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

3. Exploratory Data Analysis

In [1]:

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

```
C:\Users\Anirban\Anaconda3\lib\site-packages\sklearn\externals\six.py:31: De
precationWarning: The module is deprecated in version 0.21 and will be remov
ed in version 0.23 since we've dropped support for Python 2.7. Please rely o
n the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],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]:

```
from nltk.corpus import stopwords
```

In [5]:

```
# 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 [6]:

```
#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 : 144.16893290000002 seconds
```

In [7]:

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

Out[7]:

	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 [8]:

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

Out[8]:

	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 [9]:

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

In [10]:

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

Out[10]:

	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 [11]:

```
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 [12]:

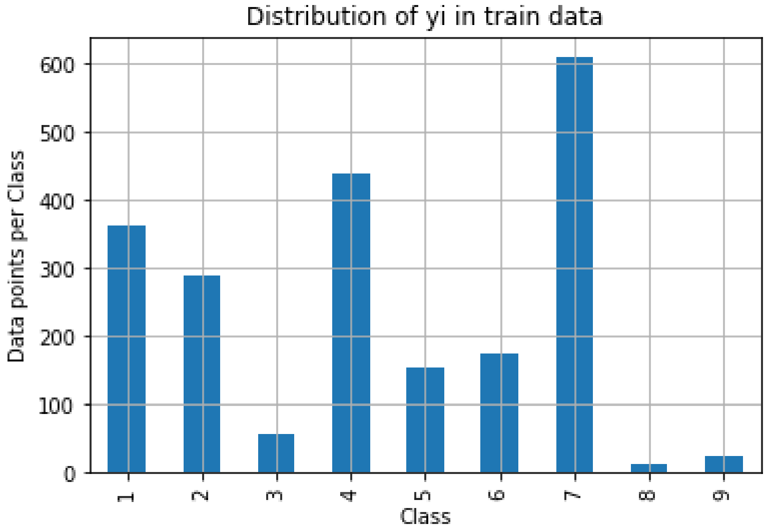
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

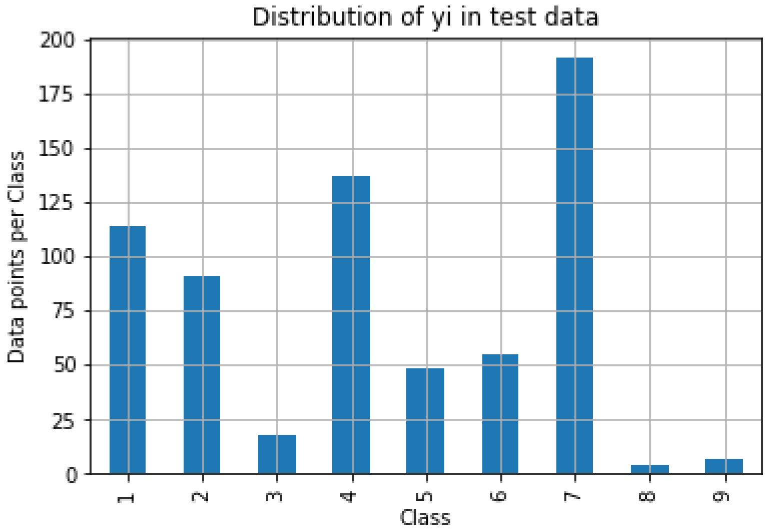
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [13]:

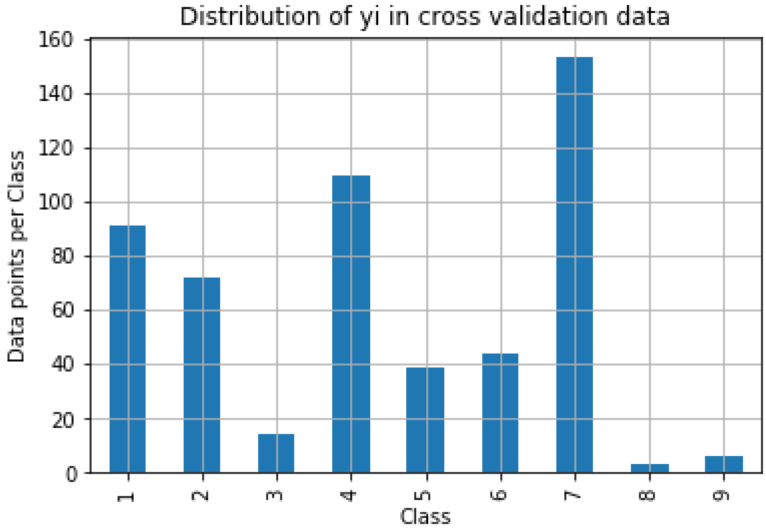
```
# it returns a dict, keys as class labels and values as the number of data points in that c
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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


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

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

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

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



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



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



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

3.2 Prediction using a 'Random' Model

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

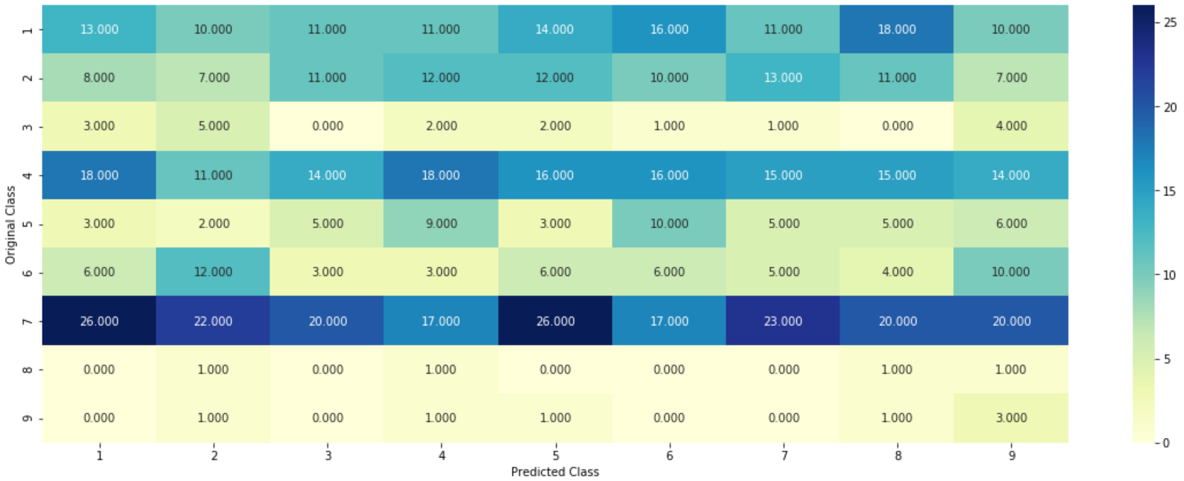
In [14]:

```
# 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 d
    # 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 d
    # 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()
```

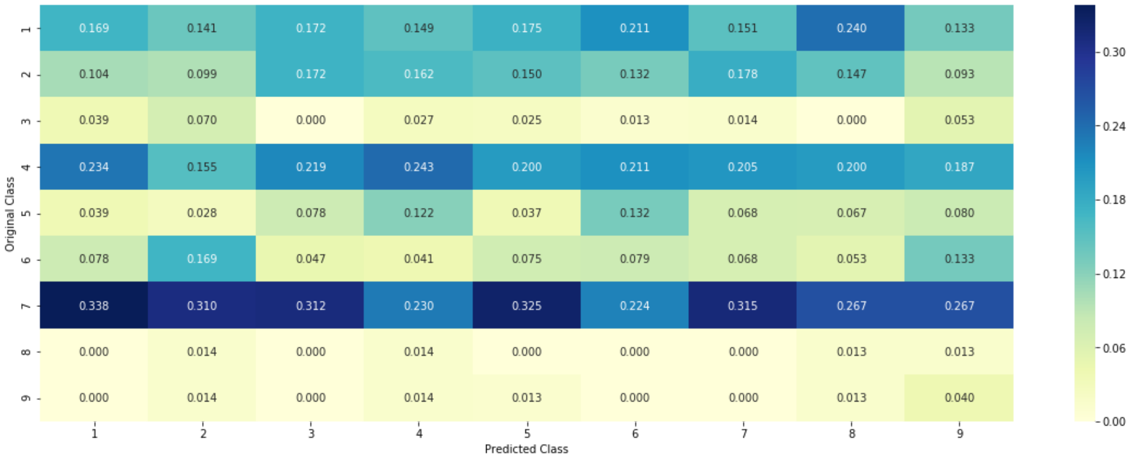
In [15]:

```
# 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)
```

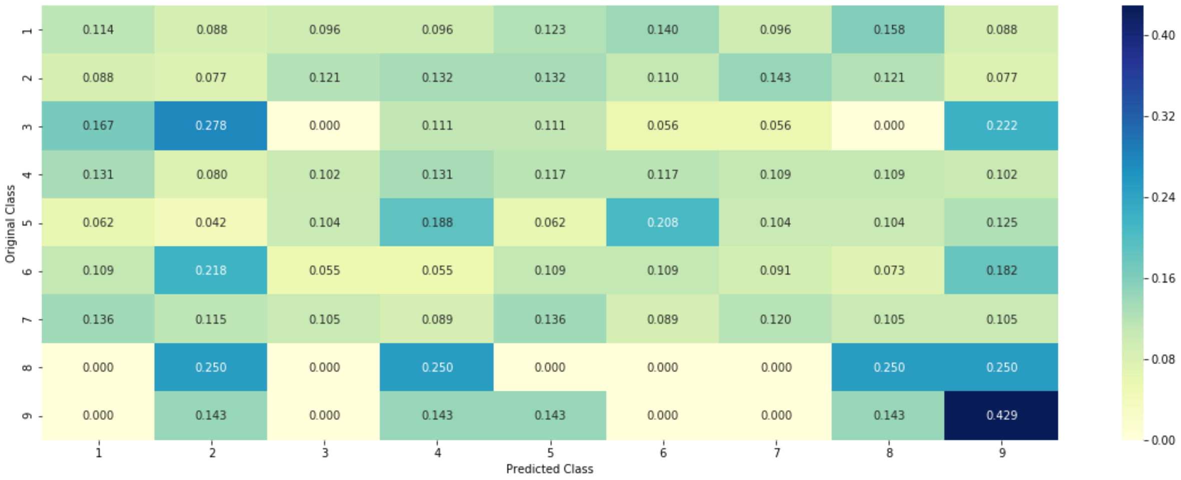
```
Log loss on Cross Validation Data using Random Model 2.508000146687834
Log loss on Test Data using Random Model 2.442771883618809
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [16]:

```
# 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 [17]:

```
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 : 231
BRCA1     166
TP53      112
EGFR       95
BRCA2      82
PTEN       80
KIT        64
BRAF       55
ERBB2      45
PIK3CA     44
ALK        44
Name: Gene, dtype: int64
```

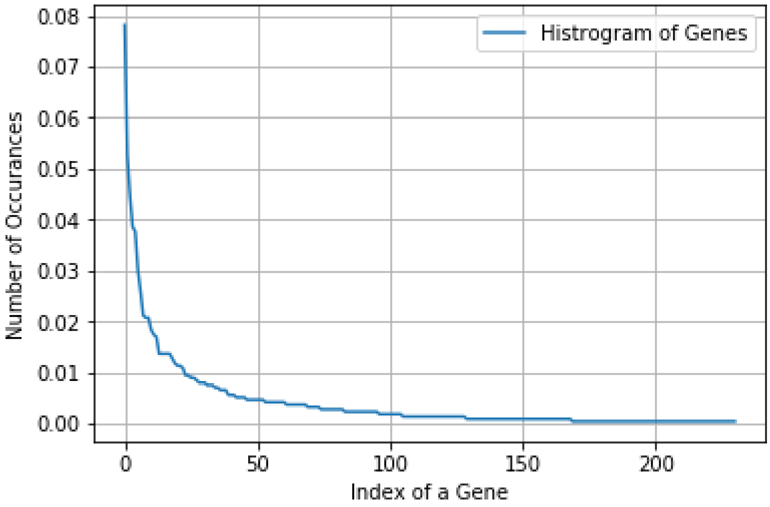
In [18]:

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

```
Ans: There are 231 different categories of genes in the train data, and they
are distibuted as follows
```

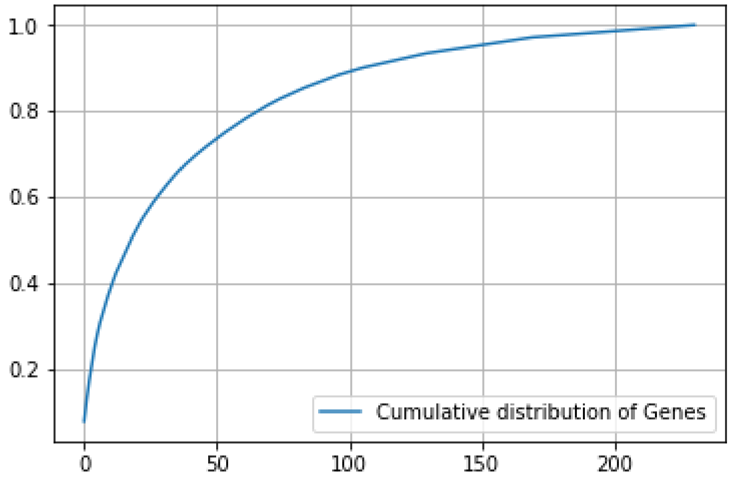
In [19]:

```
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 [20]:

```
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 [21]:

```
#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 [22]:

```
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 [23]:

```
# one-hot encoding of Gene feature.
from sklearn.feature_extraction.text import TfidfVectorizer

gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [24]:

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

Out[24]:

```
1616       VHL
1045      TSC2
677     CDKN2A
939     PDGFRB
1317      MLH1
Name: Gene, dtype: object
```

In [25]:

```
gene_vectorizer.get_feature_names()
```

Out[25]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atrx',
 'aurka',
 'aurkb'.
```

In [26]:

```
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, 231)
```

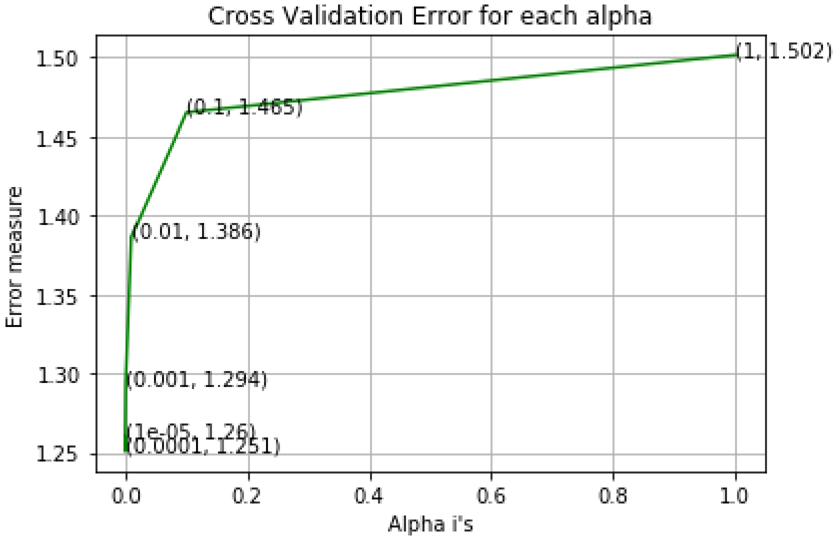
Q4. How good is this gene feature in predicting y_i?

