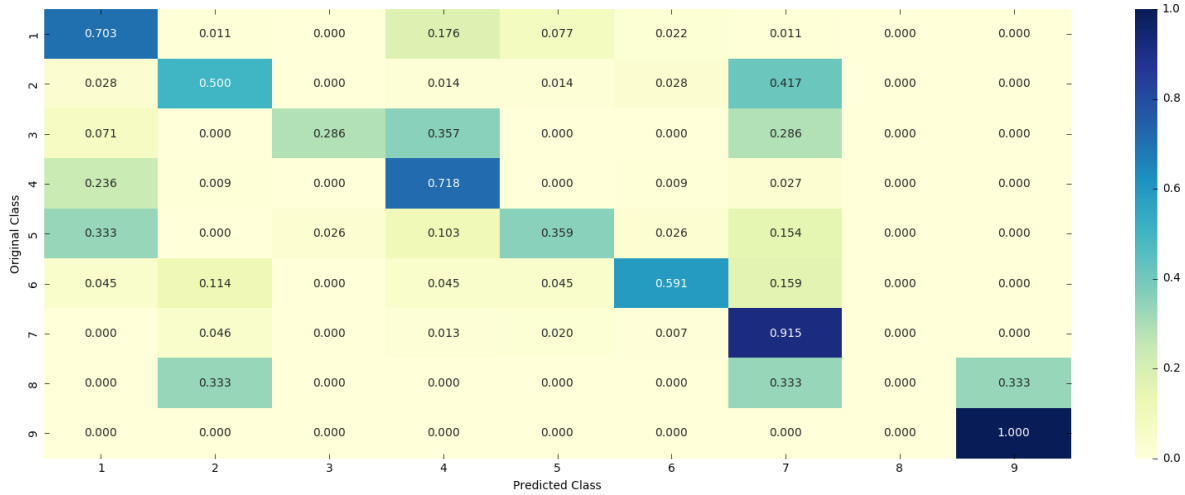
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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

```
Log loss : 0.9465802918431186
Number of mis-classified points : 0.30639097744360905
-------------------- Confusion matrix --------------------
```


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




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


Original Class
Predicted Class
1 0.703 0.011 0.000 0.176 0.077 0.022 0.011 0.000 0.000
2 0.028 0.500 0.000 0.014 0.014 0.028 0.417 0.000 0.000
3 0.071 0.000 0.286 0.357 0.000 0.000 0.286 0.000 0.000
4 0.236 0.009 0.000 0.718 0.000 0.009 0.027 0.000 0.000
5 0.333 0.000 0.026 0.103 0.359 0.026 0.154 0.000 0.000
6 0.045 0.114 0.000 0.045 0.045 0.591 0.159 0.000 0.000
7 0.000 0.046 0.000 0.013 0.020 0.007 0.915 0.000 0.000
8 0.000 0.333 0.000 0.000 0.000 0.000 0.333 0.000 0.333
9 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000
1 2 3 4 5 6 7 8 9
1.0 0.8 0.6 0.4 0.2 0.0

In [68]:

```
print()
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.field_names=["Model Name","Train","CV","Test","% Misclassified Points"]
ptable.add_row(["Naive Bayes","0.52","1.16","1.19","37.40"])
ptable.add_row(["KNN","0.64","1.02","1.06","37.03"])
ptable.add_row(["Logistic Regression With Class balancing","0.44","0.99","1.04","33.08"])
ptable.add_row(["Logistic Regression Without Class balancing","0.43","1.02","1.06","34.39"]
ptable.add_row(["Linear SVM","0.49","1.04","1.09","32.70"])
ptable.add_row(["Random Forest Classifier With One hot Encoding","0.85","1.19","1.17","43.6
ptable.add_row(["Random Forest Classifier With Response Coding","0.05","1.44","1.44","54.32
ptable.add_row(["Stacked Models:LR+NB+SVM","0.53","1.18","1.18","38.34"])
ptable.add_row(["Maximum Voting classifier","0.83","1.18","1.20","38.04"])
ptable.add_row(["Logistic Regression with Uni and Bi-grams","0.65","1.15","1.28","40.41"])
ptable.add_row(["Logistic Regression with more Features","0.46","0.94","0.98","30.63"])
print(ptable)
print()
```

```
+-------------------------------------------------+-------+------+------+----
--------------------+
|                    Model Name                   | Train |  CV  | Test | % M
isclassified Points |
+-------------------------------------------------+-------+------+------+----
--------------------+
|                   Naive Bayes                   |  0.52 | 1.16 | 1.19 |
37.40          |
|                       KNN                       |  0.64 | 1.02 | 1.06 |
37.03          |
|    Logistic Regression With Class balancing     |  0.44 | 0.99 | 1.04 |
33.08          |
|  Logistic Regression Without Class balancing    |  0.43 | 1.02 | 1.06 |
34.39          |
|                    Linear SVM                   |  0.49 | 1.04 | 1.09 |
32.70          |
| Random Forest Classifier With One hot Encoding  |  0.85 | 1.19 | 1.17 |
43.60          |
| Random Forest Classifier With Response Coding   |  0.05 | 1.44 | 1.44 |
54.32          |
|            Stacked Models:LR+NB+SVM             |  0.53 | 1.18 | 1.18 |
38.34          |
|           Maximum Voting classifier             |  0.83 | 1.18 | 1.20 |
38.04          |
|   Logistic Regression with Uni and Bi-grams     |  0.65 | 1.15 | 1.28 |
40.41          |
|     Logistic Regression with more Features      |  0.46 | 0.94 | 0.98 |
30.63          |
+-------------------------------------------------+-------+------+------+----
--------------------+
```

Procedure Followed:

- Exploratory Data Analysis: Gene, Variation and Text.
- Vectorized categorical columns Gene and Variation using CountVectorizer(One hot encoding) and Response Coding.

- Encoded Text column using TfidfVectorizer using 1000 top features.
- Stacked all the features.
- Hyperparameter tuning of various models.
- Trained models using various algorithms like Logistic Regression, Linear SVM, Stacked Models, Naive Bayes, Random Forest, KNN, Maximum Voting Classifier.
- Tried reducing the Log-loss using feature engineering.