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

In [27]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/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: 1.2600169501405563
For values of alpha =  0.0001 The log loss is: 1.2507862219100725
For values of alpha =  0.001 The log loss is: 1.2935501335729735
For values of alpha =  0.01 The log loss is: 1.3864002691746993
For values of alpha =  0.1 The log loss is: 1.4651503326868276
For values of alpha =  1 The log loss is: 1.5015247189975733
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9754617051477686
For values of best alpha =  0.0001 The cross validation log loss is: 1.25078
62219100725
For values of best alpha =  0.0001 The test log loss is: 1.2038400408929544
```

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

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

In [28]:

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

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  231  ge
nes in train dataset?
Ans
1. In test data 641 out of 665 : 96.39097744360903
2. In cross validation data 515 out of  532 : 96.80451127819549
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [29]:

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

```
Number of Unique Variations : 1941
Truncating_Mutations    59
Deletion                43
Amplification           39
Fusions                 23
Overexpression           5
Q61H                     3
P34R                     2
A146V                    2
G35R                     2
G67R                     2
Name: Variation, dtype: int64
```

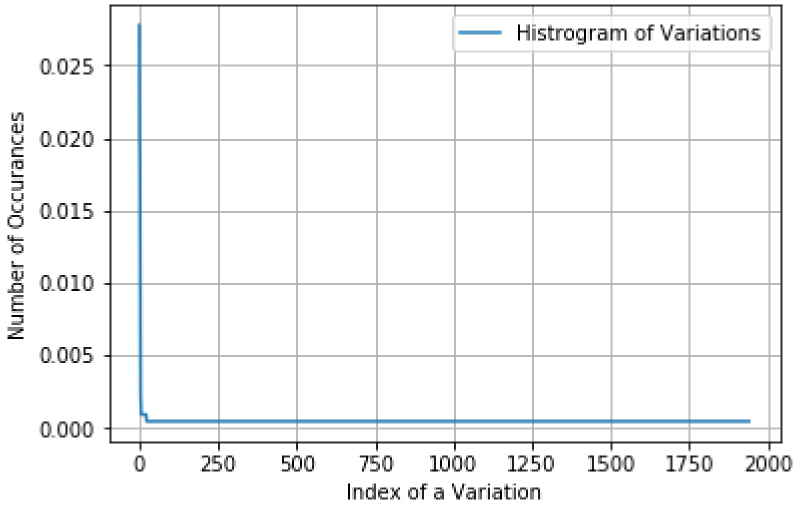
In [30]:

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

```
Ans: There are 1941 different categories of variations in the train data, an
d they are distibuted as follows
```

In [31]:

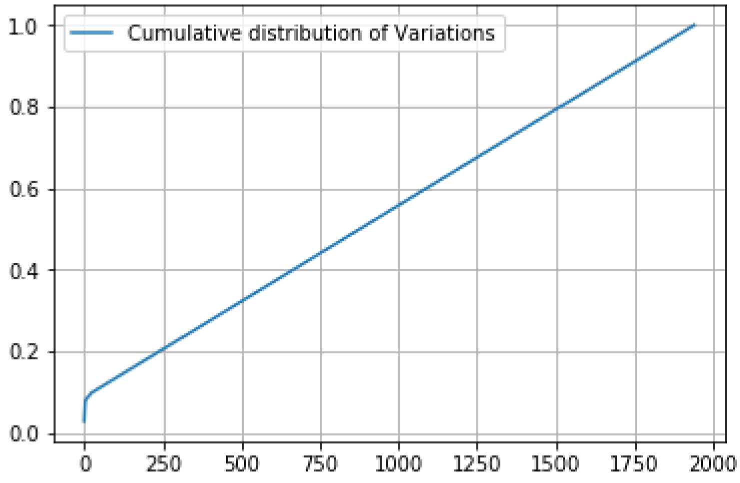
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [32]:

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

```
[0.02777778 0.0480226  0.06638418 ... 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 [33]:

```
# 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 [34]:

```
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 [35]:
```

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
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 [36]:
```

```
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, 1971)
```

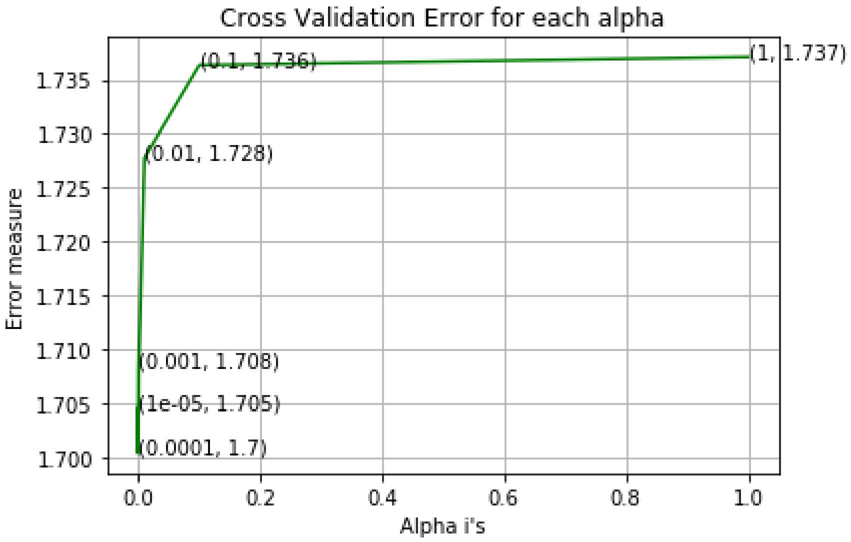
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [37]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/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: 1.704522952511729
For values of alpha =  0.0001 The log loss is: 1.7003354036851233
For values of alpha =  0.001 The log loss is: 1.7083089974985848
For values of alpha =  0.01 The log loss is: 1.7277449291315998
For values of alpha =  0.1 The log loss is: 1.7363455616266876
For values of alpha =  1 The log loss is: 1.737107470260399
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7324503557241041
For values of best alpha =  0.0001 The cross validation log loss is: 1.70033
54036851233
For values of best alpha =  0.0001 The test log loss is: 1.696348487968619
```

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

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

In [38]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " 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  1941  genes in test and cros
s validation data sets?
Ans
1. In test data 76 out of 665 : 11.428571428571429
2. In cross validation data 59 out of  532 : 11.090225563909774
```

3.2.3 Univariate Analysis on Text Feature

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

In [39]:

```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
```

In [40]:

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

In [41]:

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

Out[41]:

```
1616    eukaryotic chaperonin tric cct mediates foldin...
1045    tuberous sclerosis tsc autosomal dominant diso...
677     ink4a arf tumor suppressor locus implicated se...
939     chronic myeloproliferative disorders cmpd clon...
1317    mismatch repair factors prominent role surveyi...
Name: TEXT, dtype: object
```

In [42]:

```
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature name
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

def top_mean_feats(Xtr, features, min_tfidf=0.1, grp_ids=None, top_n=25):
    ''' Return the top n features that on average are most important amongst documents in r
        indentified by indices in grp_ids. '''
    if grp_ids:
        D = Xtr[grp_ids].toarray()
    else:
        D = Xtr.toarray()

    D[D < min_tfidf] = 0
    tfidf_means = np.mean(D, axis=0)
    return top_tfidf_feats(tfidf_means, features, top_n)
```

In [44]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])

# getting top 1000 feature names (words)
train_text_features = top_mean_feats(train_text_feature_onehotCoding,
                                                text_vectorizer.get_feature_names(),
                                                top_n=1000)['feature'].tolist()
```

In [45]:

```
# 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
train_text_fea_counts
```

Out[45]:

```
array([8.74741445, 9.17101358, 0.04050112, ..., 0.0364807 , 0.01630126,
       0.08046506])
```

In [46]:

```
# 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 [52]:

```
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer(min_df=3, ngram_range=(1,2), lowercase=False, binary=Tru
train_text_feature_onehotCoding_count_vectorizer = count_vectorizer.fit_transform(train_df[
print("Shape of matrix after one hot encodig ",train_text_feature_onehotCoding_count_vector
```

```
Shape of matrix after one hot encodig  (2124, 776298)
```

In [53]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding_count_vectorizer = normalize(train_text_feature_onehotCodir

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding_count_vectorizer = count_vectorizer.transform(test_df['TEXT'
# don't forget to normalize every feature
test_text_feature_onehotCoding_count_vectorizer = normalize(test_text_feature_onehotCoding_

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

In [54]:

```
print(train_text_feature_onehotCoding_count_vectorizer.shape)
print(test_text_feature_onehotCoding_count_vectorizer.shape)
print(cv_text_feature_onehotCoding_count_vectorizer.shape)
```

```
(2124, 776298)
(665, 776298)
(532, 776298)
```

In [47]:

```
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 [48]:

```
#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 [49]:

```
# 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 [55]:

```
# 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 [56]:

```
#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 [57]:

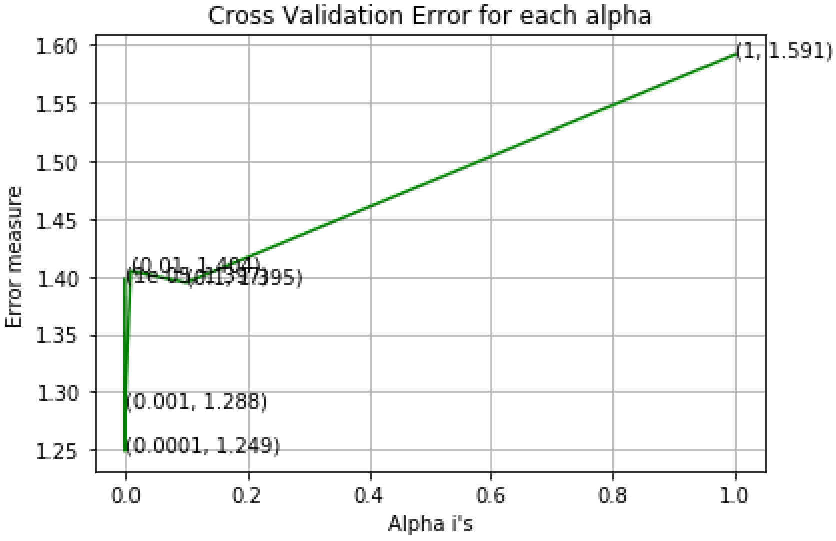
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
0.07778149460590124: 1, 0.07763658772123823: 1, 0.07762327390240781: 1,
0.07716687141278758: 1, 0.0768439274414953: 1, 0.07678986214221291: 1, 0.
0767808864911124: 1, 0.07404429893260094: 1, 0.07373220855881032: 1, 0.07
28309485997411: 1, 0.07277552722717694: 1, 0.07259594601146176: 1, 0.0724
0798200785978: 1, 0.07232902778583075: 1, 0.07203891214583771: 1, 0.07182
122815215515: 1, 0.07171047514626255: 1, 0.0704916225919203: 1, 0.0701099
458941825: 1, 0.06975690343468469: 1, 0.06948963064290126: 1, 0.069394307
17689116: 1, 0.06934486662680751: 1, 0.06903848351675562: 1, 0.0689016958
6029629: 1, 0.06873211688458081: 1, 0.06850729322950523: 1, 0.06766459953
213493: 1, 0.06761709833779048: 1, 0.06739379771844285: 1, 0.066936087383
36135: 1, 0.06647457859583478: 1, 0.06626719856248221: 1, 0.0659763884938
8872: 1, 0.06574105048048245: 1, 0.06570483456514666: 1, 0.06567013616195
504: 1, 0.06546485349359919: 1, 0.06531078625118: 1, 0.06515671274562171:
1, 0.06481739583091095: 1, 0.06473101781137564: 1, 0.06464626220741646:
1, 0.06461579975583175: 1, 0.0645889637674938: 1, 0.0641581684614308: 1,
0.06399855541182041: 1, 0.06393971976542658: 1, 0.0637059654043394: 1, 0.
06368498078302497: 1, 0.06357505528534207: 1, 0.06331807184139288: 1, 0.0
6329323059566441: 1, 0.0631897709148575: 1, 0.06237984743260923 5: 1, 0.06
1943868095691496: 1, 0.061932262364955595: 1, 0.06101959836596885: 1, 0.0
6083968470211 7494: 1, 0.06063693381486373: 1, 0.0590237273759755: 1, 0.05
```

In [58]:

```
# Train a Logistic regression+Calibration model using text features which are one-hot encod
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_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)
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: 1.397329741255539
For values of alpha =  0.0001 The log loss is: 1.2490647781710014
For values of alpha =  0.001 The log loss is: 1.2875346162198418
For values of alpha =  0.01 The log loss is: 1.4044488789093172
For values of alpha =  0.1 The log loss is: 1.3945972852257766
For values of alpha =  1 The log loss is: 1.5911783413353846
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6486587654844075
For values of best alpha =  0.0001 The cross validation log loss is: 1.24906
47781710014
For values of best alpha =  0.0001 The test log loss is: 1.129475453204576
```

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

Ans. Yes, it seems like!

In [59]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    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 [60]:

```
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")
```

```
22.4 % of word of test data appeared in train data
22.7 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [61]:

```
#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 [62]:

```
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 [86]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(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 [75]:

```
# 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_variatic
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_respc
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 [76]:

```

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

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

cv_x_onehotCoding_count_vectorizer = hstack((cv_gene_var_onehotCoding, cv_text_feature_oneh
cv_y = np.array(list(cv_df['Class']))
```

In [77]:

```
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, 55508)
(number of data points * number of features) in test data =  (665, 55508)
(number of data points * number of features) in cross validation data = (53
2, 55508)
```

In [78]:

```
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)
```

In [79]:

```
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_c
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, 778500)
(number of data points * number of features) in test data =  (665, 778500)
(number of data points * number of features) in cross validation data = (53
2, 778500)
```

4.1. Base Line Model

4.1.1. Naive Bayes

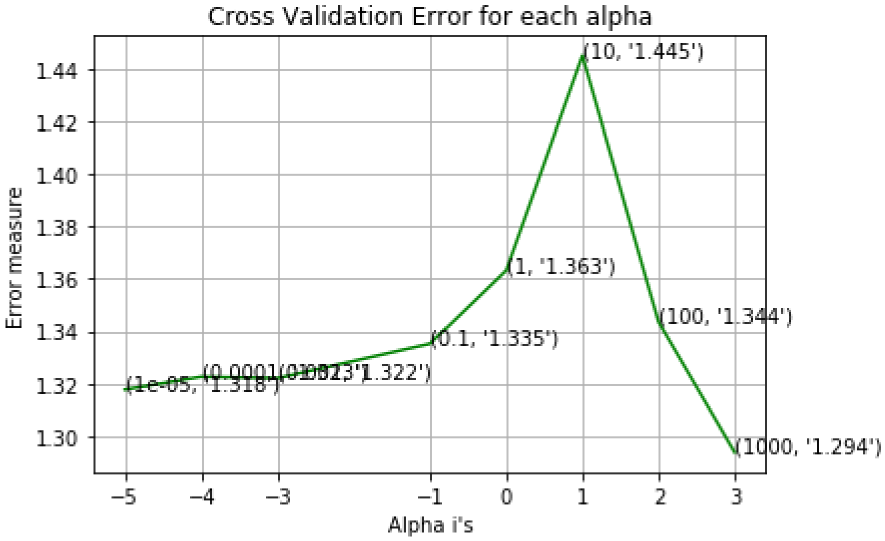
4.1.1.1. Hyper parameter tuning

In [80]:

```
# 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 : 1.3177662714995957
for alpha = 0.0001
Log Loss : 1.3226458203930707
for alpha = 0.001
Log Loss : 1.3220895788473552
for alpha = 0.1
Log Loss : 1.335200997469603
for alpha = 1
Log Loss : 1.3632616234184063
for alpha = 10
Log Loss : 1.4449701295926531
for alpha = 100
Log Loss : 1.3436565439225672
for alpha = 1000
Log Loss : 1.2939359945798898
```



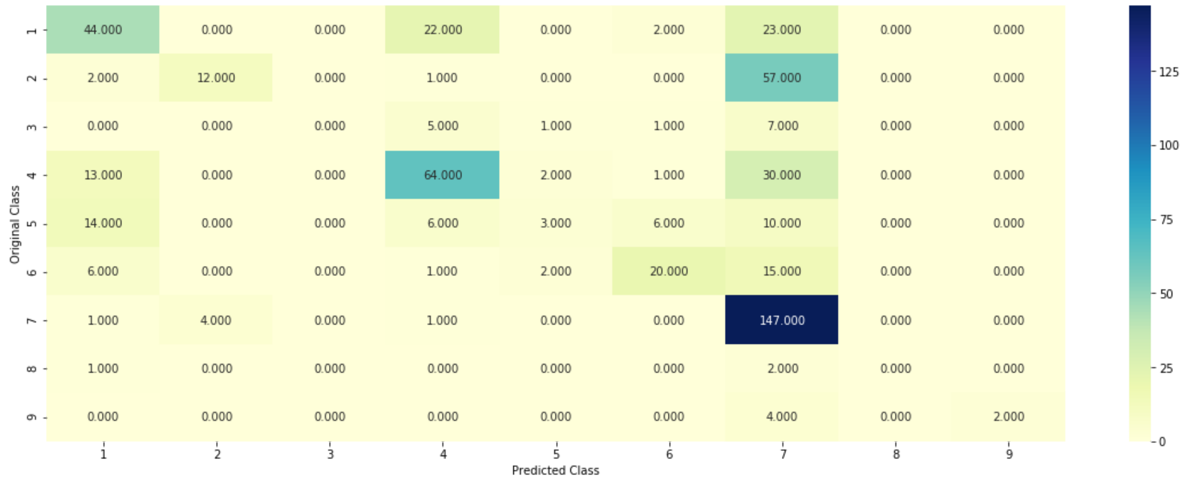
```
For values of best alpha =  1000 The train log loss is: 0.9090769572693448
For values of best alpha =  1000 The cross validation log loss is: 1.2939359
945798898
For values of best alpha =  1000 The test log loss is: 1.202351176225535
```

4.1.1.2. Testing the model with best hyper paramters

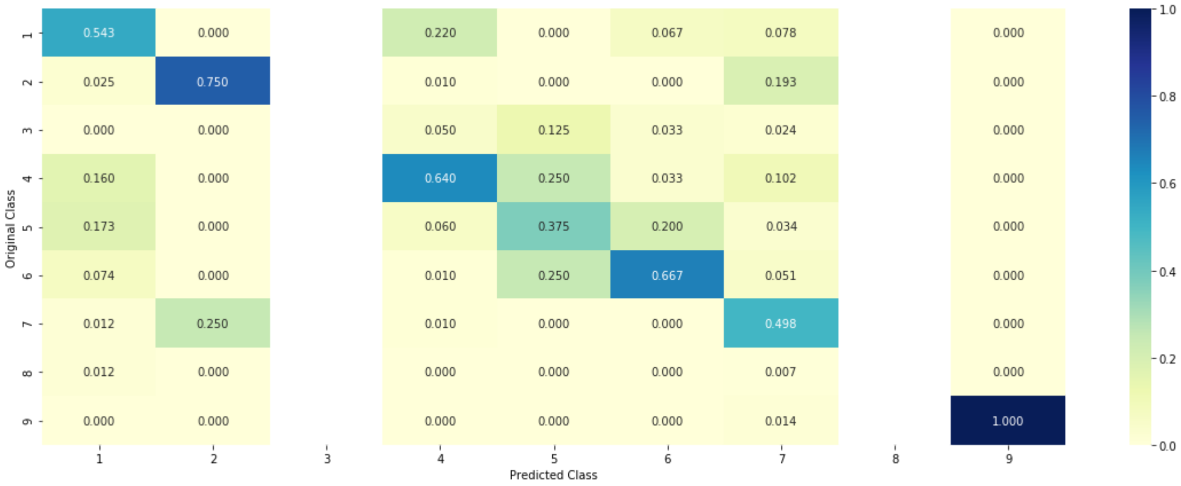
In [81]:

```
# 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()))
```

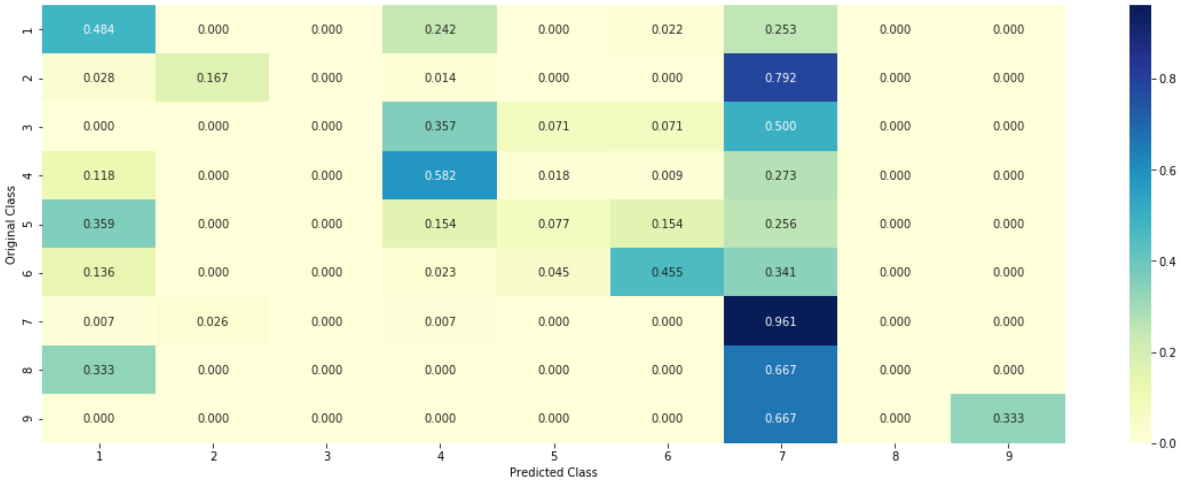
```
Log Loss : 1.2939359945798898
Number of missclassified point : 0.45112781954887216
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [87]:

```
test_point_index = 1
no_feature = 1000
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],
                     test_df['Gene'].iloc[test_point_index],
                     test_df['Variation'].iloc[test_point_index],
                     no_feature)
```

```
937 Text feature [genetic] present in test data point [True]
939 Text feature [rather] present in test data point [True]
941 Text feature [preferentially] present in test data point [True]
949 Text feature [evaluate] present in test data point [True]
950 Text feature [34] present in test data point [True]
953 Text feature [removed] present in test data point [True]
954 Text feature [prognostic] present in test data point [True]
956 Text feature [61] present in test data point [True]
959 Text feature [26] present in test data point [True]
967 Text feature [though] present in test data point [True]
968 Text feature [striking] present in test data point [True]
969 Text feature [rates] present in test data point [True]
975 Text feature [potentially] present in test data point [True]
976 Text feature [residues] present in test data point [True]
978 Text feature [listed] present in test data point [True]
980 Text feature [matched] present in test data point [True]
981 Text feature [accounts] present in test data point [True]
982 Text feature [ten] present in test data point [True]
985 Text feature [proliferate] present in test data point [True]
992 Text feature [deletion] present in test data point [True]
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [88]:

```
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'].ilc
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.260e-02 1.454e-01 4.000e-03 3.930e-02 2.7
60e-02 2.404e-01 5.086e-01
  1.600e-03 4.000e-04]]
Actual Class : 6
--------------------------------------------------
15 Text feature [cells] present in test data point [True]
16 Text feature [kinase] present in test data point [True]
17 Text feature [activated] present in test data point [True]
18 Text feature [activation] present in test data point [True]
19 Text feature [cell] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [presence] present in test data point [True]
24 Text feature [contrast] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [phosphorylation] present in test data point [True]
27 Text feature [factor] present in test data point [True]
28 Text feature [shown] present in test data point [True]
29 Text feature [signaling] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [suggest] present in test data point [True]
36 Text feature [recently] present in test data point [True]
37 Text feature [tyrosine] present in test data point [True]
38 Text feature [independent] present in test data point [True]
39 Text feature [however] present in test data point [True]
40 Text feature [higher] present in test data point [True]
42 Text feature [compared] present in test data point [True]
43 Text feature [10] present in test data point [True]
44 Text feature [found] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [similar] present in test data point [True]
47 Text feature [mechanism] present in test data point [True]
48 Text feature [addition] present in test data point [True]
49 Text feature [showed] present in test data point [True]
50 Text feature [treatment] present in test data point [True]
53 Text feature [increased] present in test data point [True]
54 Text feature [mutations] present in test data point [True]
55 Text feature [various] present in test data point [True]
56 Text feature [mutant] present in test data point [True]
57 Text feature [sensitive] present in test data point [True]
58 Text feature [potential] present in test data point [True]
59 Text feature [well] present in test data point [True]
61 Text feature [inhibition] present in test data point [True]
62 Text feature [may] present in test data point [True]
64 Text feature [consistent] present in test data point [True]
66 Text feature [observed] present in test data point [True]
```

```
68 Text feature [including] present in test data point [True]
69 Text feature [constitutive] present in test data point [True]
70 Text feature [enhanced] present in test data point [True]
72 Text feature [pathways] present in test data point [True]
73 Text feature [total] present in test data point [True]
74 Text feature [absence] present in test data point [True]
76 Text feature [antibodies] present in test data point [True]
77 Text feature [using] present in test data point [True]
78 Text feature [reported] present in test data point [True]
79 Text feature [furthermore] present in test data point [True]
81 Text feature [molecular] present in test data point [True]
82 Text feature [activate] present in test data point [True]
84 Text feature [described] present in test data point [True]
85 Text feature [respectively] present in test data point [True]
86 Text feature [inhibitors] present in test data point [True]
87 Text feature [antibody] present in test data point [True]
90 Text feature [mutation] present in test data point [True]
91 Text feature [activating] present in test data point [True]
92 Text feature [identified] present in test data point [True]
94 Text feature [followed] present in test data point [True]
95 Text feature [fig] present in test data point [True]
96 Text feature [two] present in test data point [True]
97 Text feature [domain] present in test data point [True]
98 Text feature [new] present in test data point [True]
99 Text feature [expressed] present in test data point [True]
Out of the top  100  features  66 are present in query point
```

4.2. K Nearest Neighbour Classification

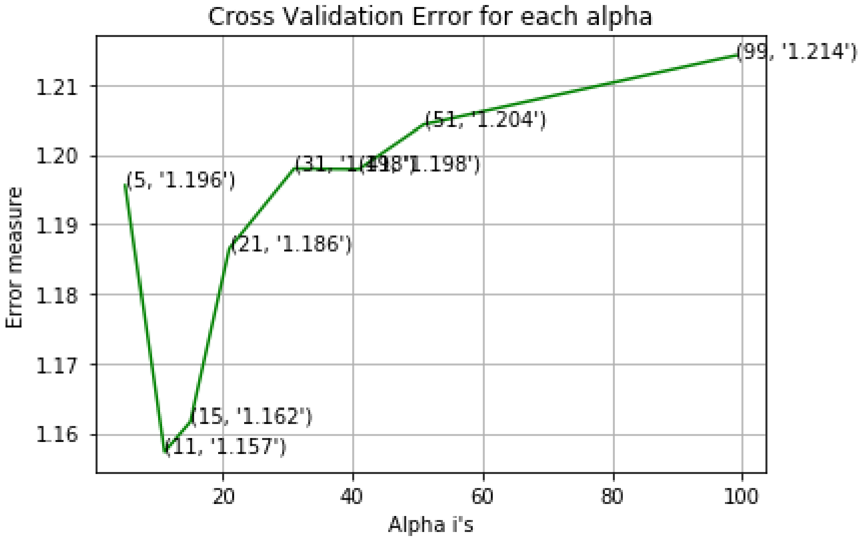
4.2.1. Hyper parameter tuning

In [89]:

```
# 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)
    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])
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 : 1.1955560617871481
for alpha = 11
Log Loss : 1.1573253403761115
for alpha = 15
Log Loss : 1.1616630855724301
for alpha = 21
Log Loss : 1.1864931915306138
for alpha = 31
Log Loss : 1.1979884912350671
for alpha = 41
Log Loss : 1.1978937536140934
for alpha = 51
Log Loss : 1.204278619851912
for alpha = 99
Log Loss : 1.21417081573141
```



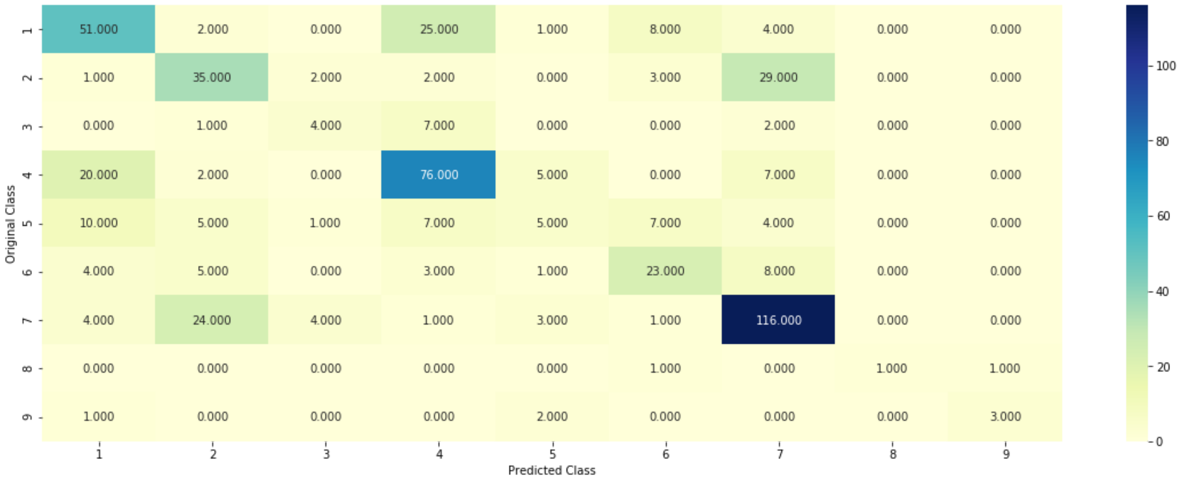
```
For values of best alpha =  11 The train log loss is: 0.6323871078588027
For values of best alpha =  11 The cross validation log loss is: 1.157325340
3761115
For values of best alpha =  11 The test log loss is: 1.051555369629674
```

4.2.2. Testing the model with best hyper paramters

In [90]:

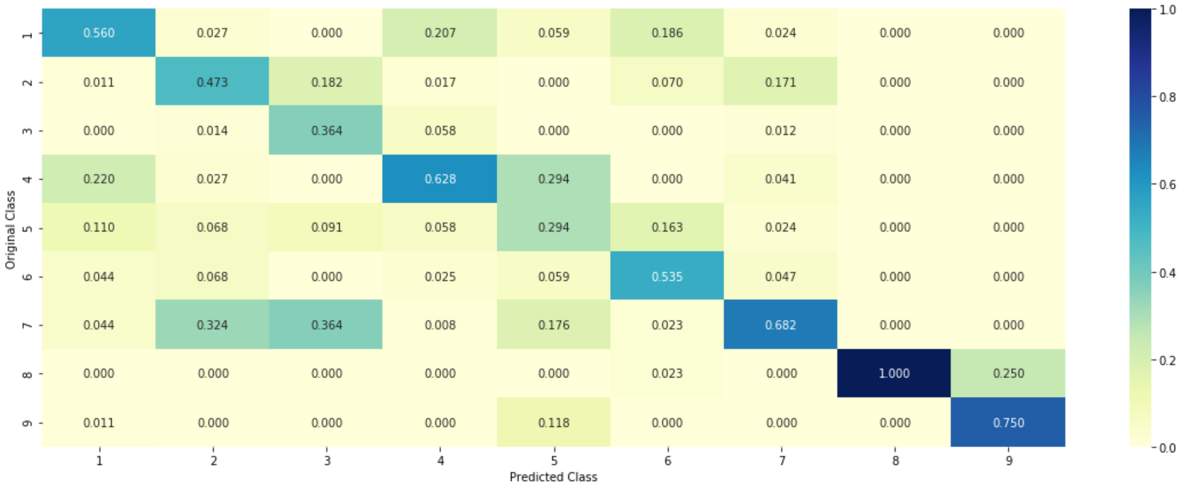
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/genera
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# 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])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_
```

```
Log loss : 1.1573253403761115
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```



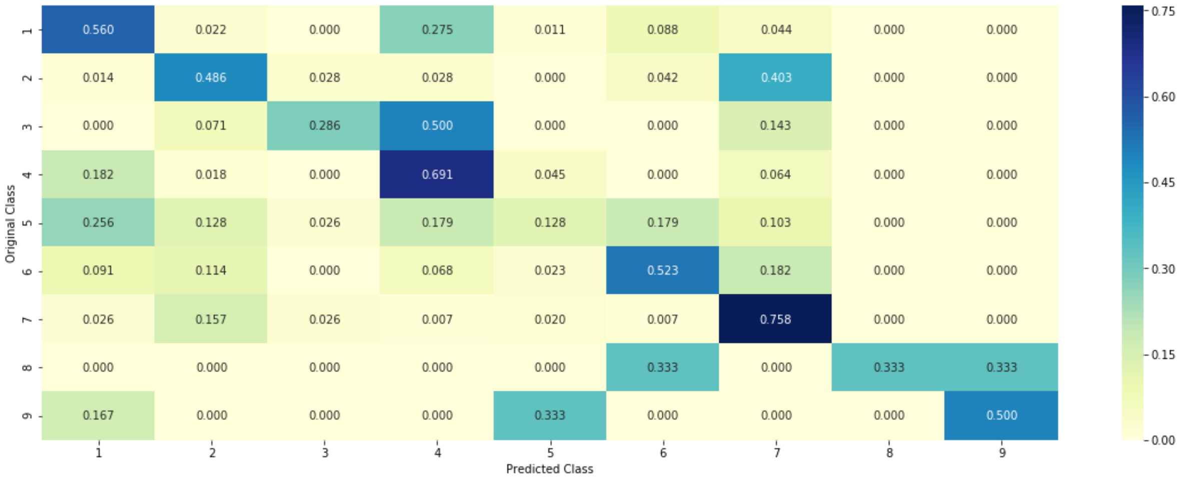
	1	2	3	4	5	6	7	8	9
1	51.000	2.000	0.000	25.000	1.000	8.000	4.000	0.000	0.000
2	1.000	35.000	2.000	2.000	0.000	3.000	29.000	0.000	0.000
3	0.000	1.000	4.000	7.000	0.000	0.000	2.000	0.000	0.000
4	20.000	2.000	0.000	76.000	5.000	0.000	7.000	0.000	0.000
5	10.000	5.000	1.000	7.000	5.000	7.000	4.000	0.000	0.000
6	4.000	5.000	0.000	3.000	1.000	23.000	8.000	0.000	0.000
7	4.000	24.000	4.000	1.000	3.000	1.000	116.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	1.000
9	1.000	0.000	0.000	0.000	2.000	0.000	0.000	0.000	3.000

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



	1	2	3	4	5	6	7	8	9
1	0.560	0.027	0.000	0.207	0.059	0.186	0.024	0.000	0.000
2	0.011	0.473	0.182	0.017	0.000	0.070	0.171	0.000	0.000
3	0.000	0.014	0.364	0.058	0.000	0.000	0.012	0.000	0.000
4	0.220	0.027	0.000	0.628	0.294	0.000	0.041	0.000	0.000
5	0.110	0.068	0.091	0.058	0.294	0.163	0.024	0.000	0.000
6	0.044	0.068	0.000	0.025	0.059	0.535	0.047	0.000	0.000
7	0.044	0.324	0.364	0.008	0.176	0.023	0.682	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.023	0.000	1.000	0.250
9	0.011	0.000	0.000	0.000	0.118	0.000	0.000	0.000	0.750

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



4.2.3.Sample Query point -1

In [91]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), 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 : 6
Actual Class : 2
The  11  nearest neighbours of the test points belongs to classes [2 7 7 2 2
2 2 2 1 7 7]
Fequency of nearest points : Counter({2: 6, 7: 4, 1: 1})
```

4.2.4. Sample Query Point-2

In [92]:

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

test_point_index = 100

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

```
Predicted Class : 6
Actual Class : 6
the k value for knn is 11 and the nearest neighbours of the test points belo
ngs to classes [2 6 6 6 6 6 6 6 2 7 7]
Fequency of nearest points : Counter({6: 7, 2: 2, 7: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

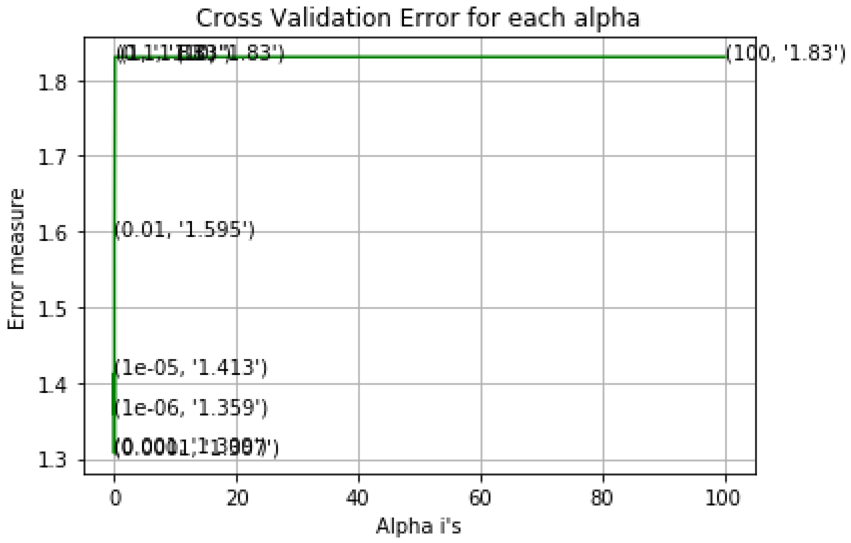
4.3.1.1. Hyper paramter tuning

In [94]:

```

# 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='l1', 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 : 1.3593016788406298
for alpha = 1e-05
Log Loss : 1.4127715971522592
for alpha = 0.0001
Log Loss : 1.3069800341143707
for alpha = 0.001
Log Loss : 1.3091049031448472
for alpha = 0.01
Log Loss : 1.5949839980310916
for alpha = 0.1
Log Loss : 1.830259980694803
for alpha = 1
Log Loss : 1.8302599806230293
for alpha = 10
Log Loss : 1.8302599806220337
for alpha = 100
Log Loss : 1.8302599806220095
```



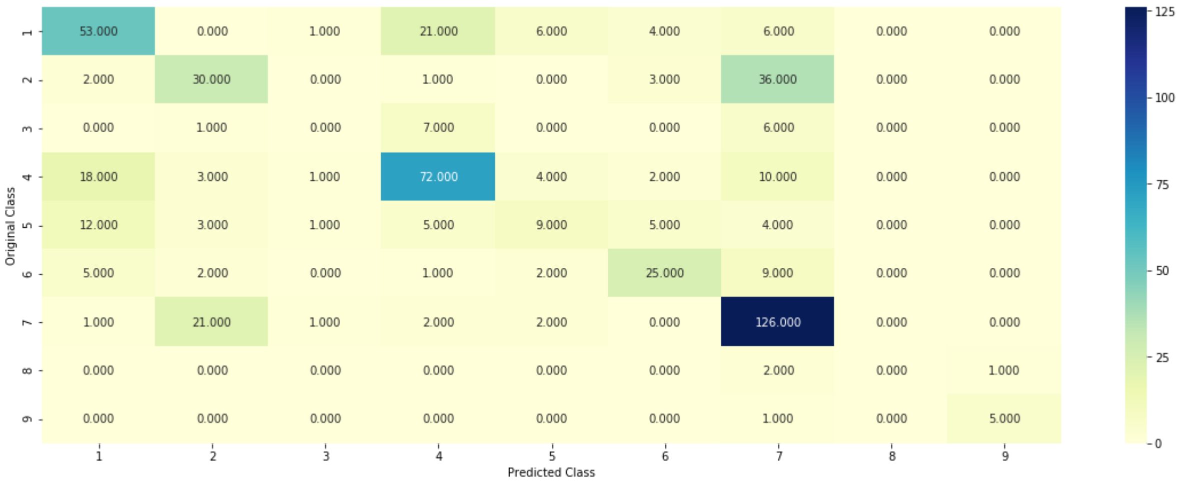
```
For values of best alpha =  0.0001 The train log loss is: 0.5510473434283524
For values of best alpha =  0.0001 The cross validation log loss is: 1.19213
09231505126
For values of best alpha =  0.0001 The test log loss is: 1.0958916103965863
```

4.3.1.2. Testing the model with best hyper paramters

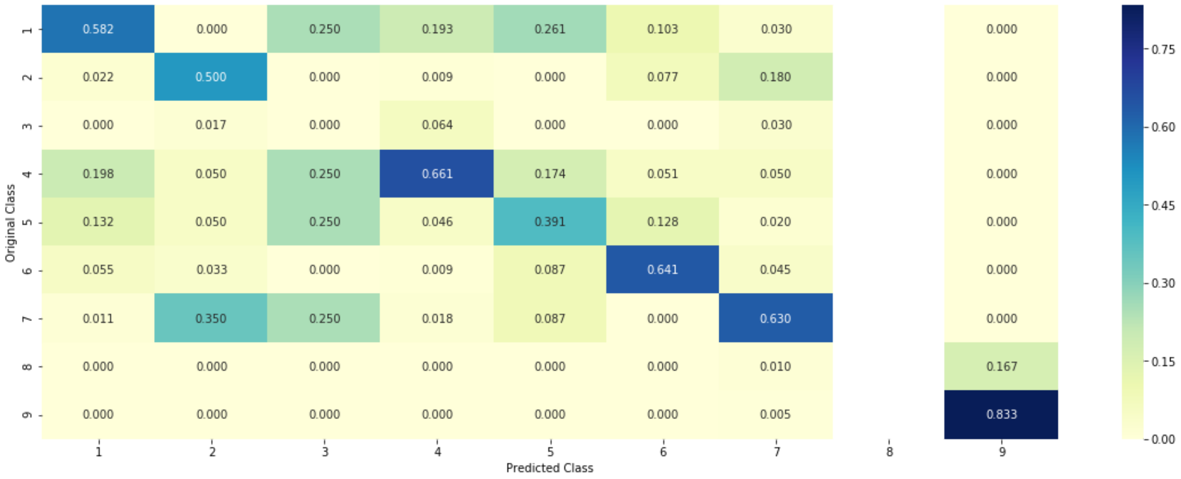
In [95]:

```
# 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
```

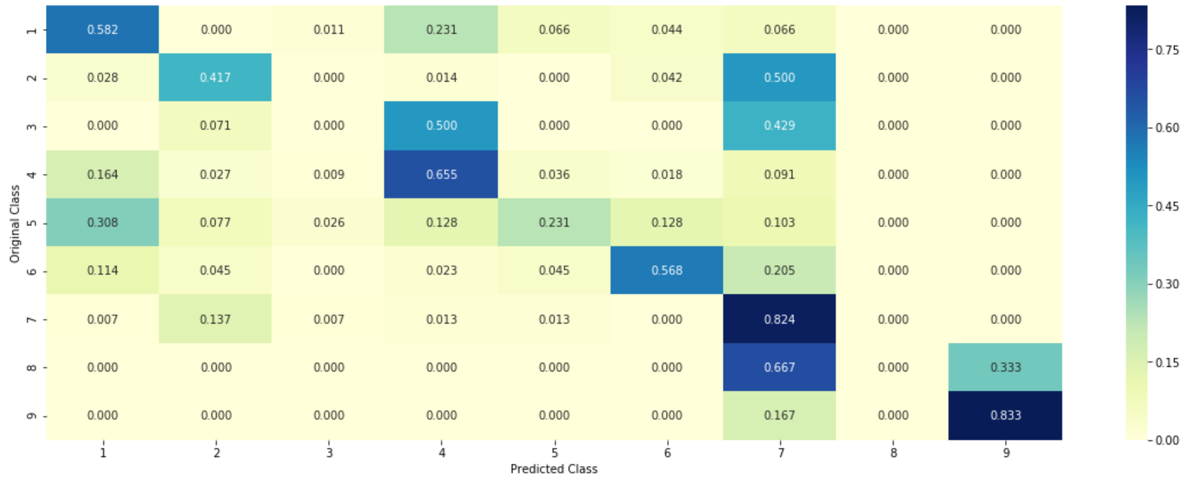
```
Log loss : 1.1921309231505126
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [96]:

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

4.3.1.3.1. Correctly Classified point

In [100]:

```
# 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: [[0.0052 0.44   0.0039 0.0122 0.0057 0.0011
0.5232 0.004  0.0048]]
Actual Class : 2
--------------------------------------------------
156 Text feature [activated] present in test data point [True]
163 Text feature [constitutive] present in test data point [True]
193 Text feature [activation] present in test data point [True]
219 Text feature [murine] present in test data point [True]
286 Text feature [nf] present in test data point [True]
290 Text feature [expressing] present in test data point [True]
340 Text feature [activate] present in test data point [True]
351 Text feature [phosphorylation] present in test data point [True]
356 Text feature [inhibitor] present in test data point [True]
368 Text feature [proportional] present in test data point [True]
384 Text feature [transformation] present in test data point [True]
397 Text feature [transforming] present in test data point [True]
403 Text feature [transformed] present in test data point [True]
407 Text feature [205] present in test data point [True]
446 Text feature [synergy] present in test data point [True]
457 Text feature [120] present in test data point [True]
488 Text feature [phospho] present in test data point [True]
Out of the top  500  features  17 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [101]:

```
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 : 6
Predicted Class Probabilities: [[0.0072 0.1141 0.0104 0.008  0.0122 0.7755
0.0622 0.0043 0.0061]]
Actual Class : 6
--------------------------------------------------
346 Text feature [s783] present in test data point [True]
420 Text feature [singlet] present in test data point [True]
423 Text feature [g776] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2. Without Class balancing

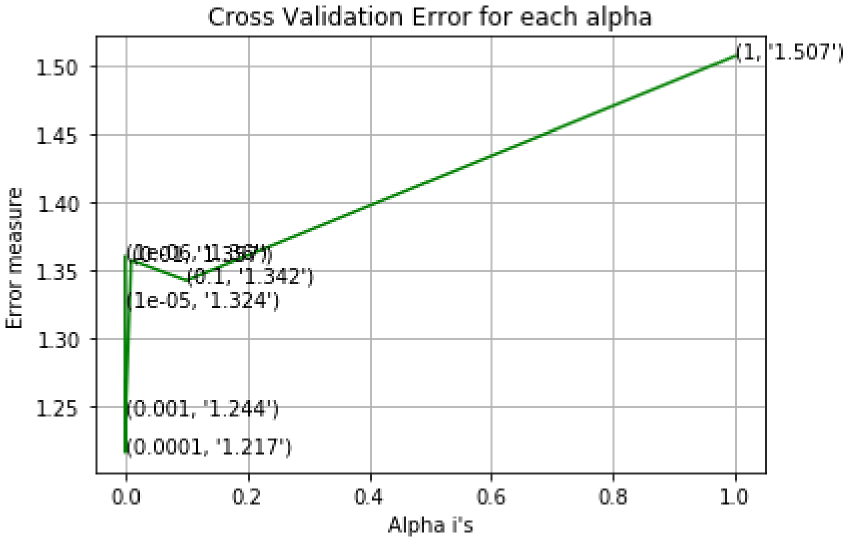
4.3.2.1. Hyper paramter tuning

In [102]:

```
# 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 : 1.3599680366418216
for alpha = 1e-05
Log Loss : 1.323523538407815
for alpha = 0.0001
Log Loss : 1.2165349624175101
for alpha = 0.001
Log Loss : 1.2441447342398333
for alpha = 0.01
Log Loss : 1.357196675740398
for alpha = 0.1
Log Loss : 1.342413824871523
for alpha = 1
Log Loss : 1.506819686753777
```



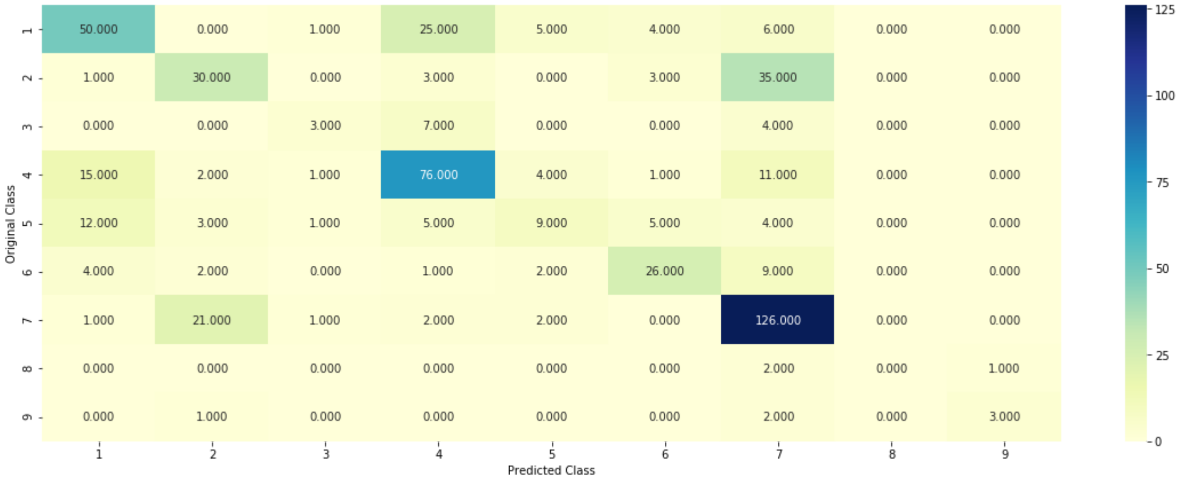
```
For values of best alpha =  0.0001 The train log loss is: 0.5105395982593547
For values of best alpha =  0.0001 The cross validation log loss is: 1.21653
49624175101
For values of best alpha =  0.0001 The test log loss is: 1.1000613732973816
```

4.3.2.2. Testing model with best hyper parameters

In [103]:

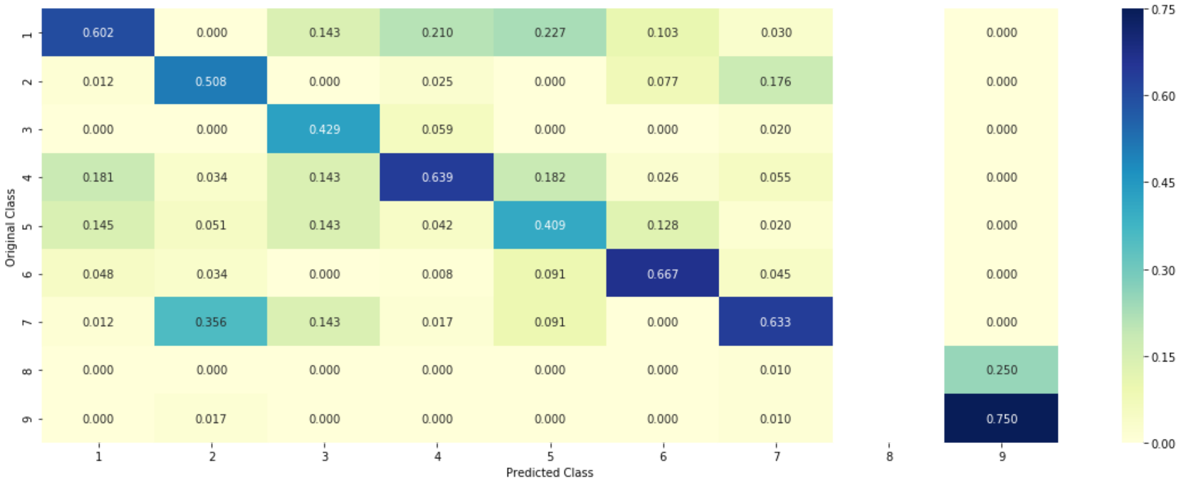
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, 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 : 1.2165349624175101
Number of mis-classified points : 0.39285714285714285
-------------------- Confusion matrix --------------------
```



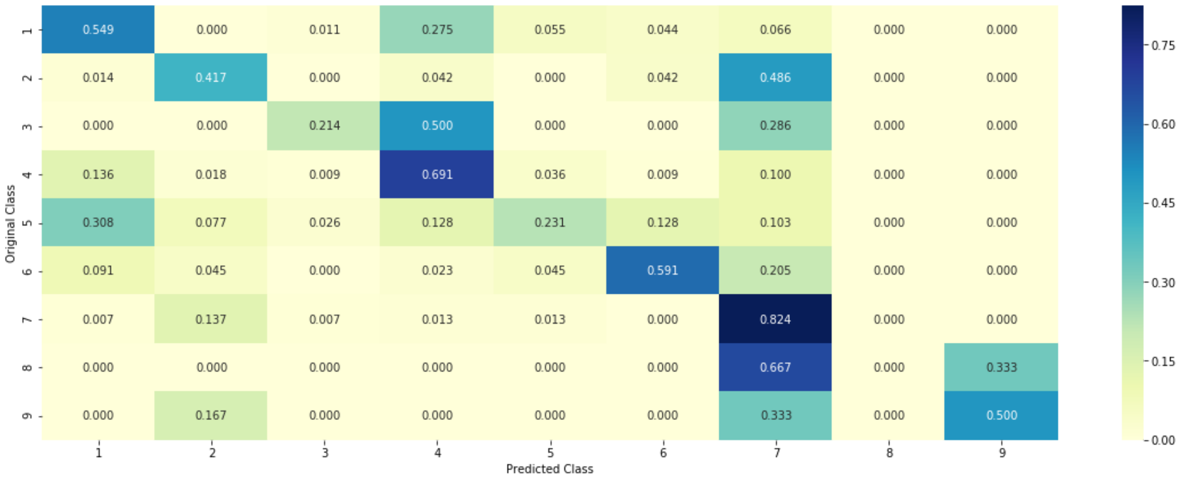
	1	2	3	4	5	6	7	8	9
1	50.000	0.000	1.000	25.000	5.000	4.000	6.000	0.000	0.000
2	1.000	30.000	0.000	3.000	0.000	3.000	35.000	0.000	0.000
3	0.000	0.000	3.000	7.000	0.000	0.000	4.000	0.000	0.000
4	15.000	2.000	1.000	76.000	4.000	1.000	11.000	0.000	0.000
5	12.000	3.000	1.000	5.000	9.000	5.000	4.000	0.000	0.000
6	4.000	2.000	0.000	1.000	2.000	26.000	9.000	0.000	0.000
7	1.000	21.000	1.000	2.000	2.000	0.000	126.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000
9	0.000	1.000	0.000	0.000	0.000	0.000	2.000	0.000	3.000

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



	1	2	3	4	5	6	7	8	9
1	0.602	0.000	0.143	0.210	0.227	0.103	0.030		0.000
2	0.012	0.508	0.000	0.025	0.000	0.077	0.176		0.000
3	0.000	0.000	0.429	0.059	0.000	0.000	0.020		0.000
4	0.181	0.034	0.143	0.639	0.182	0.026	0.055		0.000
5	0.145	0.051	0.143	0.042	0.409	0.128	0.020		0.000
6	0.048	0.034	0.000	0.008	0.091	0.667	0.045		0.000
7	0.012	0.356	0.143	0.017	0.091	0.000	0.633		0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.010		0.250
9	0.000	0.017	0.000	0.000	0.000	0.000	0.010		0.750

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



4.3.2.3. Feature Importance, Correctly Classified point

In [104]:

```
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'].ilo
```

```
Predicted Class : 7
Predicted Class Probabilities: [[6.800e-03 4.351e-01 4.000e-04 1.730e-02 3.2
00e-03 8.000e-04 5.342e-01
  2.200e-03 1.000e-04]]
Actual Class : 2
--------------------------------------------------
153 Text feature [activated] present in test data point [True]
168 Text feature [constitutive] present in test data point [True]
188 Text feature [activation] present in test data point [True]
196 Text feature [murine] present in test data point [True]
303 Text feature [expressing] present in test data point [True]
337 Text feature [inhibitor] present in test data point [True]
344 Text feature [phosphorylation] present in test data point [True]
345 Text feature [activate] present in test data point [True]
362 Text feature [transformation] present in test data point [True]
406 Text feature [proportional] present in test data point [True]
424 Text feature [infiltration] present in test data point [True]
428 Text feature [phospho] present in test data point [True]
432 Text feature [120] present in test data point [True]
435 Text feature [205] present in test data point [True]
464 Text feature [transforming] present in test data point [True]
482 Text feature [nf] present in test data point [True]
487 Text feature [kinase] present in test data point [True]
Out of the top  500  features  17 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [105]:

```
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 : 6
Predicted Class Probabilities: [[8.800e-03 1.241e-01 2.400e-03 1.090e-02 8.4
00e-03 7.563e-01 8.550e-02
  3.000e-03 7.000e-04]]
Actual Class : 6
--------------------------------------------------
338 Text feature [s783] present in test data point [True]
400 Text feature [singlet] present in test data point [True]
423 Text feature [g776] present in test data point [True]
486 Text feature [salt] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.4. Linear Support Vector Machines

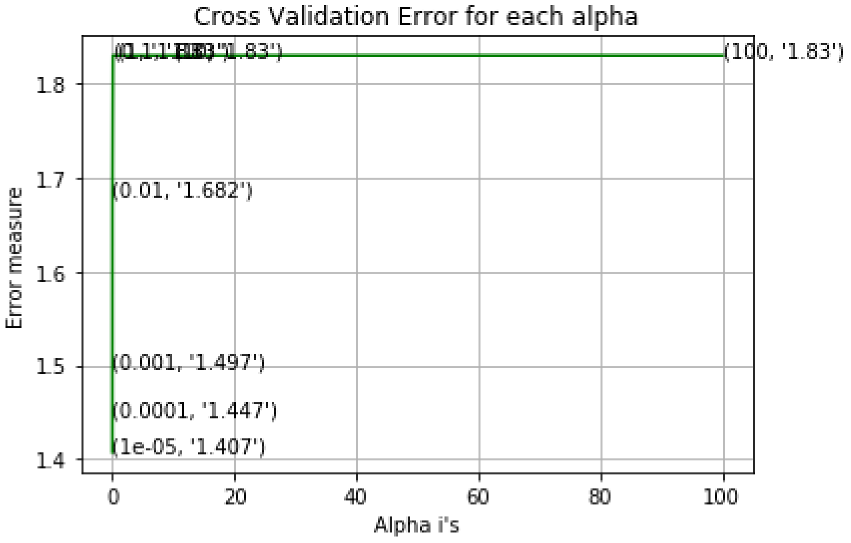
4.4.1. Hyper paramter tuning

In [107]:

```
# 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='l1', loss='hinge', ranc
    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 : 1.4074057498400376
for C = 0.0001
Log Loss : 1.4465129103959362
for C = 0.001
Log Loss : 1.4970416670023292
for C = 0.01
Log Loss : 1.6820308418293461
for C = 0.1
Log Loss : 1.8302564886232473
for C = 1
Log Loss : 1.8302599806267397
for C = 10
Log Loss : 1.8302599806221134
for C = 100
Log Loss : 1.8302599806220166
```



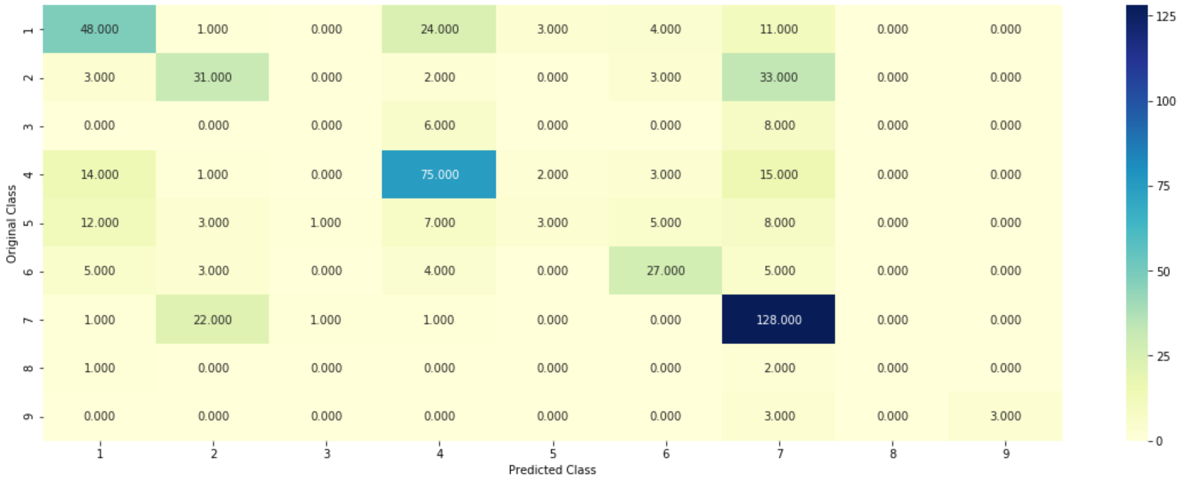
```
For values of best alpha =  1e-05 The train log loss is: 0.8672581114186617
For values of best alpha =  1e-05 The cross validation log loss is: 1.369068
9448726336
For values of best alpha =  1e-05 The test log loss is: 1.3256977646056118
```

4.4.2. Testing model with best hyper parameters

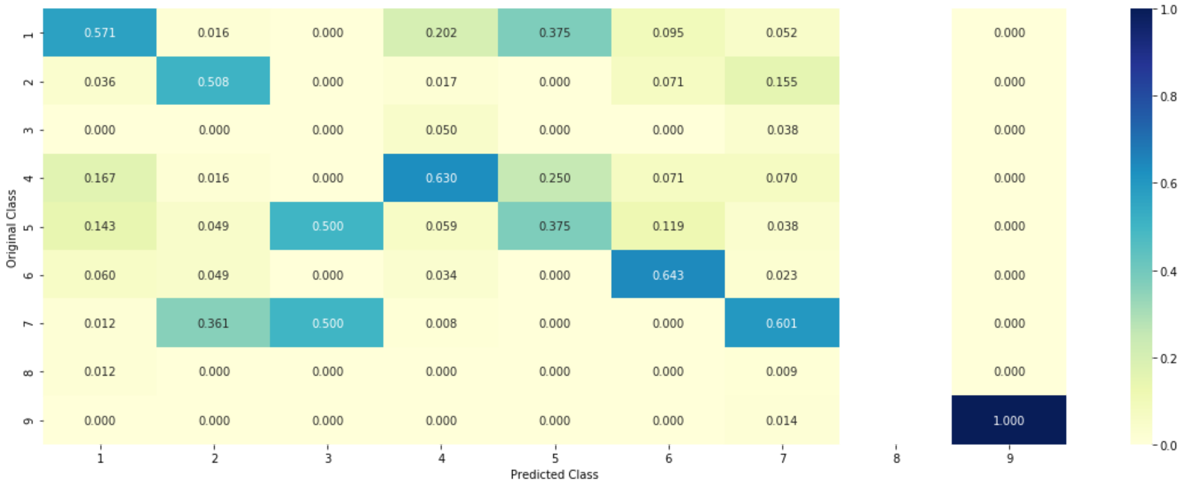
In [108]:

```
# 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='c

# 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
```

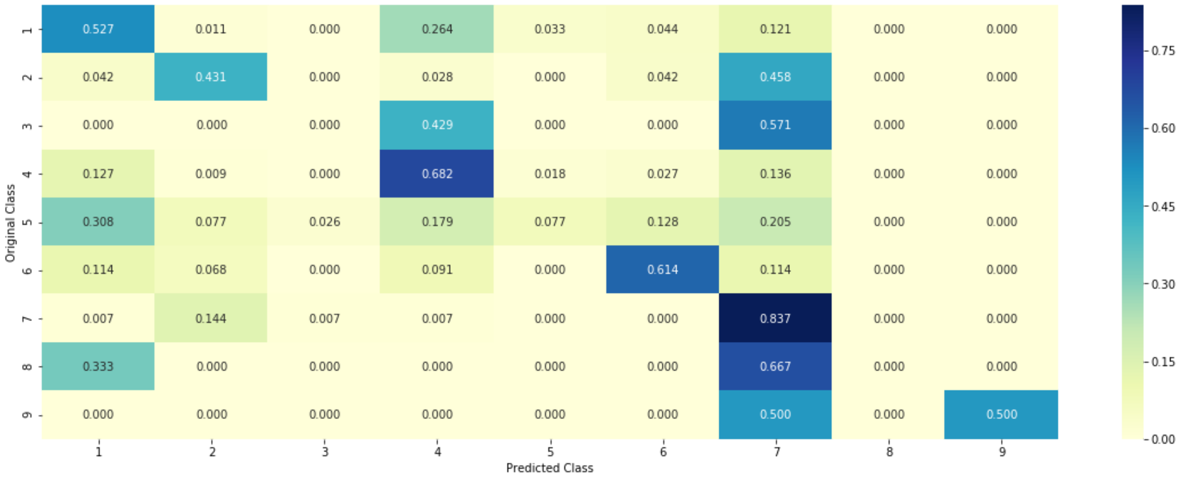
```
Log loss : 1.3690689448726336
Number of mis-classified points : 0.40789473684210525
-------------------- Confusion matrix --------------------
```



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



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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.527	0.011	0.000	0.264	0.033	0.044	0.121	0.000	0.000
2	0.042	0.431	0.000	0.028	0.000	0.042	0.458	0.000	0.000
3	0.000	0.000	0.000	0.429	0.000	0.000	0.571	0.000	0.000
4	0.127	0.009	0.000	0.682	0.018	0.027	0.136	0.000	0.000
5	0.308	0.077	0.026	0.179	0.077	0.128	0.205	0.000	0.000
6	0.114	0.068	0.000	0.091	0.000	0.614	0.114	0.000	0.000
7	0.007	0.144	0.007	0.007	0.000	0.000	0.837	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.667	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.000	0.500

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [110]:

```
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: [[0.0658 0.3539 0.0096 0.0748 0.0221 0.0047
0.4603 0.0043 0.0045]]
Actual Class : 2
--------------------------------------------------
187 Text feature [murine] present in test data point [True]
292 Text feature [constitutive] present in test data point [True]
301 Text feature [activated] present in test data point [True]
374 Text feature [activation] present in test data point [True]
384 Text feature [infiltration] present in test data point [True]
389 Text feature [expressing] present in test data point [True]
476 Text feature [oncogene] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [111]:

```
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 : 6
Predicted Class Probabilities: [[0.0415 0.1232 0.0151 0.1197 0.0238 0.4152
0.2503 0.0045 0.0066]]
Actual Class : 6
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [113]:

```
# --------------------------------
# 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:",
```

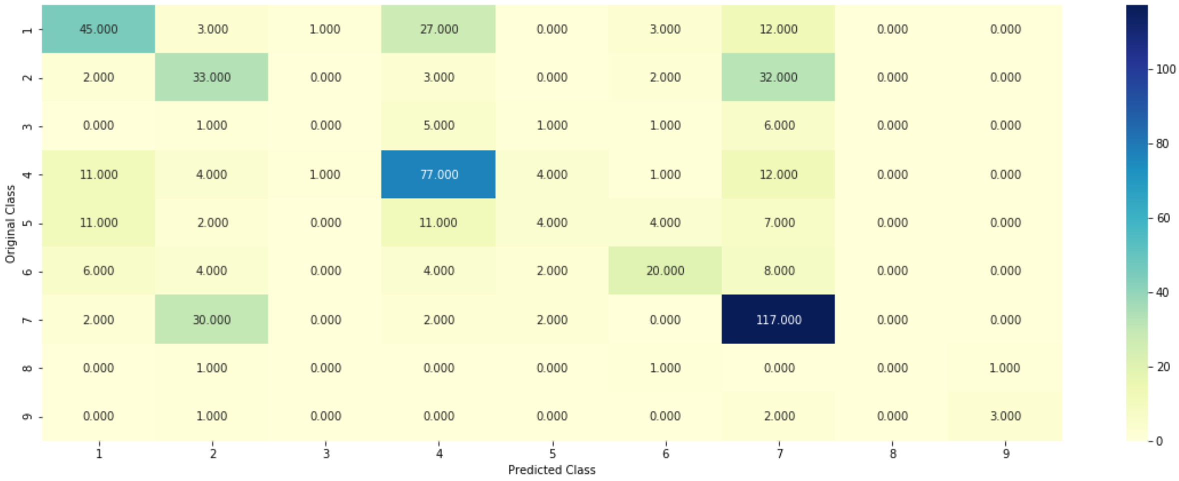
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2936656361878032
for n_estimators = 100 and max depth =  10
Log Loss : 1.256342984516931
for n_estimators = 200 and max depth =  5
Log Loss : 1.2803395283072254
for n_estimators = 200 and max depth =  10
Log Loss : 1.2441600751856337
for n_estimators = 500 and max depth =  5
Log Loss : 1.2814273058332855
for n_estimators = 500 and max depth =  10
Log Loss : 1.2398998578748885
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2779127162313408
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2369613387467098
for n_estimators = 2000 and max depth =  5
Log Loss : 1.273103054651229
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2337464096059254
For values of best estimator =  2000 The train log loss is: 0.64603481782939
73
For values of best estimator =  2000 The cross validation log loss is: 1.233
7464096059254
For values of best estimator =  2000 The test log loss is: 1.118986002895834
3
```

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

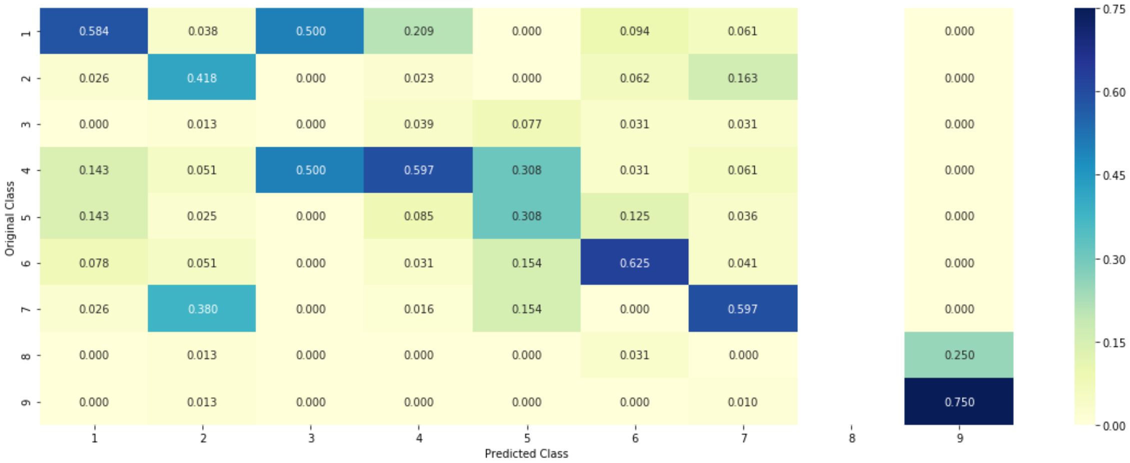
In [114]:

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

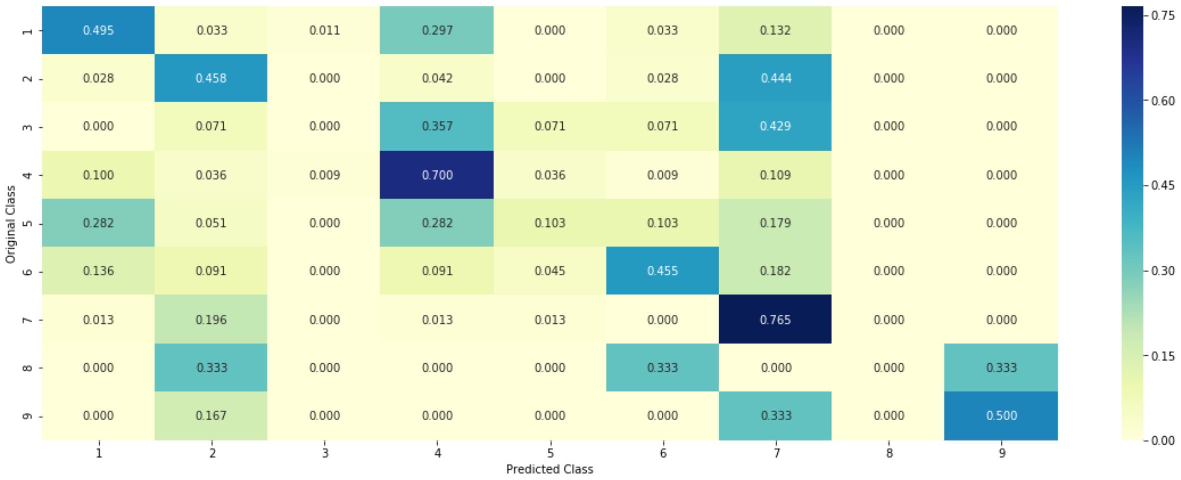
```
Log loss : 1.2337464096059254
Number of mis-classified points : 0.43796992481203006
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [115]:

```
# 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 : 2
Predicted Class Probabilities: [[0.0968 0.3511 0.0171 0.0892 0.0529 0.04
0.3401 0.0055 0.0073]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [phosphorylation] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
7 Text feature [constitutive] present in test data point [True]
8 Text feature [inhibitors] present in test data point [True]
9 Text feature [suppressor] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [signaling] present in test data point [True]
12 Text feature [drug] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
14 Text feature [therapy] present in test data point [True]
16 Text feature [growth] present in test data point [True]
17 Text feature [treated] present in test data point [True]
18 Text feature [function] present in test data point [True]
19 Text feature [loss] present in test data point [True]
20 Text feature [receptor] present in test data point [True]
21 Text feature [therapeutic] present in test data point [True]
23 Text feature [kinases] present in test data point [True]
24 Text feature [missense] present in test data point [True]
25 Text feature [transforming] present in test data point [True]
26 Text feature [nonsense] present in test data point [True]
27 Text feature [inhibition] present in test data point [True]
30 Text feature [cells] present in test data point [True]
32 Text feature [activate] present in test data point [True]
33 Text feature [protein] present in test data point [True]
35 Text feature [trials] present in test data point [True]
37 Text feature [akt] present in test data point [True]
38 Text feature [unstable] present in test data point [True]
39 Text feature [yeast] present in test data point [True]
40 Text feature [stability] present in test data point [True]
41 Text feature [dose] present in test data point [True]
45 Text feature [patients] present in test data point [True]
```

```
46 Text feature [phospho] present in test data point [True]
48 Text feature [pathogenic] present in test data point [True]
50 Text feature [inhibited] present in test data point [True]
51 Text feature [amplification] present in test data point [True]
54 Text feature [variants] present in test data point [True]
56 Text feature [potential] present in test data point [True]
62 Text feature [phosphorylated] present in test data point [True]
63 Text feature [survival] present in test data point [True]
66 Text feature [expressing] present in test data point [True]
69 Text feature [effective] present in test data point [True]
70 Text feature [cell] present in test data point [True]
72 Text feature [functional] present in test data point [True]
73 Text feature [factor] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
76 Text feature [transformation] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
84 Text feature [lines] present in test data point [True]
86 Text feature [proliferation] present in test data point [True]
90 Text feature [harboring] present in test data point [True]
93 Text feature [serum] present in test data point [True]
95 Text feature [proteins] present in test data point [True]
97 Text feature [benefit] present in test data point [True]
99 Text feature [expression] present in test data point [True]
Out of the top  100  features  59 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [116]:

```
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 : 6
Predicted Class Probabilities: [[0.0294 0.1331 0.0136 0.0254 0.0378 0.6282
0.1255 0.0038 0.0032]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [phosphorylation] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
7 Text feature [constitutive] present in test data point [True]
8 Text feature [inhibitors] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [signaling] present in test data point [True]
12 Text feature [drug] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
14 Text feature [therapy] present in test data point [True]
16 Text feature [growth] present in test data point [True]
18 Text feature [function] present in test data point [True]
19 Text feature [loss] present in test data point [True]
20 Text feature [receptor] present in test data point [True]
21 Text feature [therapeutic] present in test data point [True]
22 Text feature [extracellular] present in test data point [True]
23 Text feature [kinases] present in test data point [True]
25 Text feature [transforming] present in test data point [True]
27 Text feature [inhibition] present in test data point [True]
28 Text feature [erk] present in test data point [True]
29 Text feature [f3] present in test data point [True]
30 Text feature [cells] present in test data point [True]
32 Text feature [activate] present in test data point [True]
33 Text feature [protein] present in test data point [True]
34 Text feature [resistance] present in test data point [True]
37 Text feature [akt] present in test data point [True]
40 Text feature [stability] present in test data point [True]
41 Text feature [dose] present in test data point [True]
42 Text feature [3t3] present in test data point [True]
43 Text feature [mitogen] present in test data point [True]
45 Text feature [patients] present in test data point [True]
47 Text feature [egfr] present in test data point [True]
49 Text feature [repair] present in test data point [True]
51 Text feature [amplification] present in test data point [True]
52 Text feature [ligand] present in test data point [True]
53 Text feature [efficacy] present in test data point [True]
54 Text feature [variants] present in test data point [True]
55 Text feature [variant] present in test data point [True]
```

```
56 Text feature [potential] present in test data point [True]
60 Text feature [retained] present in test data point [True]
61 Text feature [ba] present in test data point [True]
62 Text feature [phosphorylated] present in test data point [True]
63 Text feature [survival] present in test data point [True]
66 Text feature [expressing] present in test data point [True]
67 Text feature [autophosphorylation] present in test data point [True]
69 Text feature [effective] present in test data point [True]
70 Text feature [cell] present in test data point [True]
72 Text feature [functional] present in test data point [True]
73 Text feature [factor] present in test data point [True]
74 Text feature [respond] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
76 Text feature [transformation] present in test data point [True]
82 Text feature [sensitivity] present in test data point [True]
83 Text feature [ic50] present in test data point [True]
88 Text feature [imatinib] present in test data point [True]
93 Text feature [serum] present in test data point [True]
95 Text feature [proteins] present in test data point [True]
97 Text feature [benefit] present in test data point [True]
99 Text feature [expression] present in test data point [True]
Out of the top  100  features  63 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [117]:

```
# ---------------------------------
# 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 : 2.0093913157862557
for n_estimators = 10 and max depth =  3
Log Loss : 1.7023490046062248
for n_estimators = 10 and max depth =  5
Log Loss : 1.4307881901282458
for n_estimators = 10 and max depth =  10
Log Loss : 1.8104053982877883
for n_estimators = 50 and max depth =  2
Log Loss : 1.6877657752373665
for n_estimators = 50 and max depth =  3
Log Loss : 1.5572405556547657
for n_estimators = 50 and max depth =  5
Log Loss : 1.4440048011987656
for n_estimators = 50 and max depth =  10
Log Loss : 1.7310167496177535
for n_estimators = 100 and max depth =  2
Log Loss : 1.5601652145325042
for n_estimators = 100 and max depth =  3
Log Loss : 1.5571217518302463
for n_estimators = 100 and max depth =  5
Log Loss : 1.381410987133976
for n_estimators = 100 and max depth =  10
Log Loss : 1.7271338932376588
for n_estimators = 200 and max depth =  2
Log Loss : 1.6153737668801353
for n_estimators = 200 and max depth =  3
Log Loss : 1.5447426205038397
for n_estimators = 200 and max depth =  5
Log Loss : 1.4909026464035138
for n_estimators = 200 and max depth =  10
Log Loss : 1.7794493635182582
for n_estimators = 500 and max depth =  2
Log Loss : 1.7039135226315825
for n_estimators = 500 and max depth =  3
Log Loss : 1.6149416576958164
for n_estimators = 500 and max depth =  5
Log Loss : 1.4942665174496965
for n_estimators = 500 and max depth =  10
Log Loss : 1.7962351286716658
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6967892387882357
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6186597850561273
for n_estimators = 1000 and max depth =  5
```

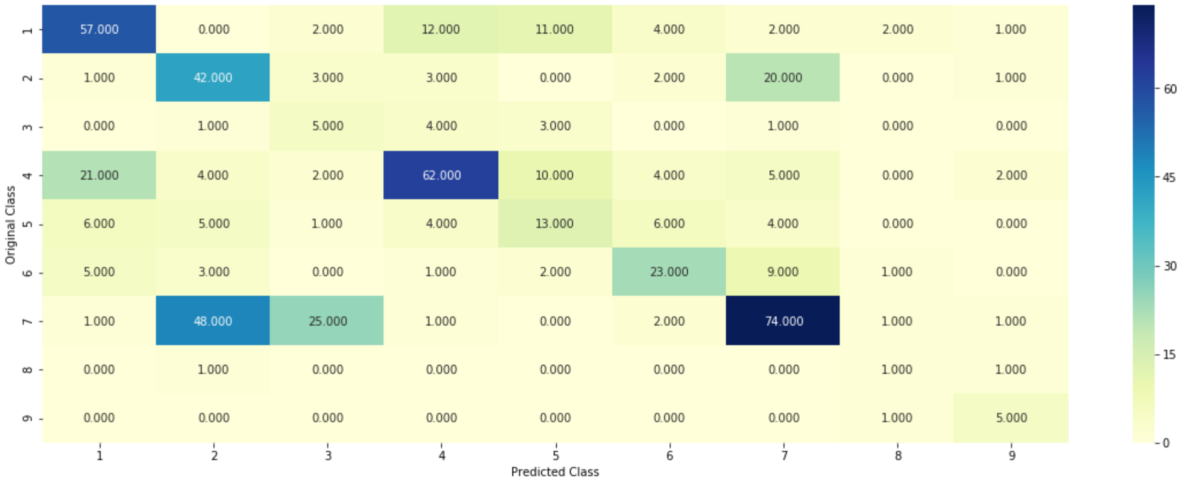
```
Log Loss : 1.497104269575992
for n_estimators = 1000 and max depth =  10
Log Loss : 1.804039449710611
For values of best alpha =  100 The train log loss is: 0.05487496401265011
For values of best alpha =  100 The cross validation log loss is: 1.38141098
7133976
For values of best alpha =  100 The test log loss is: 1.3242106845941963
```

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

In [118]:

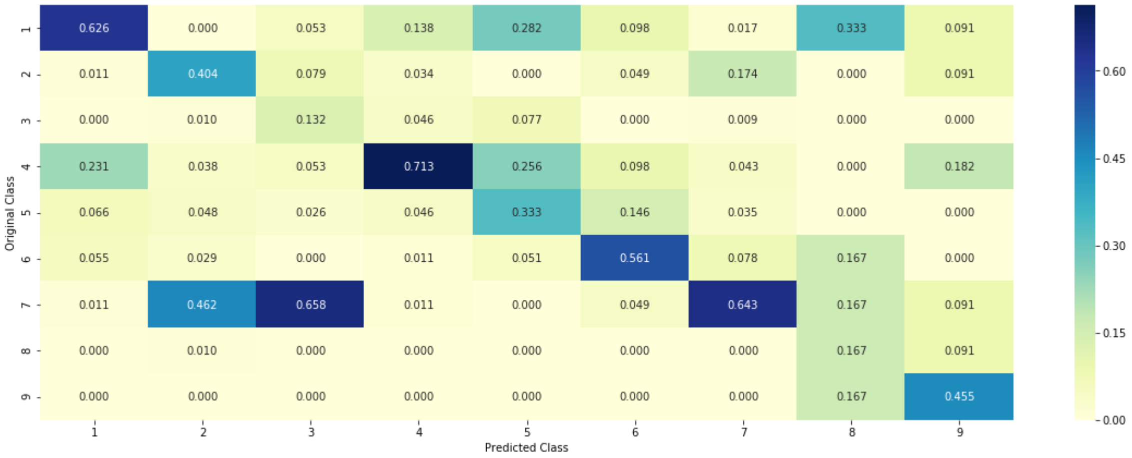
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=Non
# min_impurity_split=None, bootstrap=True, oob_score=False, 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 : 1.381410987133976
Number of mis-classified points : 0.4699248120300752
-------------------- Confusion matrix --------------------
```



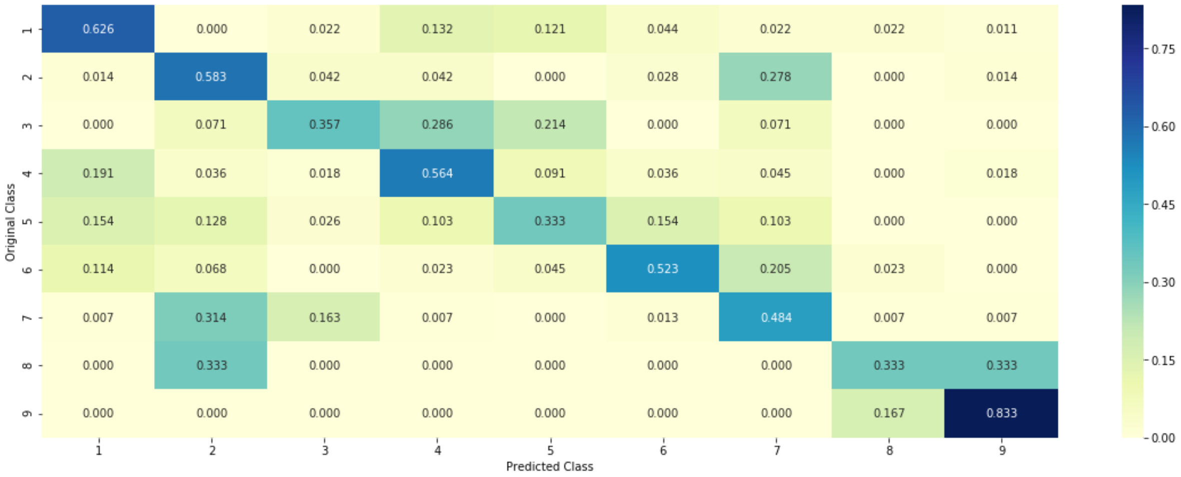
	1	2	3	4	5	6	7	8	9
1	57.000	0.000	2.000	12.000	11.000	4.000	2.000	2.000	1.000
2	1.000	42.000	3.000	3.000	0.000	2.000	20.000	0.000	1.000
3	0.000	1.000	5.000	4.000	3.000	0.000	1.000	0.000	0.000
4	21.000	4.000	2.000	62.000	10.000	4.000	5.000	0.000	2.000
5	6.000	5.000	1.000	4.000	13.000	6.000	4.000	0.000	0.000
6	5.000	3.000	0.000	1.000	2.000	23.000	9.000	1.000	0.000
7	1.000	48.000	25.000	1.000	0.000	2.000	74.000	1.000	1.000
8	0.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.626	0.000	0.053	0.138	0.282	0.098	0.017	0.333	0.091
2	0.011	0.404	0.079	0.034	0.000	0.049	0.174	0.000	0.091
3	0.000	0.010	0.132	0.046	0.077	0.000	0.009	0.000	0.000
4	0.231	0.038	0.053	0.713	0.256	0.098	0.043	0.000	0.182
5	0.066	0.048	0.026	0.046	0.333	0.146	0.035	0.000	0.000
6	0.055	0.029	0.000	0.011	0.051	0.561	0.078	0.167	0.000
7	0.011	0.462	0.658	0.011	0.000	0.049	0.643	0.167	0.091
8	0.000	0.010	0.000	0.000	0.000	0.000	0.000	0.167	0.091
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.167	0.455

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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [119]:

```
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 : 2
Predicted Class Probabilities: [[0.0183 0.5825 0.0736 0.0247 0.0232 0.0428
0.1573 0.0488 0.0288]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [120]:

```
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 : 6
Predicted Class Probabilities: [[0.0124 0.2855 0.0826 0.0192 0.0193 0.3647
0.1833 0.0205 0.0125]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [121]:

```
# 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
```

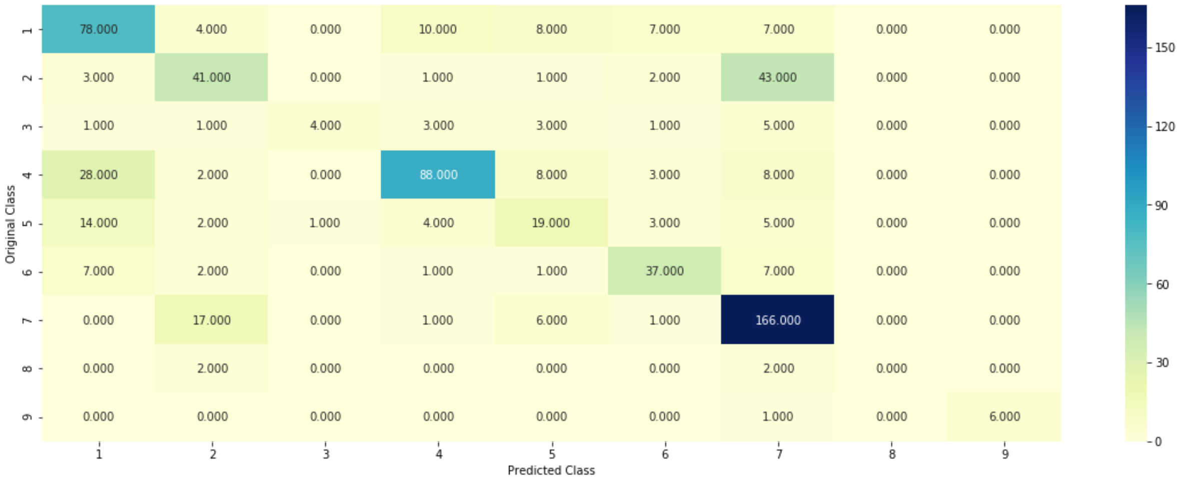
```
Logistic Regression :  Log Loss: 1.21
Support vector machines : Log Loss: 1.58
Naive Bayes : Log Loss: 1.32
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.034
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.521
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.237
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.434
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.770
```

4.7.2 testing the model with the best hyper parameters

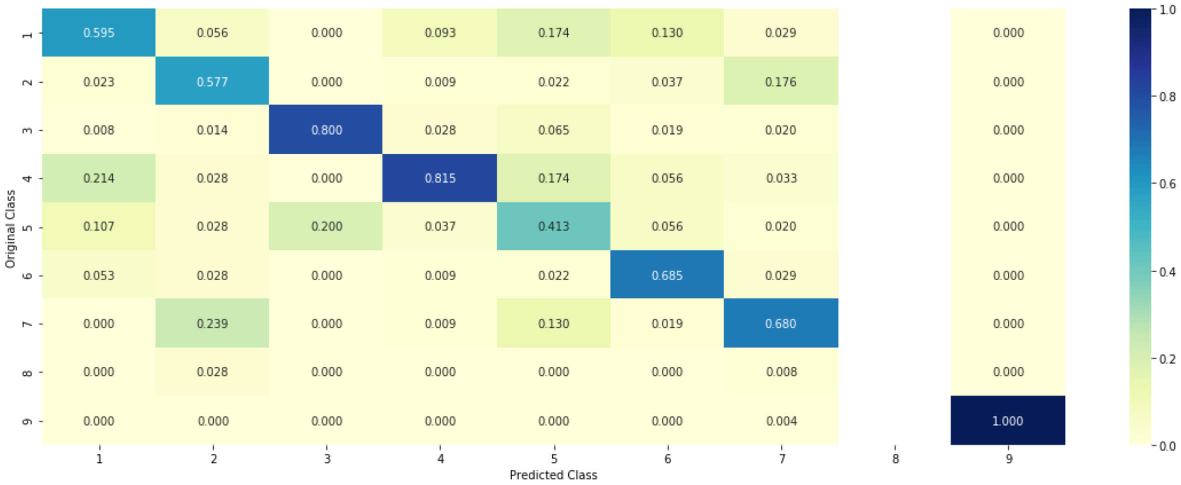
In [122]:

```
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))
```

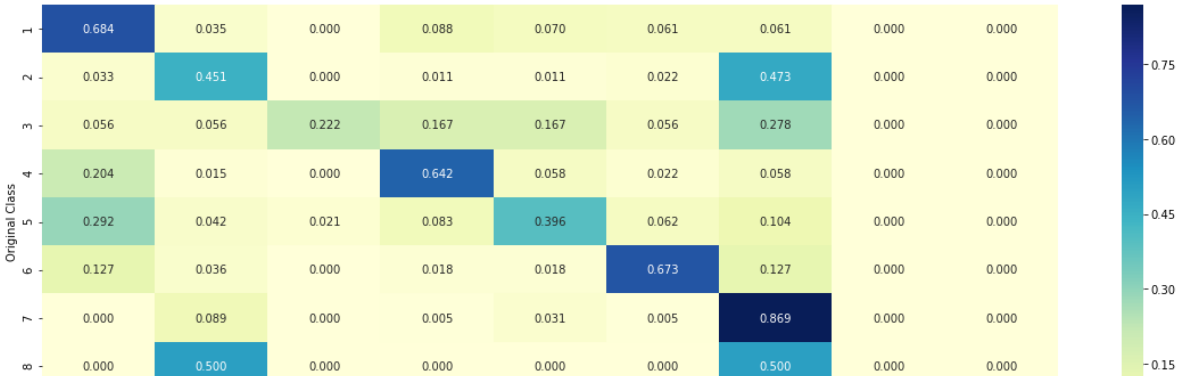
```
Log loss (train) on the stacking classifier : 0.6034318602061024
Log loss (CV) on the stacking classifier : 1.236818346135763
Log loss (test) on the stacking classifier : 1.1061608734525015
Number of missclassified point : 0.3398496240601504
-------------------- Confusion matrix --------------------
```



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



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

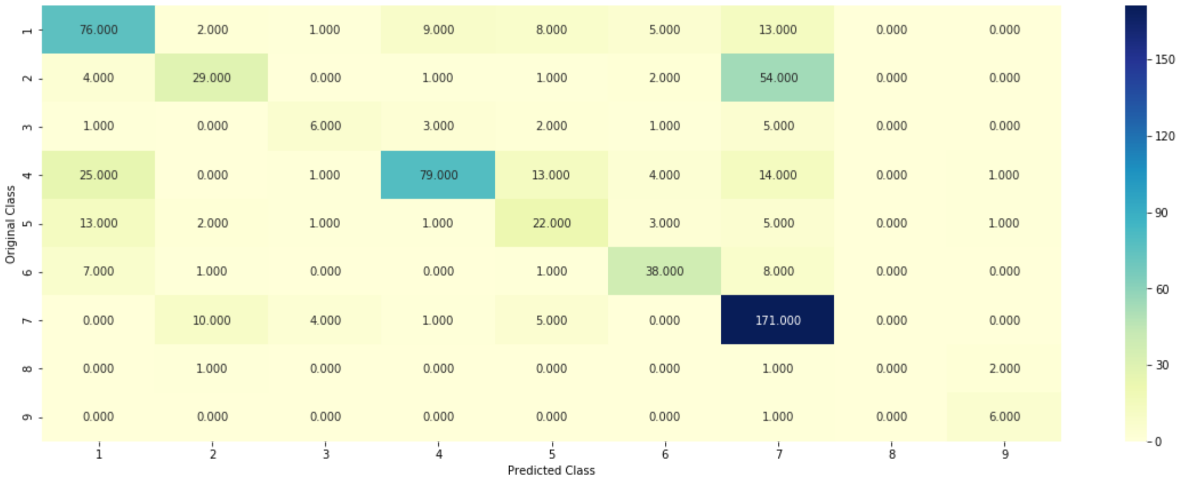


4.7.3 Maximum Voting classifier

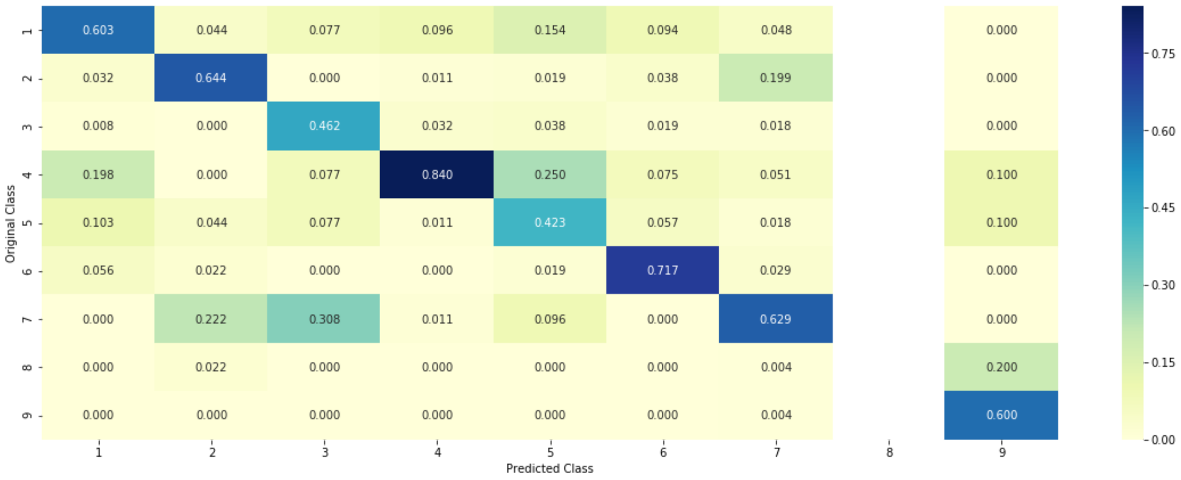
In [123]:

```
#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_count_vectorizer, 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_count_vectc
```

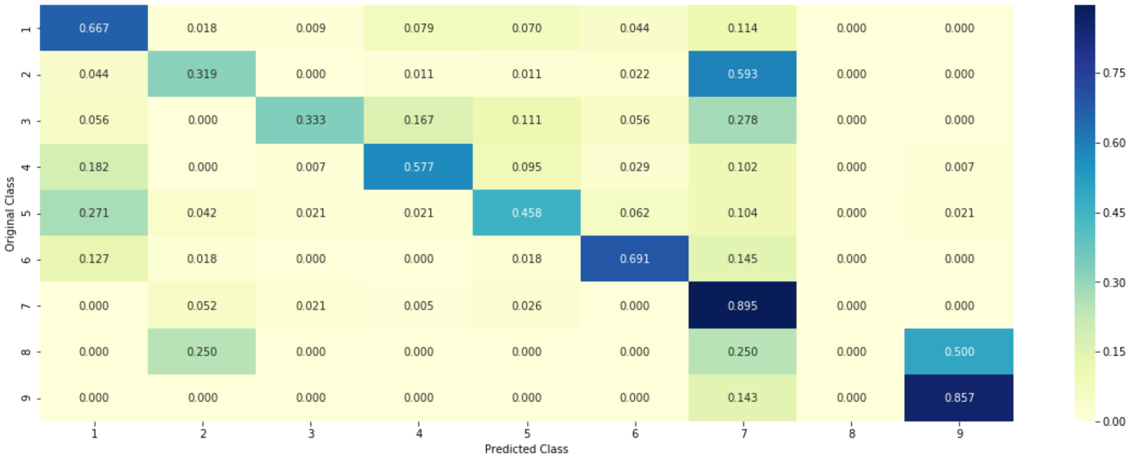
```
Log loss (train) on the VotingClassifier : 0.8706539098795264
Log loss (CV) on the VotingClassifier : 1.2312398746353785
Log loss (test) on the VotingClassifier : 1.1671172180862601
Number of missclassified point : 0.35789473684210527
-------------------- Confusion matrix --------------------
```



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



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



Logistic Regression with Unigram and Bigram

In [125]:

```
#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 [126]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
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'])

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

In [127]:

```
# 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 [128]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
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'])


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

In [129]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1, 2))
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 : 776298
```

In [130]:

```
#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)

# 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 [131]:

```
# 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 [134]:

```
#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_variatic
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_respc
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 [135]:

```
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_onehc
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 778596)
(number of data points * number of features) in test data =  (665, 778596)
(number of data points * number of features) in cross validation data = (53
2, 778596)
```

In [136]:

```
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
```

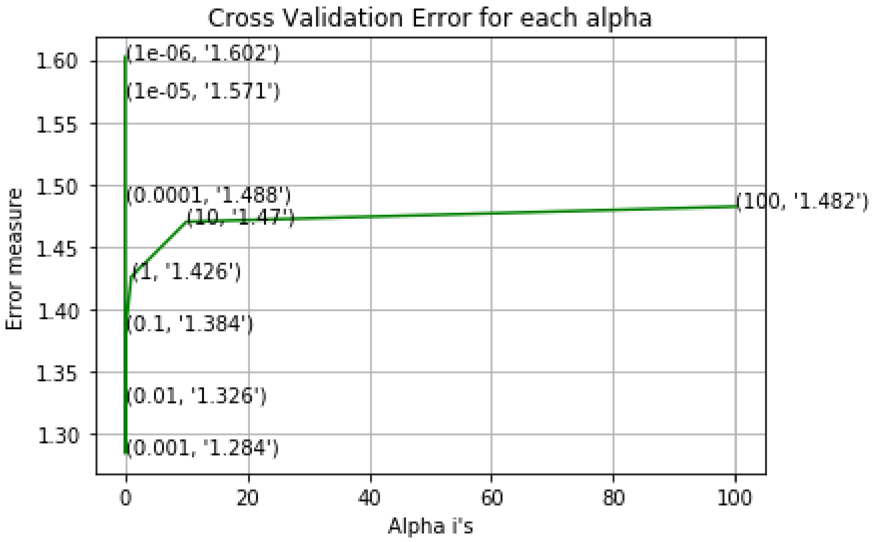
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (53
2, 27)
```

In [137]:

```
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_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.6021992726588188
for alpha = 1e-05
Log Loss : 1.5710843559734535
for alpha = 0.0001
Log Loss : 1.487799347753972
for alpha = 0.001
Log Loss : 1.2841812205038972
for alpha = 0.01
Log Loss : 1.3263254371583522
for alpha = 0.1
```

```
Log Loss : 1.3839679325177634
for alpha = 1
Log Loss : 1.4257968131693906
for alpha = 10
Log Loss : 1.4698460176409665
for alpha = 100
Log Loss : 1.4820247772325243
```

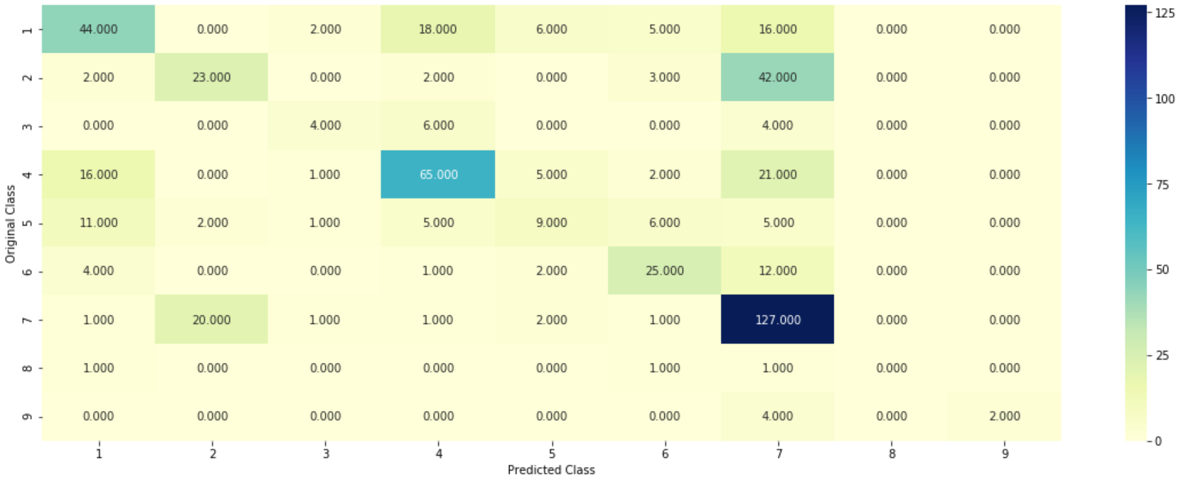


```
For values of best alpha =  0.001 The train log loss is: 0.7803109920702945
For values of best alpha =  0.001 The cross validation log loss is: 1.284181
2205038972
For values of best alpha =  0.001 The test log loss is: 1.206003640602135
```

In [138]:

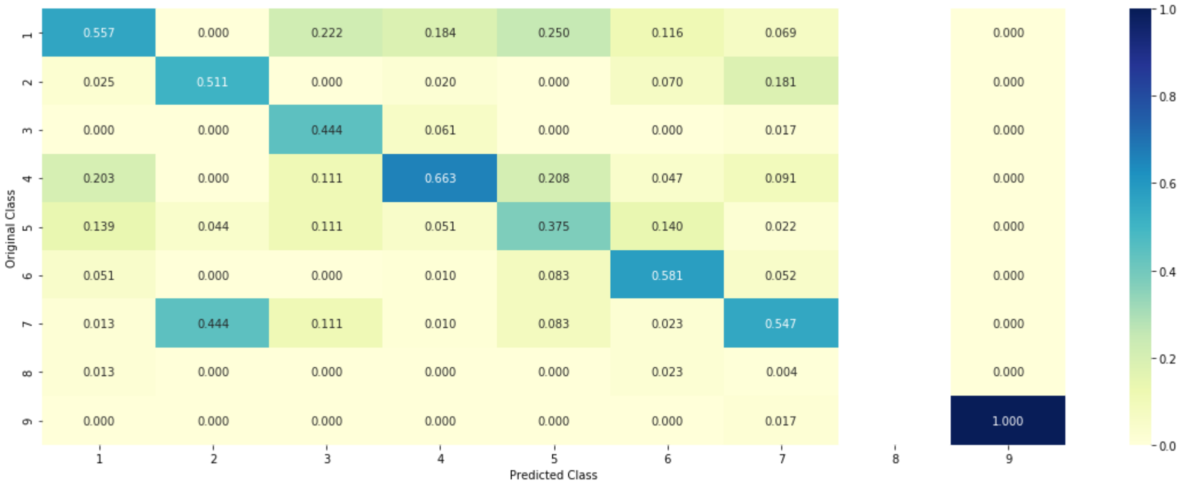
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

```
Log loss : 1.2841812205038972
Number of mis-classified points : 0.43796992481203006
-------------------- Confusion matrix --------------------
```



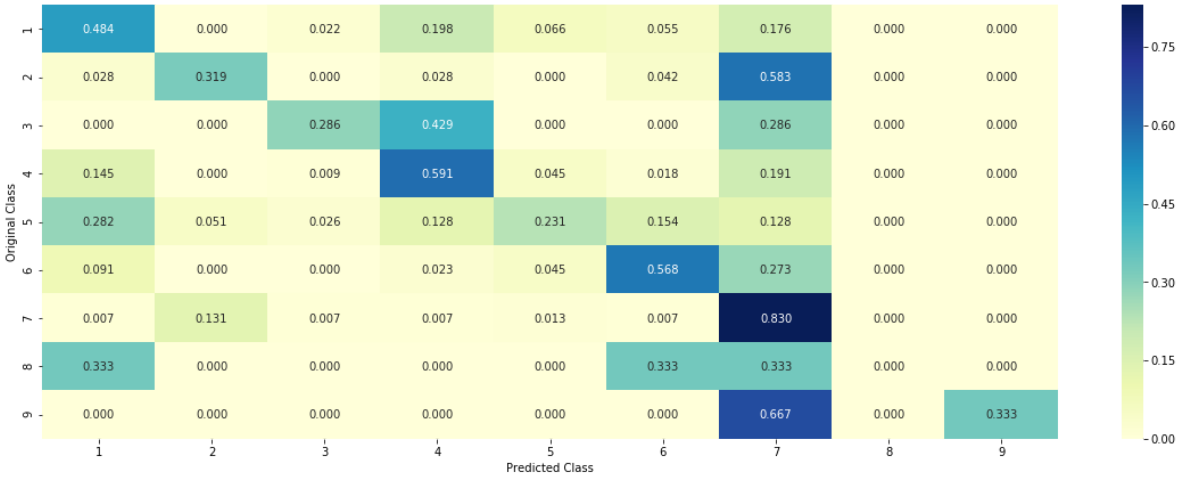
	1	2	3	4	5	6	7	8	9
1	44.000	0.000	2.000	18.000	6.000	5.000	16.000	0.000	0.000
2	2.000	23.000	0.000	2.000	0.000	3.000	42.000	0.000	0.000
3	0.000	0.000	4.000	6.000	0.000	0.000	4.000	0.000	0.000
4	16.000	0.000	1.000	65.000	5.000	2.000	21.000	0.000	0.000
5	11.000	2.000	1.000	5.000	9.000	6.000	5.000	0.000	0.000
6	4.000	0.000	0.000	1.000	2.000	25.000	12.000	0.000	0.000
7	1.000	20.000	1.000	1.000	2.000	1.000	127.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.557	0.000	0.222	0.184	0.250	0.116	0.069		0.000
2	0.025	0.511	0.000	0.020	0.000	0.070	0.181		0.000
3	0.000	0.000	0.444	0.061	0.000	0.000	0.017		0.000
4	0.203	0.000	0.111	0.663	0.208	0.047	0.091		0.000
5	0.139	0.044	0.111	0.051	0.375	0.140	0.022		0.000
6	0.051	0.000	0.000	0.010	0.083	0.581	0.052		0.000
7	0.013	0.444	0.111	0.010	0.083	0.023	0.547		0.000
8	0.013	0.000	0.000	0.000	0.000	0.023	0.004		0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.017		1.000

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



	1	2	3	4	5	6	7	8	9
1	0.484	0.000	0.022	0.198	0.066	0.055	0.176	0.000	0.000
2	0.028	0.319	0.000	0.028	0.000	0.042	0.583	0.000	0.000
3	0.000	0.000	0.286	0.429	0.000	0.000	0.286	0.000	0.000
4	0.145	0.000	0.009	0.591	0.045	0.018	0.191	0.000	0.000
5	0.282	0.051	0.026	0.128	0.231	0.154	0.128	0.000	0.000
6	0.091	0.000	0.000	0.023	0.045	0.568	0.273	0.000	0.000
7	0.007	0.131	0.007	0.007	0.013	0.007	0.830	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.333	0.333	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.667	0.000	0.333

Logistic Regression with Feature Engineering

In [140]:

```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_s

x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train, test_si
```

In [141]:

```
# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    value_count = x_train[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    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])
    return gv_fea
```

In [142]:

```
#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", x_train))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_test))

# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_cv))
```

In [143]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

In [144]:

```
# alpha is used for laplace smoothing
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_trai

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_test)

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_cv))
```

In [145]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['Variatio
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'])
```

In [146]:

```
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


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 [147]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['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 : 127583
```

In [148]:

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['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(x_train)


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 [149]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# 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 [150]:

```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['TEXT'])
```

In [152]:

```
# Collecting all the genes and variations data into a single list
gene_variation = []

for gene in data['Gene'].values:
    gene_variation.append(gene)

for variation in data['Variation'].values:
    gene_variation.append(variation)
```

In [153]:

```
tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
cv_text = tfidfVectorizer.transform(x_cv['TEXT'])
```

In [154]:

```
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

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_onehotCoding)).tocs
train_y = np.array(list(x_train['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(x_test['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(x_cv['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variatic
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 [155]:

```
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_onehc
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 130779)
(number of data points * number of features) in test data =  (665, 130779)
(number of data points * number of features) in cross validation data = (53
2, 130779)
```

In [156]:

```
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_respc
```

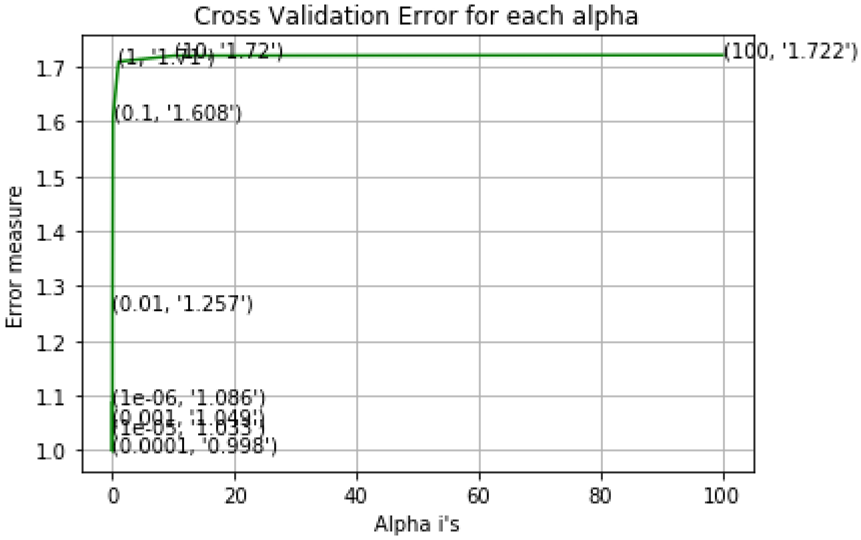
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (53
2, 27)
```

In [157]:

```
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_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.0863245617885202
for alpha = 1e-05
Log Loss : 1.0330639512332485
for alpha = 0.0001
Log Loss : 0.9976164844838811
for alpha = 0.001
Log Loss : 1.0493255403970059
for alpha = 0.01
Log Loss : 1.2568820422261708
for alpha = 0.1
```

```
Log Loss : 1.6076436888420493
for alpha = 1
Log Loss : 1.7095924806207385
for alpha = 10
Log Loss : 1.720482183733094
for alpha = 100
Log Loss : 1.7216245779867398
```

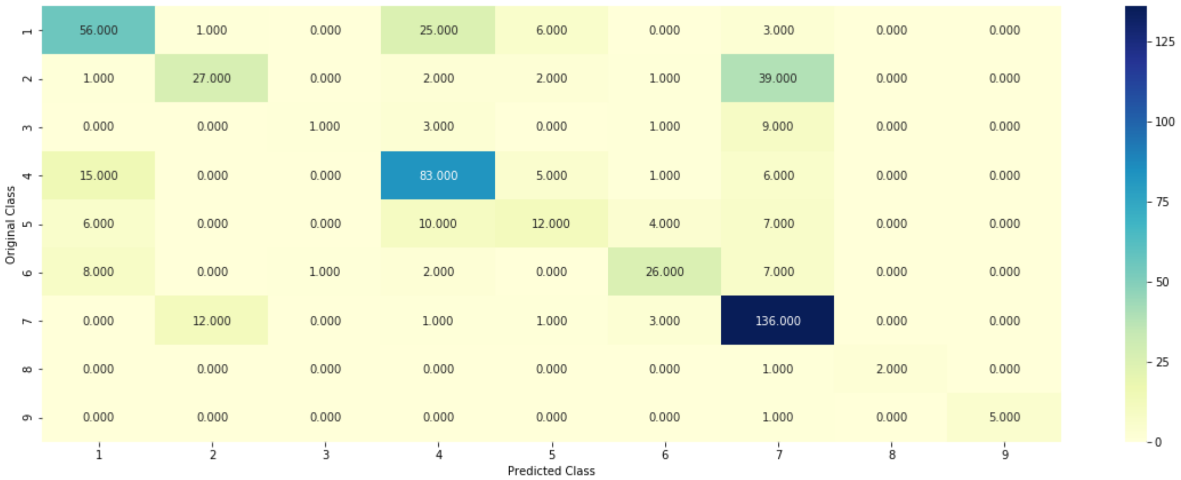


```
For values of best alpha =  0.0001 The train log loss is: 0.4318903316494556
For values of best alpha =  0.0001 The cross validation log loss is: 0.99761
64844838811
For values of best alpha =  0.0001 The test log loss is: 0.9515513597682231
```

In [158]:

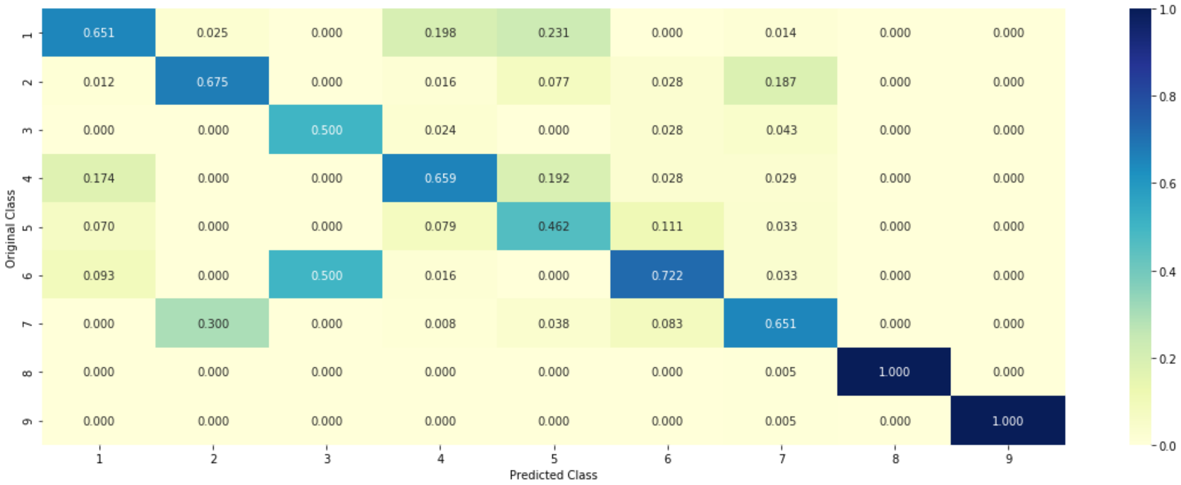
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

```
Log loss : 0.9976164844838811
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```



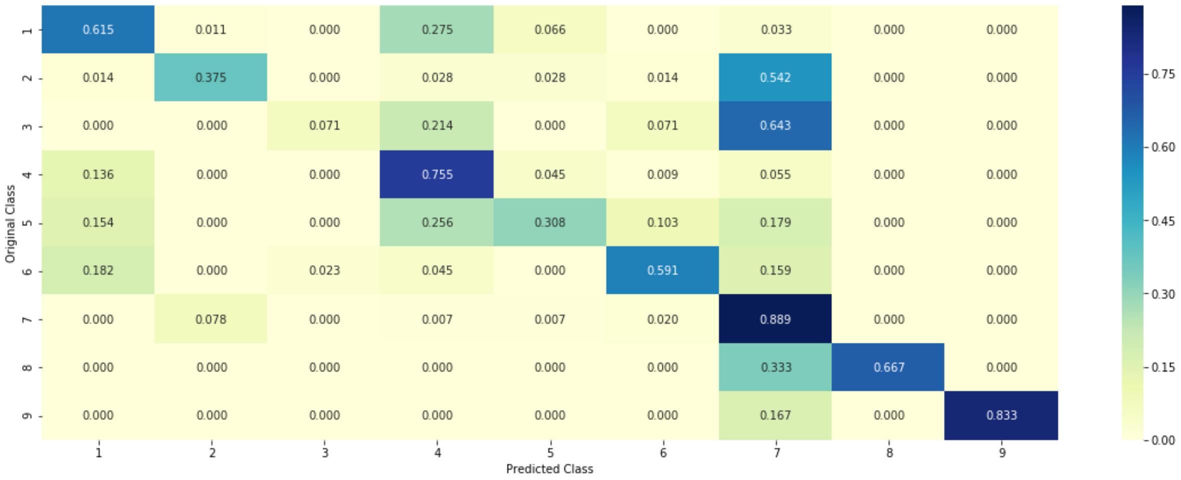
	1	2	3	4	5	6	7	8	9
1	56.000	1.000	0.000	25.000	6.000	0.000	3.000	0.000	0.000
2	1.000	27.000	0.000	2.000	2.000	1.000	39.000	0.000	0.000
3	0.000	0.000	1.000	3.000	0.000	1.000	9.000	0.000	0.000
4	15.000	0.000	0.000	83.000	5.000	1.000	6.000	0.000	0.000
5	6.000	0.000	0.000	10.000	12.000	4.000	7.000	0.000	0.000
6	8.000	0.000	1.000	2.000	0.000	26.000	7.000	0.000	0.000
7	0.000	12.000	0.000	1.000	1.000	3.000	136.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.651	0.025	0.000	0.198	0.231	0.000	0.014	0.000	0.000
2	0.012	0.675	0.000	0.016	0.077	0.028	0.187	0.000	0.000
3	0.000	0.000	0.500	0.024	0.000	0.028	0.043	0.000	0.000
4	0.174	0.000	0.000	0.659	0.192	0.028	0.029	0.000	0.000
5	0.070	0.000	0.000	0.079	0.462	0.111	0.033	0.000	0.000
6	0.093	0.000	0.500	0.016	0.000	0.722	0.033	0.000	0.000
7	0.000	0.300	0.000	0.008	0.038	0.083	0.651	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.005	1.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.615	0.011	0.000	0.275	0.066	0.000	0.033	0.000	0.000
2	0.014	0.375	0.000	0.028	0.028	0.014	0.542	0.000	0.000
3	0.000	0.000	0.071	0.214	0.000	0.071	0.643	0.000	0.000
4	0.136	0.000	0.000	0.755	0.045	0.009	0.055	0.000	0.000
5	0.154	0.000	0.000	0.256	0.308	0.103	0.179	0.000	0.000
6	0.182	0.000	0.023	0.045	0.000	0.591	0.159	0.000	0.000
7	0.000	0.078	0.000	0.007	0.007	0.020	0.889	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.167	0.000	0.833

In [159]:

```
print()
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = "*** Model Summary *** [Performance Metric: Log-Loss]"
ptable.field_names=["Model Name","Train","CV","Test","% Misclassified Points"]
ptable.add_row(["Naive Bayes","0.91","1.29","1.20","45"])
ptable.add_row(["KNN","0.63","1.15","1.05","41"])
ptable.add_row(["Logistic Regression With Class balancing","0.55","1.19","1.09","39"])
ptable.add_row(["Logistic Regression Without Class balancing","0.51","1.21","1.10","39"])
ptable.add_row(["Linear SVM","0.86","1.36","1.32","41"])
ptable.add_row(["Random Forest Classifier With One hot Encoding","0.64","1.23","1.12","43"]
ptable.add_row(["Random Forest Classifier With Response Coding","0.05","1.38","1.32","47"])
ptable.add_row(["Stack Models:LR+NB+SVM","0.","1.15","1.06","32"])
ptable.add_row(["Maximum Voting classifier","0.63","1.23","1.06","34"])
ptable.add_row(["Logistic Regression with Unigram and Bigram ","0.78","1.28","1.20","43"])
ptable.add_row(["Logistic Regression with Feature Engineering ","0.43","0.99","0.95","35"])
print(ptable)
print()
```

```
+------------------------------------------------+-------+------+------+----
--------------------+
|                   Model Name                   | Train |  CV  | Test | % M
isclassified Points |
+------------------------------------------------+-------+------+------+----
--------------------+
|                  Naive Bayes                   |  0.91 | 1.29 | 1.20 |
45           |
|                      KNN                       |  0.63 | 1.15 | 1.05 |
41           |
|    Logistic Regression With Class balancing    |  0.55 | 1.19 | 1.09 |
39           |
|  Logistic Regression Without Class balancing   |  0.51 | 1.21 | 1.10 |
39           |
|                   Linear SVM                   |  0.86 | 1.36 | 1.32 |
41           |
| Random Forest Classifier With One hot Encoding |  0.64 | 1.23 | 1.12 |
43           |
| Random Forest Classifier With Response Coding  |  0.05 | 1.38 | 1.32 |
47           |
|             Stack Models:LR+NB+SVM             |   0.  | 1.15 | 1.06 |
32           |
|           Maximum Voting classifier            |  0.63 | 1.23 | 1.06 |
34           |
|  Logistic Regression with Unigram and Bigram   |  0.78 | 1.28 | 1.20 |
43           |
| Logistic Regression with Feature Engineering   |  0.43 | 0.99 | 0.95 |
35           |
+------------------------------------------------+-------+------+------+----
--------------------+
```

Steps Followed:

1. The data is loaded in two different files 'training_variants' where it contains features like ID, Gene,

Variations and Class and 'training_text' where it contains ID and Text data

2. We Preprocess the Text using nltk to remove the stopwords
3. We then merge the two files on ID which is the common field in both training_variants and training_text
4. We fill in the rows where the Text is NULL with with the Gene and Variation value of the same row
5. We then split the entire data into Training data(64%), CV data(20%) and Test Data(16%)
6. We then plot bar graph to check the distribution of Yi's in the Train, CV and Test Data
7. We start with doing prediction with a Random Model and get a CV log loss of 2.5 and Test Log loss of 2.44
8. We then do Univariate Analysis of Genes, Variations and Text Features.
9. We also use Logistic Regression to find out how each of the features behave and how good they are in Prediction of the Yi's (Class label)
10. We do both Response Encoding and One Hot Encoding of the above features and stack them to accordingly to apply to different models which we will use to find the best model which will provide the lease log loss
11. We use Naive Bayes as our base line model which provides train log loss of 0.91, test log loss of 1.20 and CV log loss of 1.29
12. We try Various models such as:

```
a. Naive Bayes
b. KNN
c. Logistic Regression With Class balancing
d. Logistic Regression Without Class balancing
e. Linear SVM
f. Random Forest Classifier With One hot Encoding
g. Random Forest Classifier With Response Coding
h. Stack Models:LR+NB+SVM
i. Maximum Voting classifier
```

13. All the above models are tried with hyperparameter tuning and the best hyperparameter is used to predict the log loss of the Test and CV data
14. By Trying all the above models we were not able to reduce the log loss to less than 1 which is one of the objective od the case study
15. Hence, We move ahead with Logistic Regression with Unigram and Bigram Featurization, but still the Log Loss is at 1.28 and 1.20 for CV and Test Data respectively
16. We tried some feature engineering by merging gene and variation data into one list and apply TFidfVectorizer on top of it.
17. We apply TFidfVectorizer for the text data and combine them using Hstack with the gene and variation
18. Then we apply Logistic Regression with Hyper parameter tuning and was able to reduce Test Log loss to 0.95 and CV log loss to 0.99

In [ ]: