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 [2]:

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

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

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 81.16369210594794 seconds
```

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

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

In [8]:

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

In [9]:

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

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

In [11]:

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

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

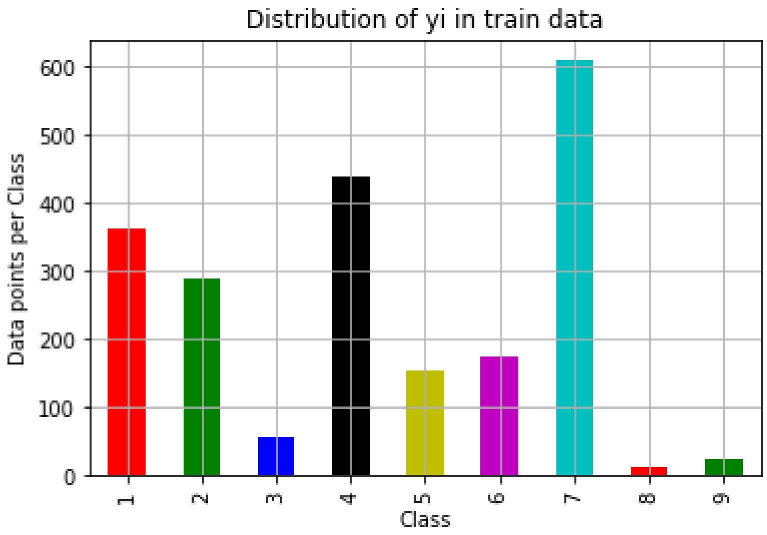
In [12]:

```
#train_df['Class'].value_counts().sort_index()
```

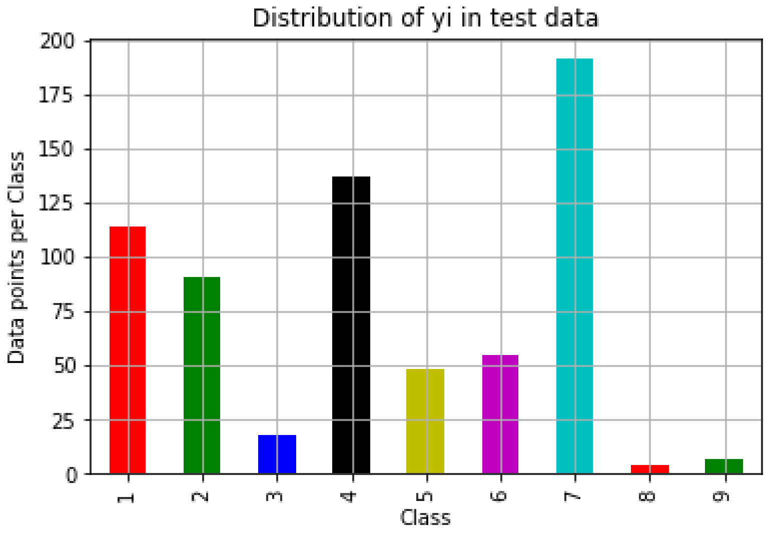
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:

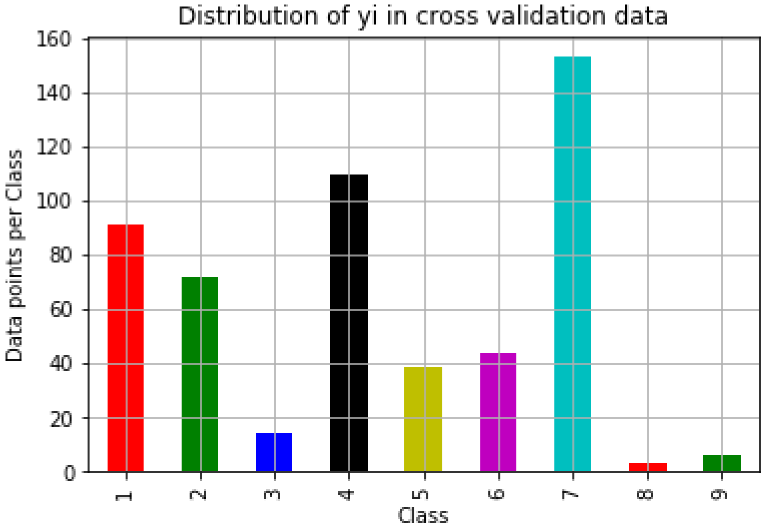
```
# it returns a dict, keys as class labels and values as the number of data points in that c
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar',color=my_colors)
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 = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar',color=my_colors)
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 = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar',color=my_colors)
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(-cv_class_distribution.values)
for i in sorted_yi:
    #print("i= ",i)
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(',
```



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



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



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

3.2 Prediction using a 'Random' Model

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

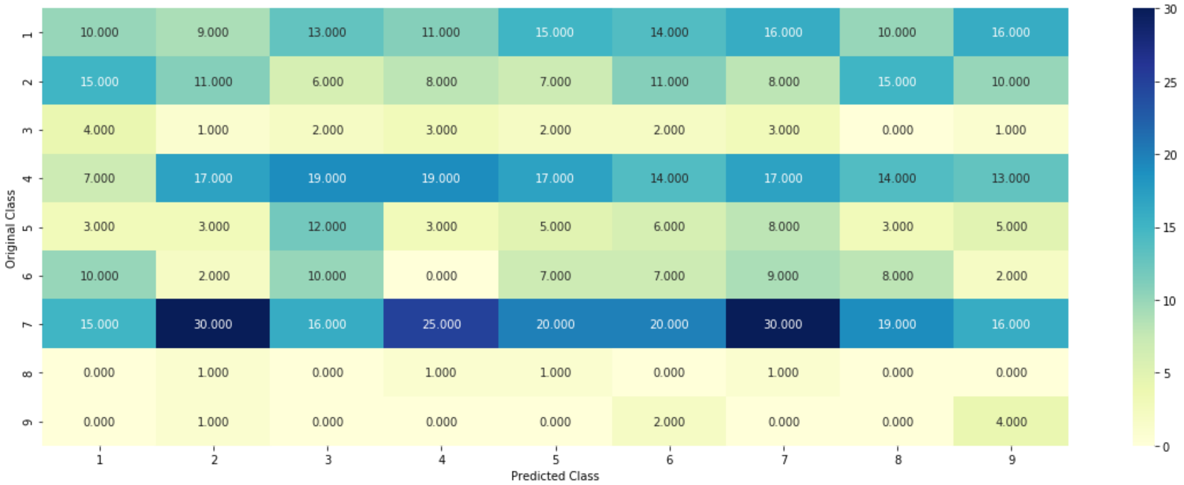
In [13]:

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

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

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

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

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

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

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

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

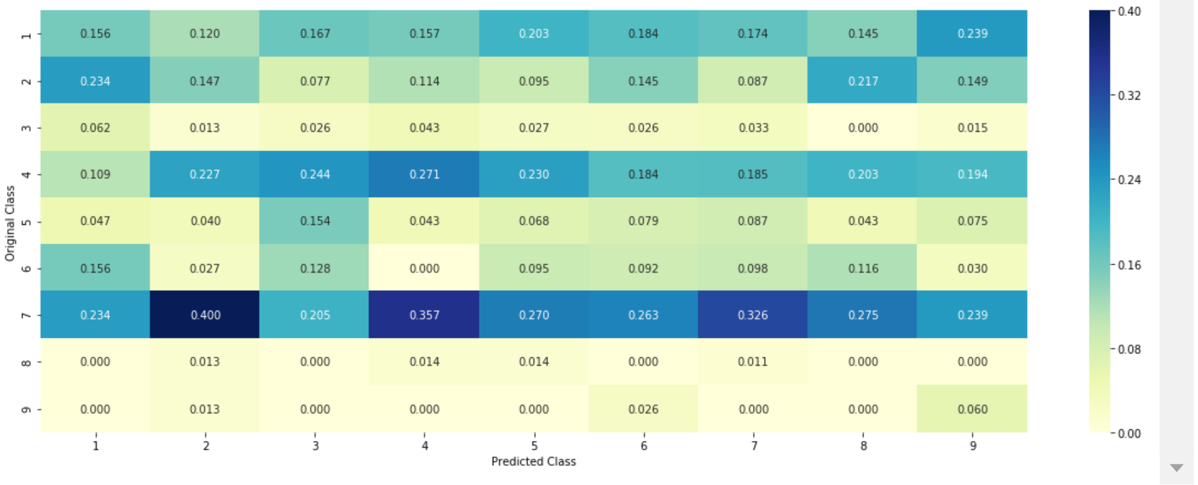
In [14]:

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

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
#print(cv_predicted_y)
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)
```

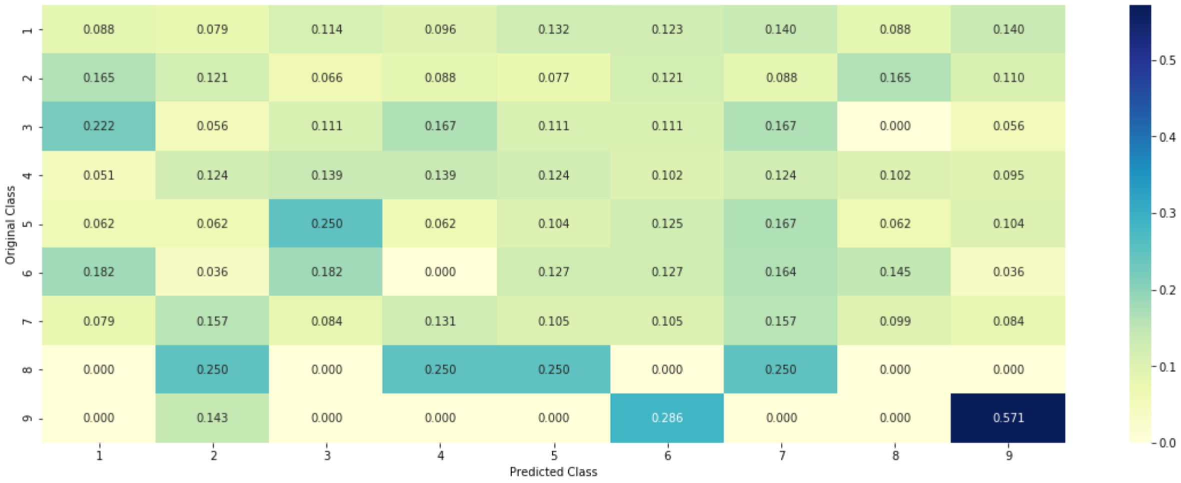
```
Log loss on Cross Validation Data using Random Model 2.5071989616476253
Log loss on Test Data using Random Model 2.5085821480343857
-------------------- Confusion matrix --------------------
```



```
-------------------- 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 : 233
BRCA1     167
TP53      111
EGFR       94
BRCA2      81
PTEN       77
BRAF       66
KIT        64
ALK        48
ERBB2      42
PIK3CA     36
Name: Gene, dtype: int64
```

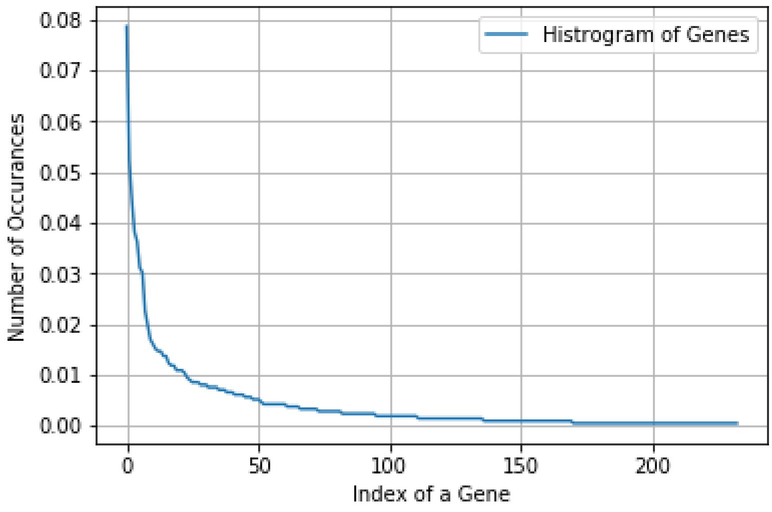
In [18]:

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

```
Ans: There are 233 different categories of genes in the train data, and they
are distibuted as follows
```

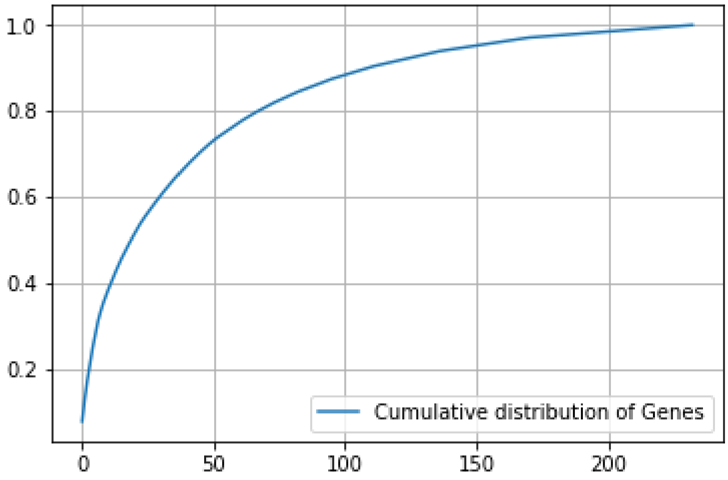
In [19]:

```
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/)](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]:

```
onseCoding is converted feature using respone coding method. The shape of gene feature:", tr
```

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

In [24]:

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

Out[24]:

```
2574     BRCA1
907     PDGFRA
679     CDKN2A
1698      PMS2
2953      GNAS
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',
 'atr',
 'atrx',
 'aurka',
```

In [27]:

```
len(gene_vectorizer.get_feature_names())
```

Out[27]:

```
232
```

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, 232)
```

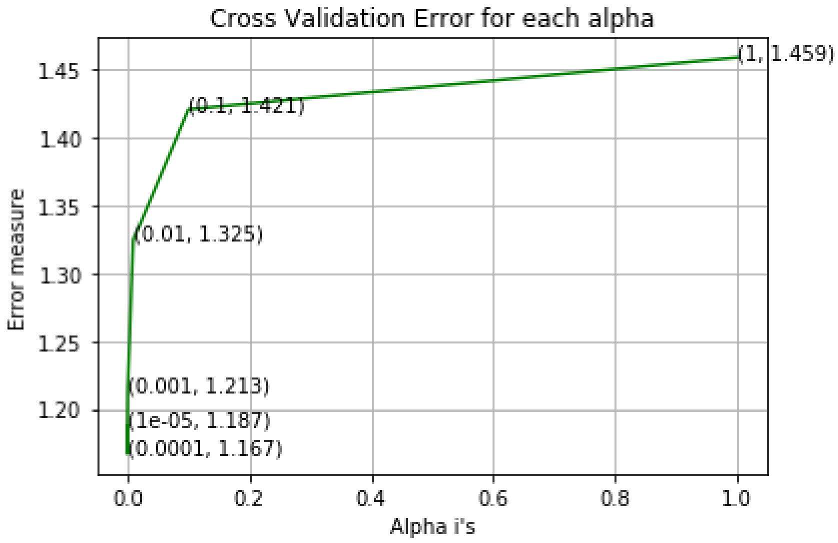
Q4. How good is this gene feature in predicting y_i?

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 [28]:

```
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: 1.1874045350318165
For values of alpha =  0.0001 The log loss is: 1.1669792995002348
For values of alpha =  0.001 The log loss is: 1.2131012380468342
For values of alpha =  0.01 The log loss is: 1.3246224651961658
For values of alpha =  0.1 The log loss is: 1.4205336205067303
For values of alpha =  1 The log loss is: 1.4588021980487909
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9941940431697661
For values of best alpha =  0.0001 The cross validation log loss is: 1.16697
92995002348
For values of best alpha =  0.0001 The test log loss is: 1.164871771751837
```

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 [29]:

```
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  233  ge
nes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 518 out of  532 : 97.36842105263158
```

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 [30]:

```
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 : 1938
Truncating_Mutations    57
Deletion                54
Amplification           43
Fusions                 24
Overexpression           2
T58I                     2
S222D                    2
G12V                     2
E17K                     2
V321M                    2
Name: Variation, dtype: int64
```

In [31]:

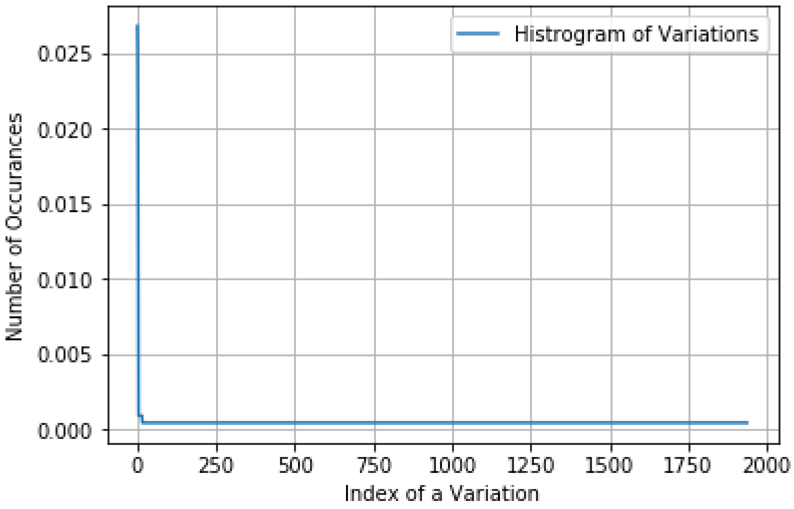
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in
```

```
Ans: There are 1938 different categories of variations in the train data, an
d they are distibuted as follows
```

In [32]:

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

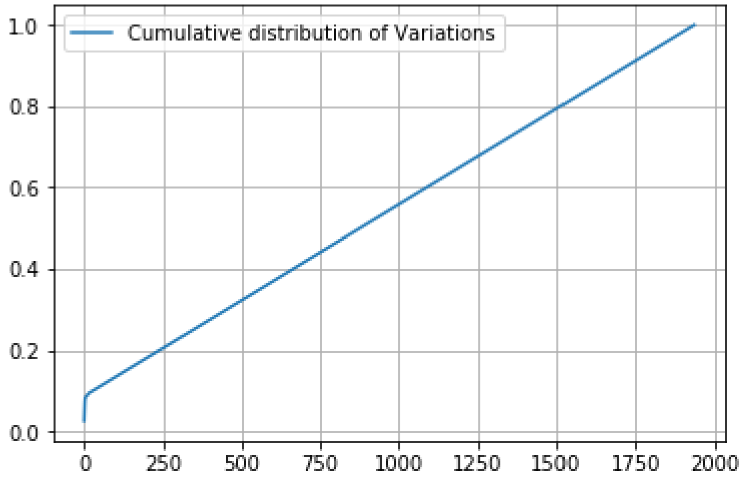
```
s= 2124
h=  [0.02683616 0.02542373 0.02024482 ... 0.00047081 0.00047081 0.00047081]
```



In [33]:

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

```
[0.02683616 0.05225989 0.07250471 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [34]:

```
# 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 [35]:

```
onverted feature using the response coding method. The shape of Variation feature:", train_v
```

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

In [36]:

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

In [37]:

```
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, 1963)
```

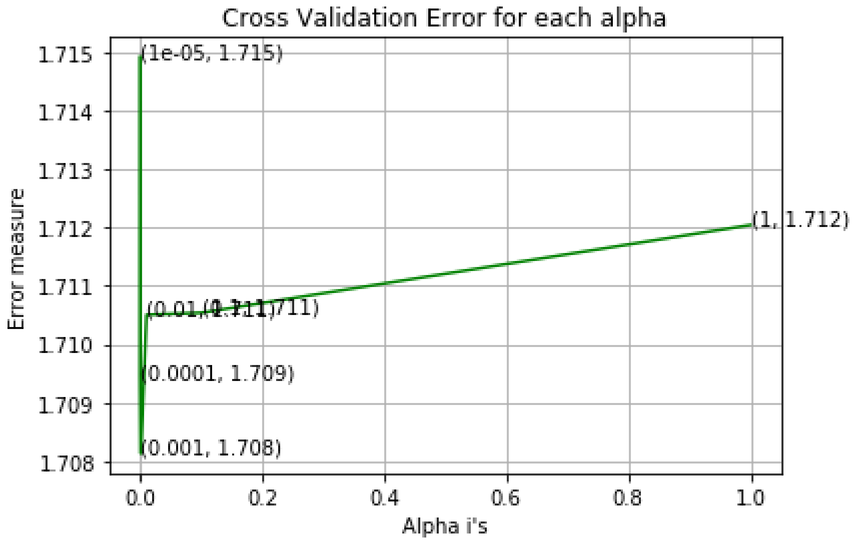
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [38]:

```
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: 1.7149130075903953
For values of alpha =  0.0001 The log loss is: 1.709433187929576
For values of alpha =  0.001 The log loss is: 1.7081408775156341
For values of alpha =  0.01 The log loss is: 1.7105061658939222
For values of alpha =  0.1 The log loss is: 1.710540324211648
For values of alpha =  1 The log loss is: 1.7120435282899213
```



```
For values of best alpha =  0.001 The train log loss is: 1.0232541297120663
For values of best alpha =  0.001 The cross validation log loss is: 1.708140
8775156341
For values of best alpha =  0.001 The test log loss is: 1.717814650819321
```

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 [39]:

```
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  1938  genes in test and cros
s validation data sets?
Ans
1. In test data 64 out of 665 : 9.624060150375941
2. In cross validation data 70 out of  532 : 13.157894736842104
```

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 [40]:

```
# 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 [41]:

```
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 [42]:

```
# dictionary.get(keyname, value)
# keyname   Required. The keyname of the item you want to return the value from
# value Optional. A value to return if the specified key does not exist.Default value None
```

In [43]:

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

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

In [44]:

```
text_fea_dict
```

Out[44]:

```
{'000': 14.068062438446978,
 '01': 9.86298205104205,
 '05': 9.315101563177922,
 '10': 56.68463830663071,
 '100': 21.755391779860794,
 '11': 29.65050454704271,
 '12': 31.497269371696103,
 '13': 24.98272850172027,
 '14': 27.572941042172886,
 '15': 29.85779448404735,
 '16': 22.69900224199266,
 '17': 21.10817969639606,
 '18': 22.6338172653546,
 '19': 20.31628125945686,
 '1996': 9.512196156393145,
 '1997': 11.671141675307313,
 '1998': 10.578087204772103,
 '1999': 9.33676089308645,
```

In [45]:

```
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 [46]:

```
#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 [47]:

```
# 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 [48]:

```
# 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 [49]:

```
print("Shape of train features with tfidf encoding :",train_text_feature_onehotCoding.shape
print("Shape of test features with tfidf encoding :",test_text_feature_onehotCoding.shape)
print("Shape of cv features with tfidf encoding :",cv_text_feature_onehotCoding.shape)
```

```
Shape of train features with tfidf encoding : (2124, 1000)
Shape of test features with tfidf encoding : (665, 1000)
Shape of cv features with tfidf encoding : (532, 1000)
```

Frequency of occurance of word in train data

In [56]:

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

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occu
fea_dict = dict(zip(list(train_features_text),train_fea_counts))


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

In [57]:

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

In [58]:

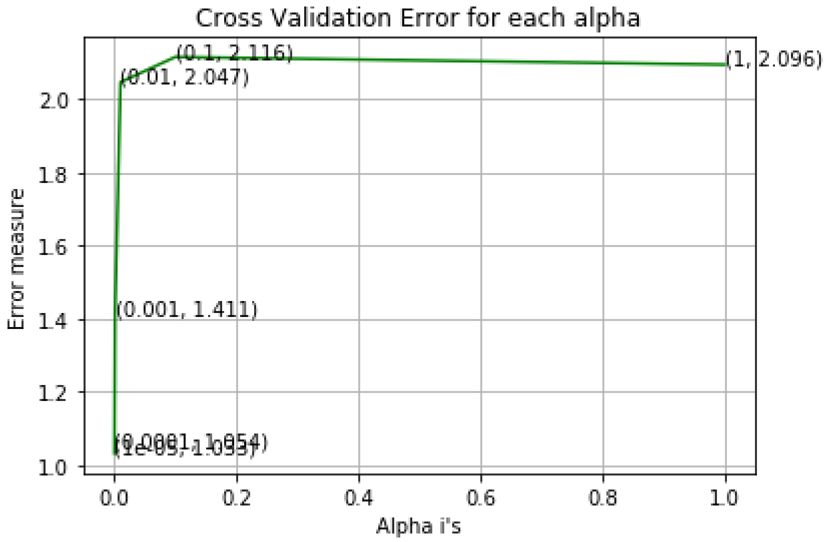
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 5220, 4: 4032, 5: 2982, 6: 2740, 9: 2065, 8: 2060, 7: 1920, 1
0: 1561, 12: 1338, 11: 1245, 16: 834, 14: 826, 13: 823, 15: 799, 18: 671,
20: 658, 17: 570, 19: 534, 24: 525, 21: 490, 22: 450, 26: 416, 40: 412, 2
8: 400, 27: 376, 25: 376, 23: 363, 30: 353, 32: 287, 50: 278, 29: 273, 36:
264, 33: 258, 38: 256, 31: 251, 37: 238, 39: 220, 34: 211, 42: 208, 35: 20
5, 41: 201, 44: 186, 45: 177, 48: 174, 43: 172, 46: 160, 56: 155, 51: 154,
57: 145, 52: 142, 49: 139, 60: 138, 53: 136, 54: 135, 47: 133, 67: 129, 5
5: 127, 62: 125, 63: 120, 72: 108, 61: 108, 59: 108, 58: 108, 66: 105, 76:
104, 65: 96, 70: 91, 64: 91, 69: 90, 80: 89, 81: 86, 75: 86, 73: 85, 71: 8
4, 74: 82, 100: 81, 79: 80, 78: 77, 77: 77, 95: 75, 84: 74, 68: 72, 88: 7
1, 99: 69, 87: 68, 116: 67, 82: 66, 85: 63, 114: 62, 83: 61, 91: 60, 98: 5
9, 93: 59, 92: 59, 86: 59, 90: 58, 113: 57, 97: 57, 111: 55, 94: 55, 109:
54, 102: 52, 101: 50, 89: 50, 120: 49, 119: 49, 130: 48, 96: 48, 105: 47,
103: 47, 107: 45, 144: 44, 106: 44, 108: 43, 104: 43, 138: 42, 150: 41, 13
3: 41, 110: 41, 152: 40, 122: 40, 121: 40, 112: 40, 136: 39, 148: 37, 139:
37, 129: 37, 124: 37, 117: 37, 180: 36, 135: 36, 126: 36, 115: 36, 141: 3
5, 125: 35, 146: 34, 145: 34, 151: 33, 149: 33, 132: 33, 190: 32, 127: 32,
174: 31, 161: 31, 128: 31, 169: 30, 143: 30, 140: 30, 137: 30, 202: 29, 18
5: 29, 164: 29, 163: 29, 156: 29, 134: 29, 123: 29, 118: 29, 192: 28, 155:
```

In [53]:

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

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

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

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



cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    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: 1.0333338976429496
For values of alpha =  0.0001 The log loss is: 1.05351627343654
For values of alpha =  0.001 The log loss is: 1.4111528491566432
For values of alpha =  0.01 The log loss is: 2.0473903457608835
For values of alpha =  0.1 The log loss is: 2.116443153317416
For values of alpha =  1 The log loss is: 2.0955224705639655
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7114688121829885
For values of best alpha =  1e-05 The cross validation log loss is: 1.033333
8976429496
For values of best alpha =  1e-05 The test log loss is: 1.0594415672149962
```

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

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

```
97.206 % of word of test data appeared in train data
97.914 % 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 [73]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

    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 [74]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_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 [75]:

```
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, 3195)
(number of data points * number of features) in test data =  (665, 3195)
(number of data points * number of features) in cross validation data = (53
2, 3195)
```

In [76]:

```
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)
```

4.1. Base Line Model

4.1.1. Naive Bayes

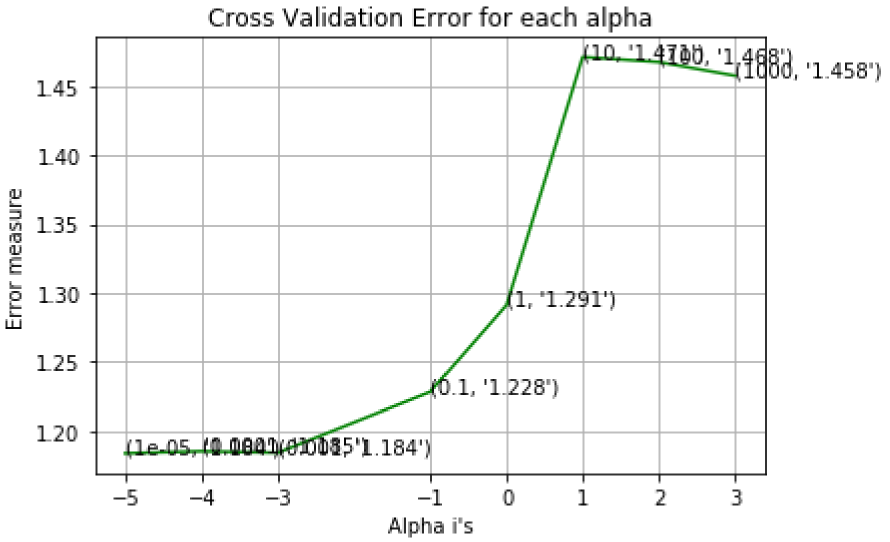
4.1.1.1. Hyper parameter tuning

In [77]:

```
# 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 : 1.1838176420293733
for alpha = 0.0001
Log Loss : 1.185344077538765
for alpha = 0.001
Log Loss : 1.1842424764827444
for alpha = 0.1
Log Loss : 1.228151843160466
for alpha = 1
Log Loss : 1.2912874120269802
for alpha = 10
Log Loss : 1.4712866399016093
for alpha = 100
Log Loss : 1.4676315532254174
for alpha = 1000
Log Loss : 1.457694272927647
```



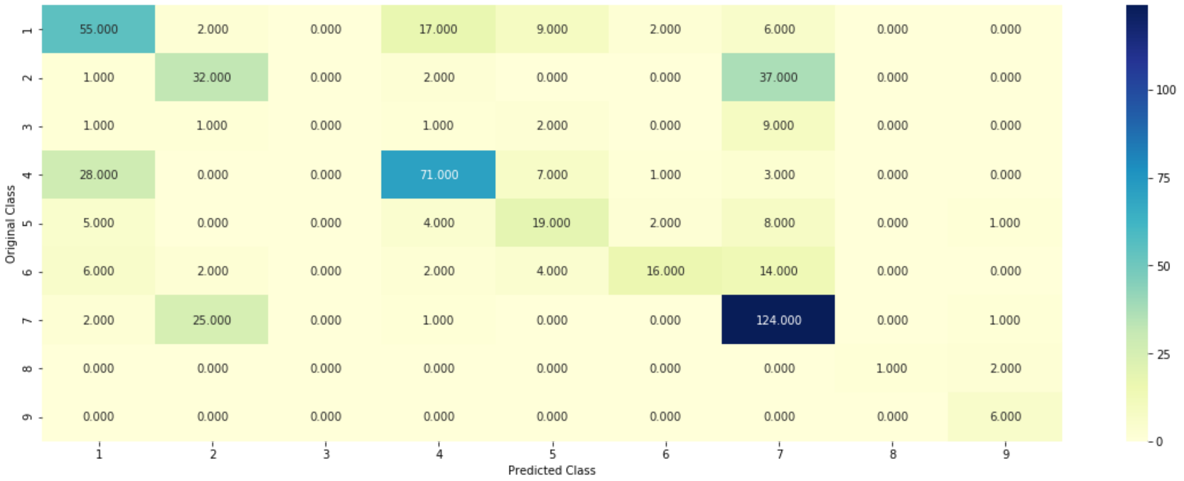
```
For values of best alpha =  1e-05 The train log loss is: 0.4526676705789037
For values of best alpha =  1e-05 The cross validation log loss is: 1.183817
6420293733
For values of best alpha =  1e-05 The test log loss is: 1.1824225949048461
```

4.1.1.2. Testing the model with best hyper paramters

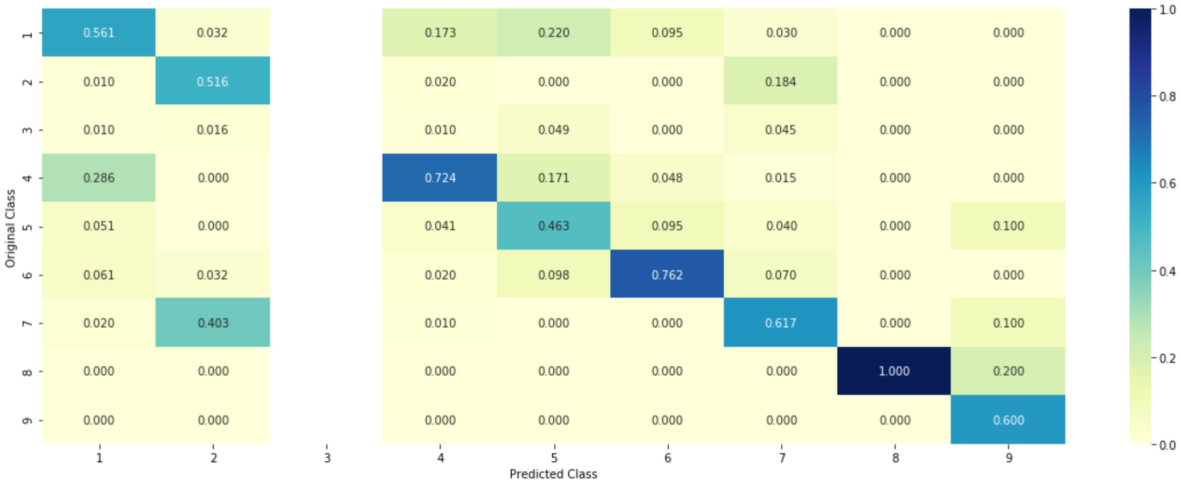
In [78]:

```
# 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()))
```

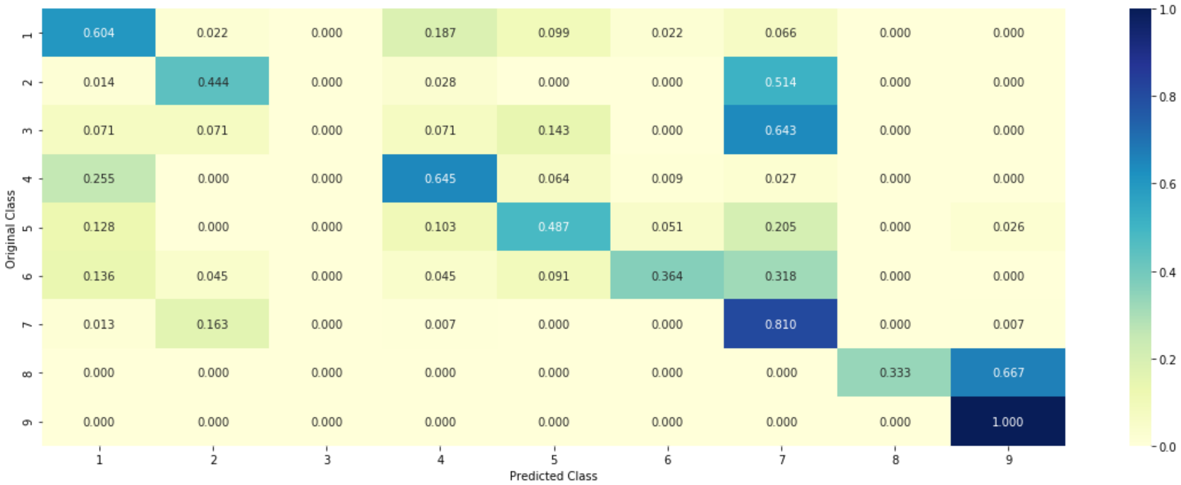
```
Log Loss : 1.1838176420293733
Number of missclassified point : 0.39097744360902253
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [80]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0945 0.04   0.0112 0.6964 0.0323 0.0308
0.0878 0.0035 0.0036]]
Actual Class : 4
--------------------------------------------------
12 Text feature [activity] present in test data point [True]
13 Text feature [protein] present in test data point [True]
14 Text feature [function] present in test data point [True]
15 Text feature [proteins] present in test data point [True]
16 Text feature [results] present in test data point [True]
17 Text feature [experiments] present in test data point [True]
18 Text feature [pten] present in test data point [True]
22 Text feature [missense] present in test data point [True]
23 Text feature [shown] present in test data point [True]
24 Text feature [whether] present in test data point [True]
25 Text feature [amino] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [type] present in test data point [True]
28 Text feature [acid] present in test data point [True]
29 Text feature [whereas] present in test data point [True]
30 Text feature [two] present in test data point [True]
31 Text feature [wild] present in test data point [True]
32 Text feature [important] present in test data point [True]
33 Text feature [suppressor] present in test data point [True]
34 Text feature [determined] present in test data point [True]
35 Text feature [may] present in test data point [True]
36 Text feature [described] present in test data point [True]
37 Text feature [mutations] present in test data point [True]
38 Text feature [indicate] present in test data point [True]
39 Text feature [functional] present in test data point [True]
40 Text feature [mammalian] present in test data point [True]
41 Text feature [determine] present in test data point [True]
42 Text feature [either] present in test data point [True]
43 Text feature [related] present in test data point [True]
44 Text feature [indicated] present in test data point [True]
45 Text feature [although] present in test data point [True]
46 Text feature [vitro] present in test data point [True]
47 Text feature [thus] present in test data point [True]
48 Text feature [levels] present in test data point [True]
49 Text feature [reduced] present in test data point [True]
50 Text feature [discussion] present in test data point [True]
51 Text feature [effects] present in test data point [True]
53 Text feature [30] present in test data point [True]
54 Text feature [however] present in test data point [True]
55 Text feature [one] present in test data point [True]
56 Text feature [similar] present in test data point [True]
57 Text feature [containing] present in test data point [True]
58 Text feature [three] present in test data point [True]
```

```
59 Text feature [analysis] present in test data point [True]
60 Text feature [previously] present in test data point [True]
61 Text feature [catalytic] present in test data point [True]
62 Text feature [therefore] present in test data point [True]
63 Text feature [loss] present in test data point [True]
64 Text feature [lower] present in test data point [True]
65 Text feature [effect] present in test data point [True]
66 Text feature [cells] present in test data point [True]
67 Text feature [transfected] present in test data point [True]
68 Text feature [result] present in test data point [True]
69 Text feature [transfection] present in test data point [True]
70 Text feature [purified] present in test data point [True]
71 Text feature [using] present in test data point [True]
72 Text feature [terminal] present in test data point [True]
73 Text feature [several] present in test data point [True]
74 Text feature [associated] present in test data point [True]
75 Text feature [addition] present in test data point [True]
76 Text feature [introduction] present in test data point [True]
77 Text feature [10] present in test data point [True]
78 Text feature [assay] present in test data point [True]
79 Text feature [generated] present in test data point [True]
80 Text feature [bind] present in test data point [True]
81 Text feature [ability] present in test data point [True]
82 Text feature [high] present in test data point [True]
83 Text feature [site] present in test data point [True]
84 Text feature [acids] present in test data point [True]
85 Text feature [found] present in test data point [True]
86 Text feature [sds] present in test data point [True]
87 Text feature [suggest] present in test data point [True]
88 Text feature [used] present in test data point [True]
89 Text feature [buffer] present in test data point [True]
90 Text feature [fact] present in test data point [True]
91 Text feature [see] present in test data point [True]
92 Text feature [figure] present in test data point [True]
93 Text feature [expressed] present in test data point [True]
94 Text feature [expression] present in test data point [True]
95 Text feature [mm] present in test data point [True]
96 Text feature [control] present in test data point [True]
97 Text feature [yeast] present in test data point [True]
98 Text feature [suggesting] present in test data point [True]
99 Text feature [tagged] present in test data point [True]
Out of the top  100  features  84 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [83]:

```
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 : 4
Predicted Class Probabilities: [[0.1527 0.0414 0.0115 0.6305 0.0339 0.0319
0.0907 0.0037 0.0037]]
Actual Class : 1
--------------------------------------------------
12 Text feature [activity] present in test data point [True]
13 Text feature [protein] present in test data point [True]
14 Text feature [function] present in test data point [True]
15 Text feature [proteins] present in test data point [True]
16 Text feature [results] present in test data point [True]
17 Text feature [experiments] present in test data point [True]
22 Text feature [missense] present in test data point [True]
23 Text feature [shown] present in test data point [True]
24 Text feature [whether] present in test data point [True]
25 Text feature [amino] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [type] present in test data point [True]
28 Text feature [acid] present in test data point [True]
29 Text feature [whereas] present in test data point [True]
30 Text feature [two] present in test data point [True]
31 Text feature [wild] present in test data point [True]
32 Text feature [important] present in test data point [True]
33 Text feature [suppressor] present in test data point [True]
34 Text feature [determined] present in test data point [True]
35 Text feature [may] present in test data point [True]
36 Text feature [described] present in test data point [True]
37 Text feature [mutations] present in test data point [True]
38 Text feature [indicate] present in test data point [True]
39 Text feature [functional] present in test data point [True]
40 Text feature [mammalian] present in test data point [True]
41 Text feature [determine] present in test data point [True]
42 Text feature [either] present in test data point [True]
43 Text feature [related] present in test data point [True]
44 Text feature [indicated] present in test data point [True]
45 Text feature [although] present in test data point [True]
46 Text feature [vitro] present in test data point [True]
47 Text feature [thus] present in test data point [True]
48 Text feature [levels] present in test data point [True]
49 Text feature [reduced] present in test data point [True]
50 Text feature [discussion] present in test data point [True]
51 Text feature [effects] present in test data point [True]
53 Text feature [30] present in test data point [True]
54 Text feature [however] present in test data point [True]
55 Text feature [one] present in test data point [True]
56 Text feature [similar] present in test data point [True]
57 Text feature [containing] present in test data point [True]
58 Text feature [three] present in test data point [True]
59 Text feature [analysis] present in test data point [True]
```

```
60 Text feature [previously] present in test data point [True]
61 Text feature [catalytic] present in test data point [True]
62 Text feature [therefore] present in test data point [True]
63 Text feature [loss] present in test data point [True]
64 Text feature [lower] present in test data point [True]
65 Text feature [effect] present in test data point [True]
66 Text feature [cells] present in test data point [True]
67 Text feature [transfected] present in test data point [True]
68 Text feature [result] present in test data point [True]
69 Text feature [transfection] present in test data point [True]
70 Text feature [purified] present in test data point [True]
71 Text feature [using] present in test data point [True]
72 Text feature [terminal] present in test data point [True]
73 Text feature [several] present in test data point [True]
74 Text feature [associated] present in test data point [True]
75 Text feature [addition] present in test data point [True]
77 Text feature [10] present in test data point [True]
78 Text feature [assay] present in test data point [True]
79 Text feature [generated] present in test data point [True]
80 Text feature [bind] present in test data point [True]
81 Text feature [ability] present in test data point [True]
82 Text feature [high] present in test data point [True]
83 Text feature [site] present in test data point [True]
84 Text feature [acids] present in test data point [True]
85 Text feature [found] present in test data point [True]
86 Text feature [sds] present in test data point [True]
87 Text feature [suggest] present in test data point [True]
88 Text feature [used] present in test data point [True]
89 Text feature [buffer] present in test data point [True]
90 Text feature [fact] present in test data point [True]
91 Text feature [see] present in test data point [True]
92 Text feature [figure] present in test data point [True]
93 Text feature [expressed] present in test data point [True]
94 Text feature [expression] present in test data point [True]
95 Text feature [mm] present in test data point [True]
96 Text feature [control] present in test data point [True]
97 Text feature [yeast] present in test data point [True]
98 Text feature [suggesting] present in test data point [True]
99 Text feature [tagged] present in test data point [True]
Out of the top  100  features  82 are present in query point
```

4.2. K Nearest Neighbour Classification

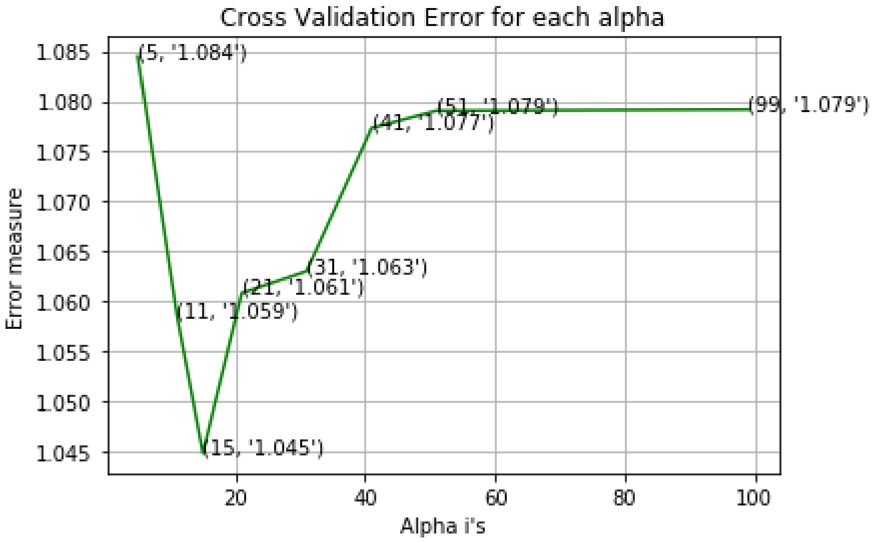
4.2.1. Hyper parameter tuning

In [84]:

```
# 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 : 1.0844734657001394
for alpha = 11
Log Loss : 1.0585967839876178
for alpha = 15
Log Loss : 1.0448448943413942
for alpha = 21
Log Loss : 1.060845318044067
for alpha = 31
Log Loss : 1.0630281436605726
for alpha = 41
Log Loss : 1.0773061913776567
for alpha = 51
Log Loss : 1.07904518011301
for alpha = 99
Log Loss : 1.0791675335529907
```



```
For values of best alpha =  15 The train log loss is: 0.6690519816966605
For values of best alpha =  15 The cross validation log loss is: 1.0448448
```

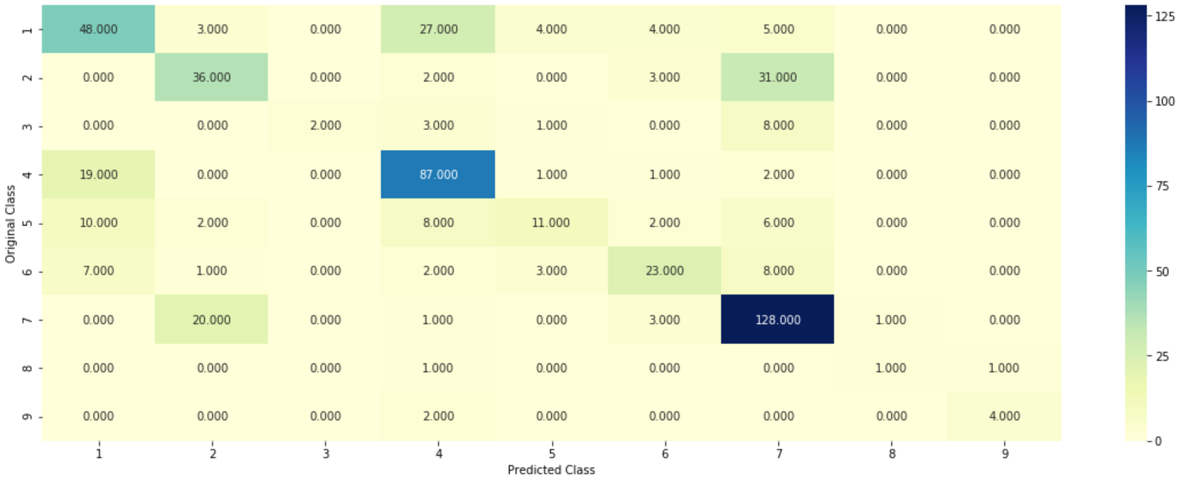
```
943413942
For values of best alpha =  15 The test log loss is: 1.0569878036283742
```

4.2.2. Testing the model with best hyper paramters

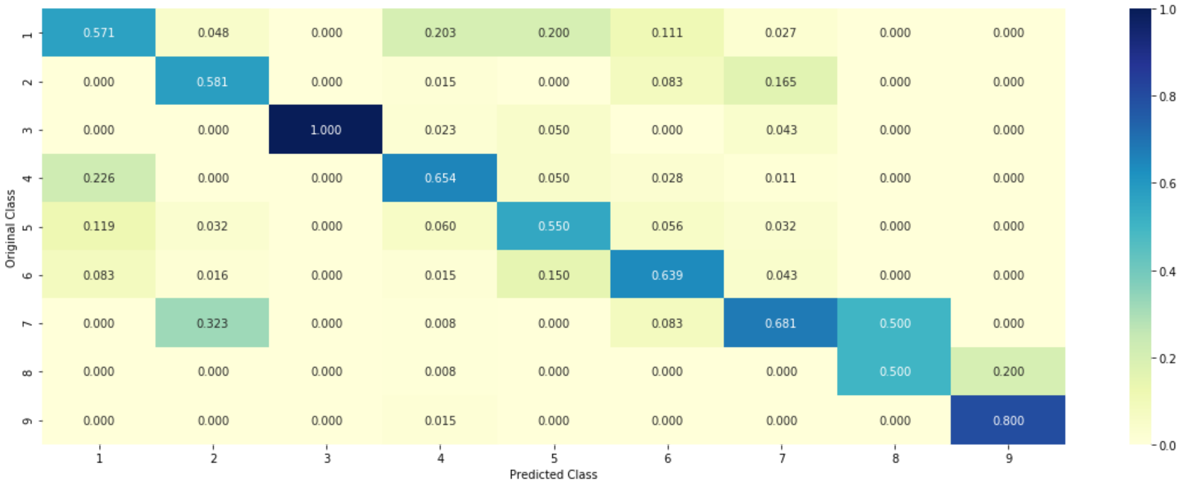
In [85]:

```
# 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_
```

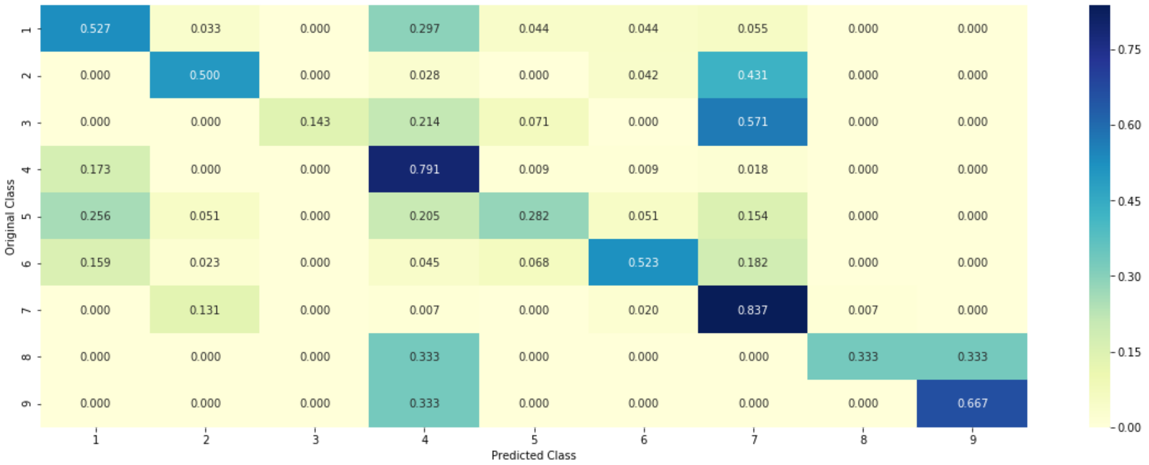
```
Log loss : 1.0448448943413942
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

In [86]:

```
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[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 ",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 : 4
Actual Class : 4
The  15  nearest neighbours of the test points belongs to classes [4 4 4 4 4
4 4 4 4 4 4 1 1 4 4]
Fequency of nearest points : Counter({4: 13, 1: 2})
```

In [71]:

```
#kneighbors(self[, X, n_neighbors, …])->Finds the K-neighbors of a point.
```

4.2.4. Sample Query Point-2

In [87]:

```
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 : 4
Actual Class : 1
the k value for knn is 15 and the nearest neighbours of the test points belo
ngs to classes [1 4 4 4 1 4 4 4 5 4 5 4 4 4 4]
Fequency of nearest points : Counter({4: 11, 1: 2, 5: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

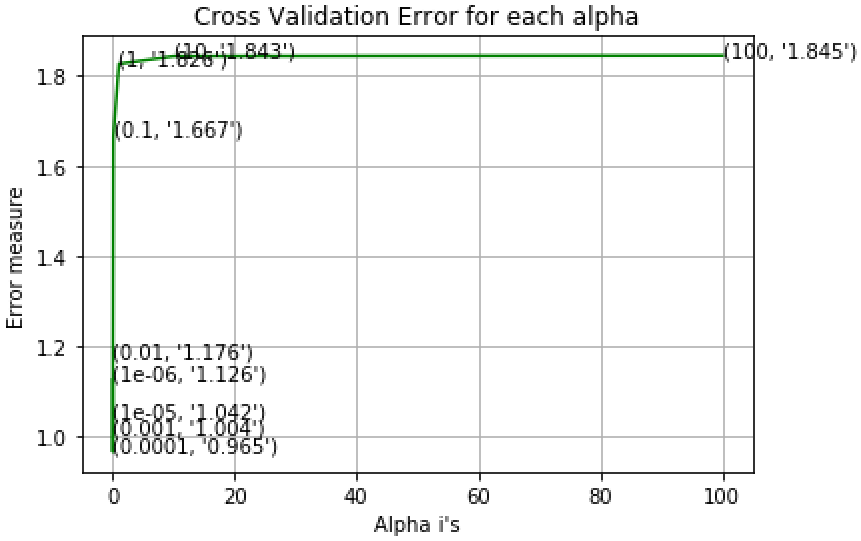
4.3.1.1. Hyper paramter tuning

In [88]:

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

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

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



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

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

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


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

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1257867028988717
for alpha = 1e-05
Log Loss : 1.0423139312524583
for alpha = 0.0001
Log Loss : 0.964570905172389
for alpha = 0.001
Log Loss : 1.0039470657121432
for alpha = 0.01
Log Loss : 1.1764289591738117
for alpha = 0.1
Log Loss : 1.6672831647212034
for alpha = 1
Log Loss : 1.826353025441565
for alpha = 10
Log Loss : 1.8433209051746489
for alpha = 100
Log Loss : 1.84488632755673
```



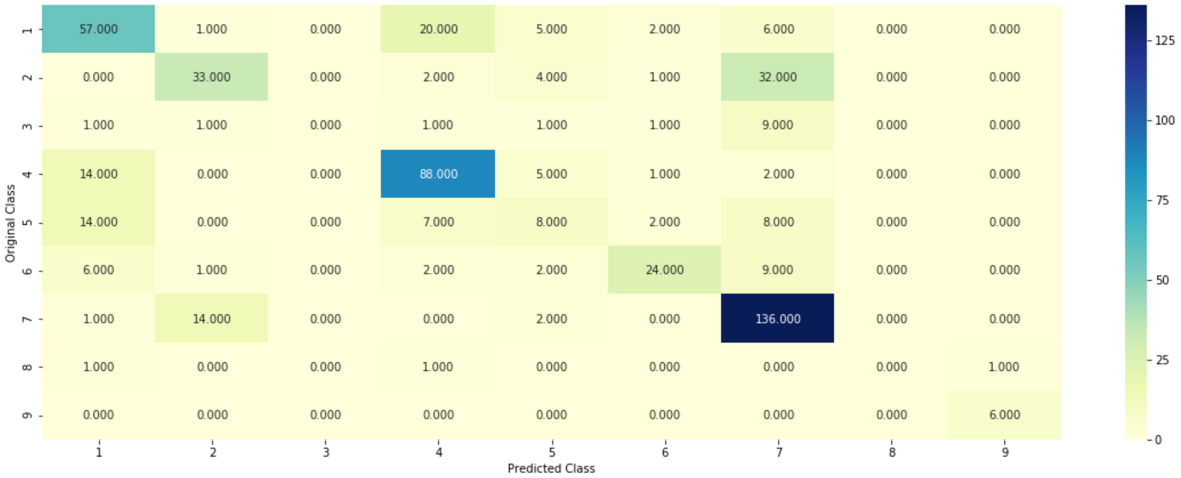
```
For values of best alpha =  0.0001 The train log loss is: 0.40840833335137249
For values of best alpha =  0.0001 The cross validation log loss is: 0.96457
0905172389
For values of best alpha =  0.0001 The test log loss is: 1.0042182582233667
```

4.3.1.2. Testing the model with best hyper paramters

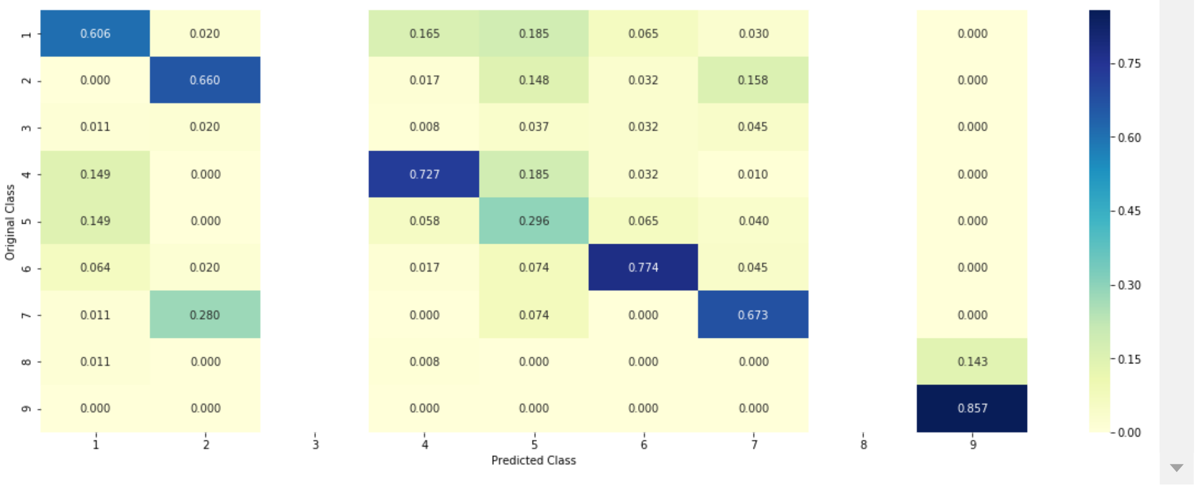
In [89]:

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

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

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

```
Log loss : 0.964570905172389
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```

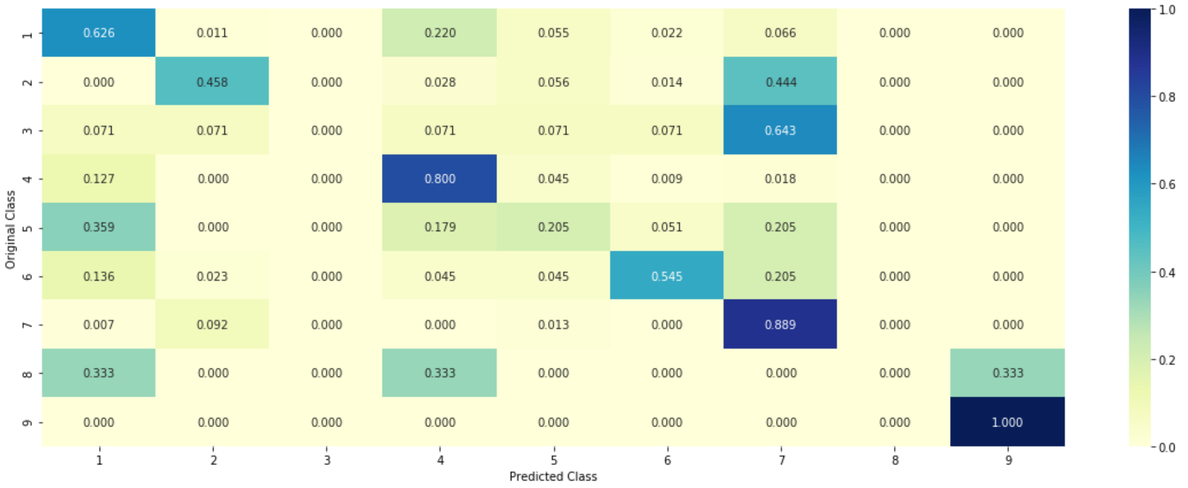


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.606	0.020		0.165	0.185	0.065	0.030		0.000
2	0.000	0.660		0.017	0.148	0.032	0.158		0.000
3	0.011	0.020		0.008	0.037	0.032	0.045		0.000
4	0.149	0.000		0.727	0.185	0.032	0.010		0.000
5	0.149	0.000		0.058	0.296	0.065	0.040		0.000
6	0.064	0.020		0.017	0.074	0.774	0.045		0.000
7	0.011	0.280		0.000	0.074	0.000	0.673		0.000
8	0.011	0.000		0.008	0.000	0.000	0.000		0.143
9	0.000	0.000		0.000	0.000	0.000	0.000		0.857

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



	1	2	3	4	5	6	7	8	9
1	0.626	0.011	0.000	0.220	0.055	0.022	0.066	0.000	0.000
2	0.000	0.458	0.000	0.028	0.056	0.014	0.444	0.000	0.000
3	0.071	0.071	0.000	0.071	0.071	0.071	0.643	0.000	0.000
4	0.127	0.000	0.000	0.800	0.045	0.009	0.018	0.000	0.000
5	0.359	0.000	0.000	0.179	0.205	0.051	0.205	0.000	0.000
6	0.136	0.023	0.000	0.045	0.045	0.545	0.205	0.000	0.000
7	0.007	0.092	0.000	0.000	0.013	0.000	0.889	0.000	0.000
8	0.333	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.333
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

4.3.1.3. Feature Importance

In [90]:

```
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 [91]:

```
# 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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].ilo
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1249 0.021  0.0097 0.7618 0.0158 0.0161
0.0354 0.0074 0.0079]]
Actual Class : 4
--------------------------------------------------
35 Text feature [suppressor] present in test data point [True]
121 Text feature [mammalian] present in test data point [True]
136 Text feature [screening] present in test data point [True]
162 Text feature [missense] present in test data point [True]
231 Text feature [values] present in test data point [True]
239 Text feature [fold] present in test data point [True]
243 Text feature [mm] present in test data point [True]
244 Text feature [yeast] present in test data point [True]
249 Text feature [formed] present in test data point [True]
250 Text feature [transfected] present in test data point [True]
254 Text feature [five] present in test data point [True]
263 Text feature [high] present in test data point [True]
267 Text feature [see] present in test data point [True]
272 Text feature [washed] present in test data point [True]
291 Text feature [dominant] present in test data point [True]
297 Text feature [activating] present in test data point [True]
304 Text feature [ca] present in test data point [True]
315 Text feature [resistance] present in test data point [True]
326 Text feature [caused] present in test data point [True]
330 Text feature [would] present in test data point [True]
340 Text feature [greater] present in test data point [True]
342 Text feature [independent] present in test data point [True]
351 Text feature [harboring] present in test data point [True]
356 Text feature [endogenous] present in test data point [True]
357 Text feature [transfection] present in test data point [True]
358 Text feature [patients] present in test data point [True]
379 Text feature [localization] present in test data point [True]
380 Text feature [lack] present in test data point [True]
397 Text feature [region] present in test data point [True]
399 Text feature [substrate] present in test data point [True]
401 Text feature [carcinoma] present in test data point [True]
403 Text feature [ability] present in test data point [True]
408 Text feature [2013] present in test data point [True]
411 Text feature [suggesting] present in test data point [True]
412 Text feature [position] present in test data point [True]
415 Text feature [bind] present in test data point [True]
423 Text feature [stability] present in test data point [True]
428 Text feature [kinases] present in test data point [True]
437 Text feature [taken] present in test data point [True]
440 Text feature [western] present in test data point [True]
```

```
446 Text feature [determine] present in test data point [True]
451 Text feature [stable] present in test data point [True]
454 Text feature [1998] present in test data point [True]
456 Text feature [age] present in test data point [True]
466 Text feature [versus] present in test data point [True]
467 Text feature [normalized] present in test data point [True]
474 Text feature [state] present in test data point [True]
479 Text feature [concentration] present in test data point [True]
486 Text feature [fusion] present in test data point [True]
491 Text feature [many] present in test data point [True]
492 Text feature [nuclear] present in test data point [True]
496 Text feature [suppression] present in test data point [True]
Out of the top  500  features  52 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [92]:

```
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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].ilc
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3164 0.0391 0.0026 0.5925 0.0283 0.0032
0.0116 0.0041 0.0022]]
Actual Class : 1
--------------------------------------------------
35 Text feature [suppressor] present in test data point [True]
121 Text feature [mammalian] present in test data point [True]
142 Text feature [value] present in test data point [True]
162 Text feature [missense] present in test data point [True]
163 Text feature [inactivation] present in test data point [True]
231 Text feature [values] present in test data point [True]
235 Text feature [damage] present in test data point [True]
239 Text feature [fold] present in test data point [True]
242 Text feature [unable] present in test data point [True]
243 Text feature [mm] present in test data point [True]
244 Text feature [yeast] present in test data point [True]
249 Text feature [formed] present in test data point [True]
250 Text feature [transfected] present in test data point [True]
254 Text feature [five] present in test data point [True]
263 Text feature [high] present in test data point [True]
267 Text feature [see] present in test data point [True]
272 Text feature [washed] present in test data point [True]
286 Text feature [tumorigenesis] present in test data point [True]
291 Text feature [dominant] present in test data point [True]
297 Text feature [activating] present in test data point [True]
304 Text feature [ca] present in test data point [True]
315 Text feature [resistance] present in test data point [True]
326 Text feature [caused] present in test data point [True]
330 Text feature [would] present in test data point [True]
340 Text feature [greater] present in test data point [True]
342 Text feature [independent] present in test data point [True]
343 Text feature [germline] present in test data point [True]
351 Text feature [harboring] present in test data point [True]
356 Text feature [endogenous] present in test data point [True]
357 Text feature [transfection] present in test data point [True]
358 Text feature [patients] present in test data point [True]
380 Text feature [lack] present in test data point [True]
388 Text feature [hotspots] present in test data point [True]
397 Text feature [region] present in test data point [True]
399 Text feature [substrate] present in test data point [True]
401 Text feature [carcinoma] present in test data point [True]
402 Text feature [driven] present in test data point [True]
403 Text feature [ability] present in test data point [True]
408 Text feature [2013] present in test data point [True]
411 Text feature [suggesting] present in test data point [True]
412 Text feature [position] present in test data point [True]
415 Text feature [bind] present in test data point [True]
423 Text feature [stability] present in test data point [True]
```

```
427 Text feature [transforming] present in test data point [True]
434 Text feature [consequences] present in test data point [True]
437 Text feature [taken] present in test data point [True]
438 Text feature [hif] present in test data point [True]
440 Text feature [western] present in test data point [True]
446 Text feature [determine] present in test data point [True]
451 Text feature [stable] present in test data point [True]
458 Text feature [confer] present in test data point [True]
466 Text feature [versus] present in test data point [True]
467 Text feature [normalized] present in test data point [True]
474 Text feature [state] present in test data point [True]
476 Text feature [calculated] present in test data point [True]
479 Text feature [concentration] present in test data point [True]
491 Text feature [many] present in test data point [True]
Out of the top  500  features  57 are present in query point
```

4.3.2. Without Class balancing

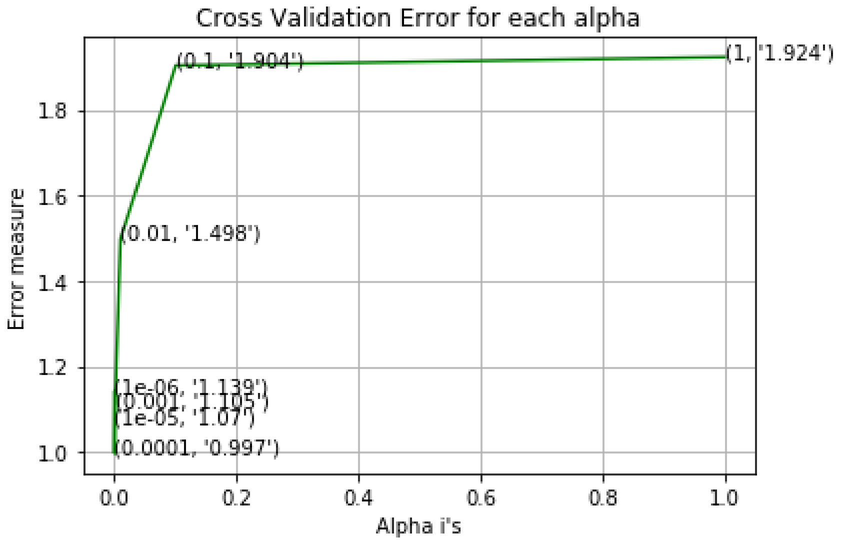
4.3.2.1. Hyper paramter tuning

In [93]:

```
# 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 : 1.1394116818348166
for alpha = 1e-05
Log Loss : 1.0696746893719769
for alpha = 0.0001
Log Loss : 0.9968764219055808
for alpha = 0.001
Log Loss : 1.1052662064046785
for alpha = 0.01
Log Loss : 1.4980778087746605
for alpha = 0.1
Log Loss : 1.9040311487780166
for alpha = 1
Log Loss : 1.9237267406438967
```



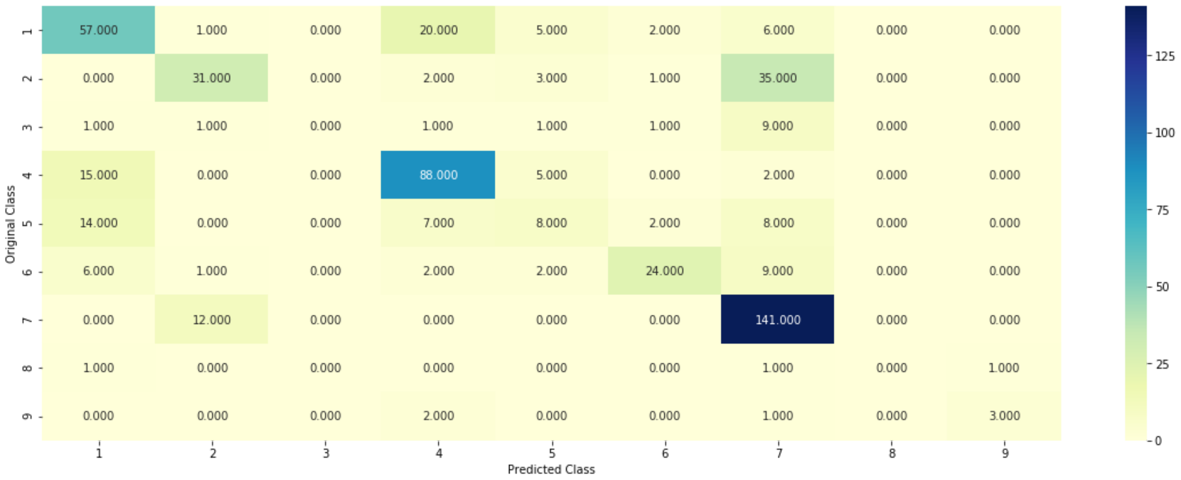
```
For values of best alpha =  0.0001 The train log loss is: 0.3988942895387268
For values of best alpha =  0.0001 The cross validation log loss is: 0.99687
64219055808
For values of best alpha =  0.0001 The test log loss is: 1.025832779416868
```

4.3.2.2. Testing model with best hyper parameters

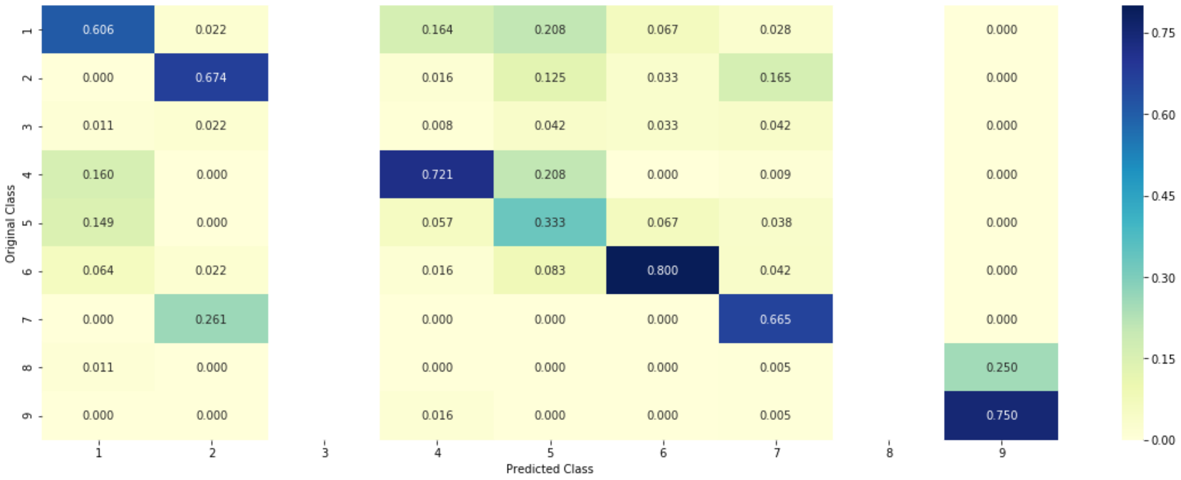
In [94]:

```
# 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:
#------------------------------

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
```

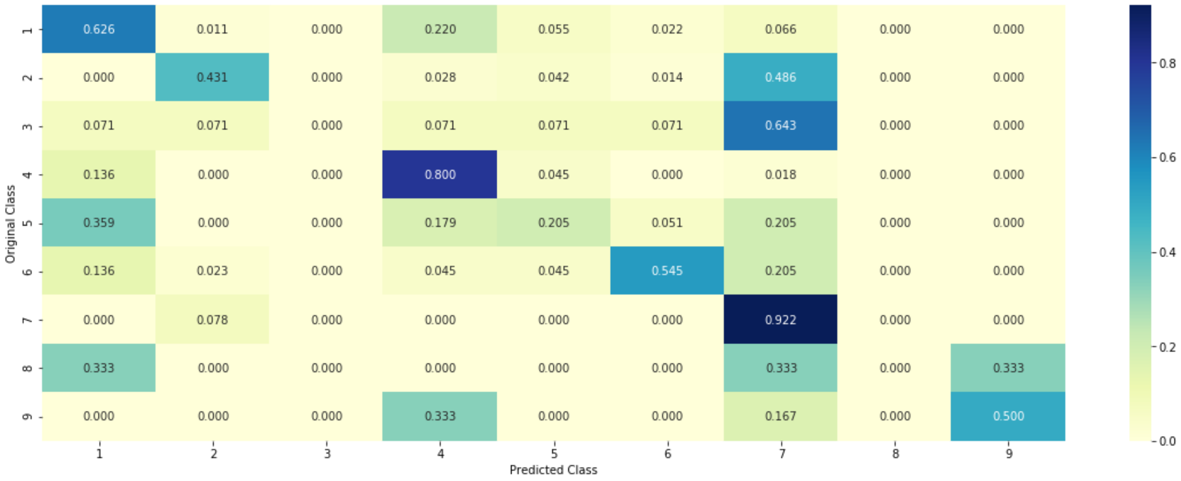
```
Log loss : 0.99687642190558O8
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [95]:

```
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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].ilc
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1302 0.0207 0.0101 0.7609 0.0159 0.0162
0.0328 0.0066 0.0066]]
Actual Class : 4
--------------------------------------------------
37 Text feature [suppressor] present in test data point [True]
120 Text feature [screening] present in test data point [True]
136 Text feature [mammalian] present in test data point [True]
213 Text feature [formed] present in test data point [True]
219 Text feature [missense] present in test data point [True]
239 Text feature [values] present in test data point [True]
241 Text feature [fold] present in test data point [True]
266 Text feature [washed] present in test data point [True]
269 Text feature [five] present in test data point [True]
273 Text feature [yeast] present in test data point [True]
279 Text feature [ca] present in test data point [True]
295 Text feature [high] present in test data point [True]
304 Text feature [see] present in test data point [True]
309 Text feature [caused] present in test data point [True]
312 Text feature [would] present in test data point [True]
317 Text feature [transfected] present in test data point [True]
325 Text feature [mm] present in test data point [True]
326 Text feature [endogenous] present in test data point [True]
332 Text feature [resistance] present in test data point [True]
340 Text feature [bind] present in test data point [True]
347 Text feature [dominant] present in test data point [True]
365 Text feature [substrate] present in test data point [True]
370 Text feature [region] present in test data point [True]
374 Text feature [determine] present in test data point [True]
378 Text feature [independent] present in test data point [True]
386 Text feature [suggesting] present in test data point [True]
387 Text feature [patients] present in test data point [True]
391 Text feature [transfection] present in test data point [True]
398 Text feature [kinases] present in test data point [True]
399 Text feature [greater] present in test data point [True]
400 Text feature [position] present in test data point [True]
403 Text feature [harboring] present in test data point [True]
406 Text feature [localization] present in test data point [True]
408 Text feature [lack] present in test data point [True]
411 Text feature [2013] present in test data point [True]
414 Text feature [taken] present in test data point [True]
416 Text feature [stable] present in test data point [True]
417 Text feature [ability] present in test data point [True]
418 Text feature [activating] present in test data point [True]
419 Text feature [carcinoma] present in test data point [True]
435 Text feature [western] present in test data point [True]
```

```
452 Text feature [versus] present in test data point [True]
456 Text feature [normalized] present in test data point [True]
465 Text feature [represent] present in test data point [True]
474 Text feature [1998] present in test data point [True]
478 Text feature [within] present in test data point [True]
479 Text feature [age] present in test data point [True]
481 Text feature [example] present in test data point [True]
482 Text feature [box] present in test data point [True]
485 Text feature [suppression] present in test data point [True]
486 Text feature [many] present in test data point [True]
492 Text feature [stability] present in test data point [True]
Out of the top  500  features  52 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [96]:

```
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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].ilo
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3267 0.039  0.0023 0.5899 0.0241 0.0029
0.0114 0.0026 0.0011]]
Actual Class : 1
--------------------------------------------------
37 Text feature [suppressor] present in test data point [True]
134 Text feature [value] present in test data point [True]
136 Text feature [mammalian] present in test data point [True]
190 Text feature [tumorigenesis] present in test data point [True]
192 Text feature [inactivation] present in test data point [True]
195 Text feature [damage] present in test data point [True]
213 Text feature [formed] present in test data point [True]
219 Text feature [missense] present in test data point [True]
235 Text feature [unable] present in test data point [True]
239 Text feature [values] present in test data point [True]
241 Text feature [fold] present in test data point [True]
266 Text feature [washed] present in test data point [True]
269 Text feature [five] present in test data point [True]
273 Text feature [yeast] present in test data point [True]
279 Text feature [ca] present in test data point [True]
295 Text feature [high] present in test data point [True]
304 Text feature [see] present in test data point [True]
309 Text feature [caused] present in test data point [True]
312 Text feature [would] present in test data point [True]
317 Text feature [transfected] present in test data point [True]
325 Text feature [mm] present in test data point [True]
326 Text feature [endogenous] present in test data point [True]
332 Text feature [resistance] present in test data point [True]
340 Text feature [bind] present in test data point [True]
347 Text feature [dominant] present in test data point [True]
365 Text feature [substrate] present in test data point [True]
370 Text feature [region] present in test data point [True]
374 Text feature [determine] present in test data point [True]
378 Text feature [independent] present in test data point [True]
379 Text feature [germline] present in test data point [True]
386 Text feature [suggesting] present in test data point [True]
387 Text feature [patients] present in test data point [True]
391 Text feature [transfection] present in test data point [True]
399 Text feature [greater] present in test data point [True]
400 Text feature [position] present in test data point [True]
403 Text feature [harboring] present in test data point [True]
404 Text feature [consequences] present in test data point [True]
408 Text feature [lack] present in test data point [True]
411 Text feature [2013] present in test data point [True]
414 Text feature [taken] present in test data point [True]
416 Text feature [stable] present in test data point [True]
417 Text feature [ability] present in test data point [True]
418 Text feature [activating] present in test data point [True]
```

```
419 Text feature [carcinoma] present in test data point [True]
423 Text feature [hotspots] present in test data point [True]
435 Text feature [western] present in test data point [True]
445 Text feature [transforming] present in test data point [True]
452 Text feature [versus] present in test data point [True]
454 Text feature [hif] present in test data point [True]
456 Text feature [normalized] present in test data point [True]
465 Text feature [represent] present in test data point [True]
469 Text feature [calculated] present in test data point [True]
471 Text feature [driven] present in test data point [True]
478 Text feature [within] present in test data point [True]
482 Text feature [box] present in test data point [True]
486 Text feature [many] present in test data point [True]
492 Text feature [stability] present in test data point [True]
Out of the top  500  features  57 are present in query point
```

4.4. Linear Support Vector Machines

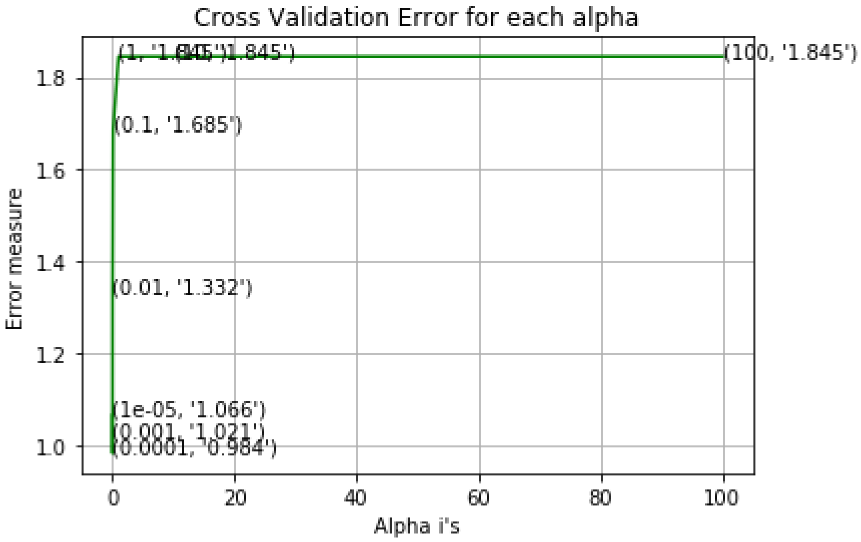
4.4.1. Hyper paramter tuning

In [97]:

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

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

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



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', 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 : 1.0662928193265757
for C = 0.0001
Log Loss : 0.9835045625237188
for C = 0.001
Log Loss : 1.0209695208365768
for C = 0.01
Log Loss : 1.3321470248947676
for C = 0.1
Log Loss : 1.6847867501363902
for C = 1
Log Loss : 1.8449759532611654
for C = 10
Log Loss : 1.8449755251185258
for C = 100
Log Loss : 1.8449758589669134
```



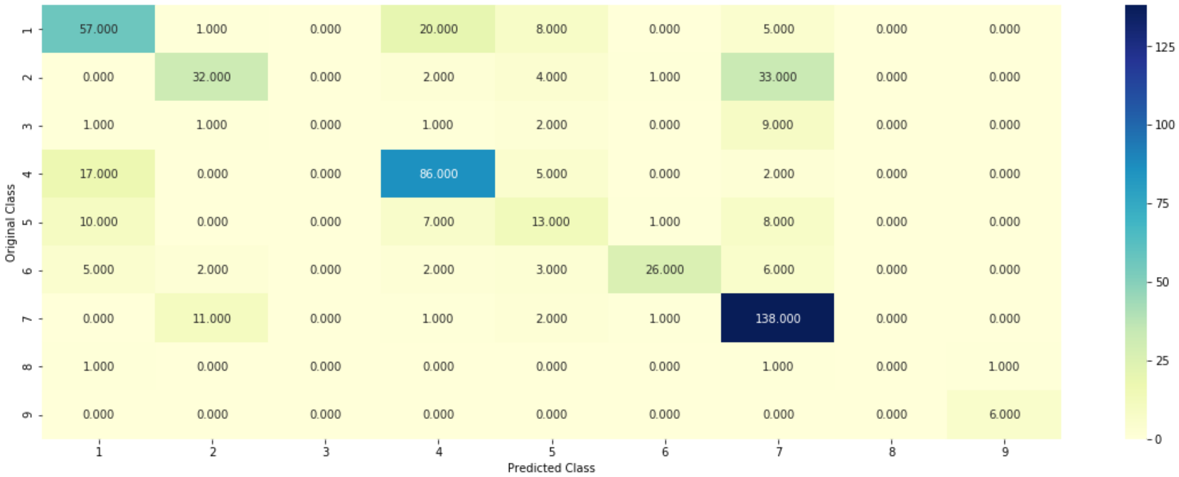
```
For values of best alpha =  0.0001 The train log loss is: 0.3344451234746053
For values of best alpha =  0.0001 The cross validation log loss is: 0.98350
45625237188
For values of best alpha =  0.0001 The test log loss is: 1.036912657057396
```

4.4.2. Testing model with best hyper parameters

In [98]:

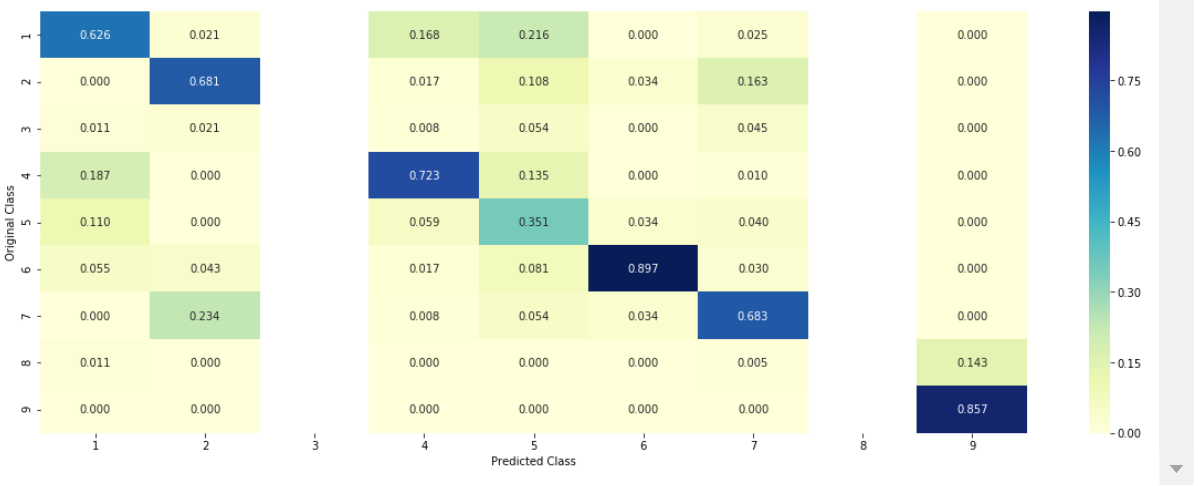
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/

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

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


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

```
Log loss : 0.9835045625237188
Number of mis-classified points : 0.32706766917293234
-------------------- Confusion matrix --------------------
```

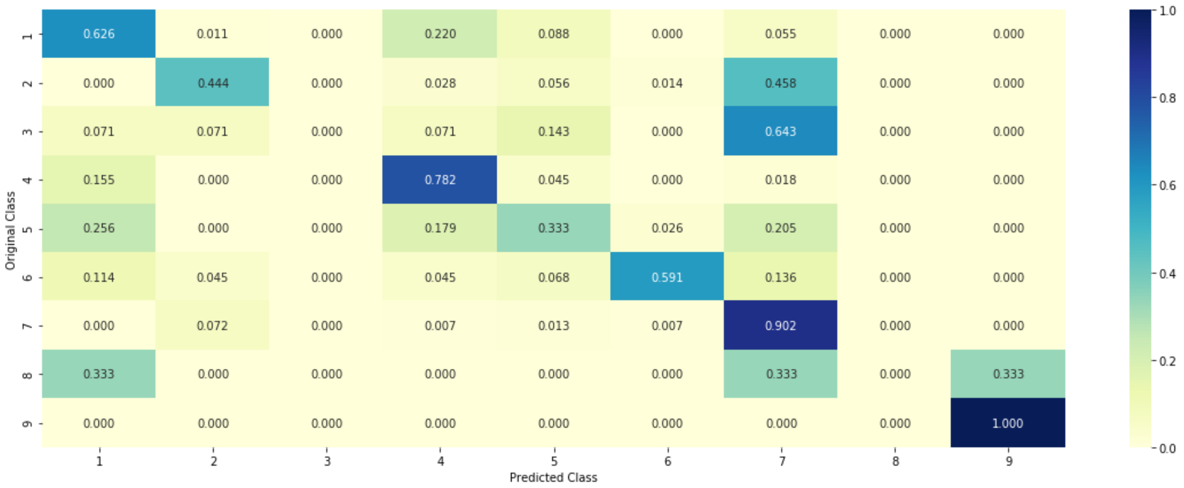


	1	2	3	4	5	6	7	8	9
1	57.000	1.000	0.000	20.000	8.000	0.000	5.000	0.000	0.000
2	0.000	32.000	0.000	2.000	4.000	1.000	33.000	0.000	0.000
3	1.000	1.000	0.000	1.000	2.000	0.000	9.000	0.000	0.000
4	17.000	0.000	0.000	86.000	5.000	0.000	2.000	0.000	0.000
5	10.000	0.000	0.000	7.000	13.000	1.000	8.000	0.000	0.000
6	5.000	2.000	0.000	2.000	3.000	26.000	6.000	0.000	0.000
7	0.000	11.000	0.000	1.000	2.000	1.000	138.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [99]:

```
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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].ilc
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1157 0.0287 0.016  0.7132 0.0274 0.026
0.0612 0.0061 0.0058]]
Actual Class : 4
--------------------------------------------------
86 Text feature [suppressor] present in test data point [True]
145 Text feature [screening] present in test data point [True]
271 Text feature [determine] present in test data point [True]
272 Text feature [ca] present in test data point [True]
274 Text feature [bind] present in test data point [True]
275 Text feature [formed] present in test data point [True]
276 Text feature [high] present in test data point [True]
377 Text feature [caused] present in test data point [True]
379 Text feature [suggesting] present in test data point [True]
380 Text feature [region] present in test data point [True]
381 Text feature [kinases] present in test data point [True]
382 Text feature [mammalian] present in test data point [True]
383 Text feature [taken] present in test data point [True]
384 Text feature [substrate] present in test data point [True]
385 Text feature [endogenous] present in test data point [True]
389 Text feature [due] present in test data point [True]
390 Text feature [fold] present in test data point [True]
391 Text feature [carcinoma] present in test data point [True]
394 Text feature [endometrial] present in test data point [True]
395 Text feature [see] present in test data point [True]
397 Text feature [would] present in test data point [True]
400 Text feature [dominant] present in test data point [True]
401 Text feature [missense] present in test data point [True]
402 Text feature [values] present in test data point [True]
403 Text feature [position] present in test data point [True]
405 Text feature [2013] present in test data point [True]
406 Text feature [stimulation] present in test data point [True]
407 Text feature [normalized] present in test data point [True]
410 Text feature [washed] present in test data point [True]
411 Text feature [box] present in test data point [True]
412 Text feature [transfection] present in test data point [True]
413 Text feature [000] present in test data point [True]
414 Text feature [cannot] present in test data point [True]
415 Text feature [mm] present in test data point [True]
417 Text feature [2004] present in test data point [True]
419 Text feature [1998] present in test data point [True]
421 Text feature [deficient] present in test data point [True]
422 Text feature [lack] present in test data point [True]
```

```
423 Text feature [ability] present in test data point [True]
Out of the top  500  features  39 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [100]:

```
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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].ilc
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2975 0.0989 0.0081 0.5188 0.0312 0.008
0.0316 0.0031 0.003 ]]
Actual Class : 1
--------------------------------------------------
86 Text feature [suppressor] present in test data point [True]
146 Text feature [value] present in test data point [True]
270 Text feature [colorectal] present in test data point [True]
271 Text feature [determine] present in test data point [True]
272 Text feature [ca] present in test data point [True]
273 Text feature [hif] present in test data point [True]
274 Text feature [bind] present in test data point [True]
275 Text feature [formed] present in test data point [True]
276 Text feature [high] present in test data point [True]
377 Text feature [caused] present in test data point [True]
378 Text feature [tumorigenesis] present in test data point [True]
379 Text feature [suggesting] present in test data point [True]
380 Text feature [region] present in test data point [True]
382 Text feature [mammalian] present in test data point [True]
383 Text feature [taken] present in test data point [True]
384 Text feature [substrate] present in test data point [True]
385 Text feature [endogenous] present in test data point [True]
386 Text feature [damage] present in test data point [True]
387 Text feature [consequences] present in test data point [True]
389 Text feature [due] present in test data point [True]
390 Text feature [fold] present in test data point [True]
391 Text feature [carcinoma] present in test data point [True]
393 Text feature [unable] present in test data point [True]
394 Text feature [endometrial] present in test data point [True]
395 Text feature [see] present in test data point [True]
396 Text feature [52] present in test data point [True]
397 Text feature [would] present in test data point [True]
399 Text feature [hotspots] present in test data point [True]
400 Text feature [dominant] present in test data point [True]
401 Text feature [missense] present in test data point [True]
402 Text feature [values] present in test data point [True]
403 Text feature [position] present in test data point [True]
405 Text feature [2013] present in test data point [True]
407 Text feature [normalized] present in test data point [True]
408 Text feature [notch1] present in test data point [True]
409 Text feature [proportion] present in test data point [True]
410 Text feature [washed] present in test data point [True]
411 Text feature [box] present in test data point [True]
412 Text feature [transfection] present in test data point [True]
413 Text feature [000] present in test data point [True]
414 Text feature [cannot] present in test data point [True]
415 Text feature [mm] present in test data point [True]
422 Text feature [lack] present in test data point [True]
```

```
423 Text feature [ability] present in test data point [True]
Out of the top  500  features  44 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [101]:

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

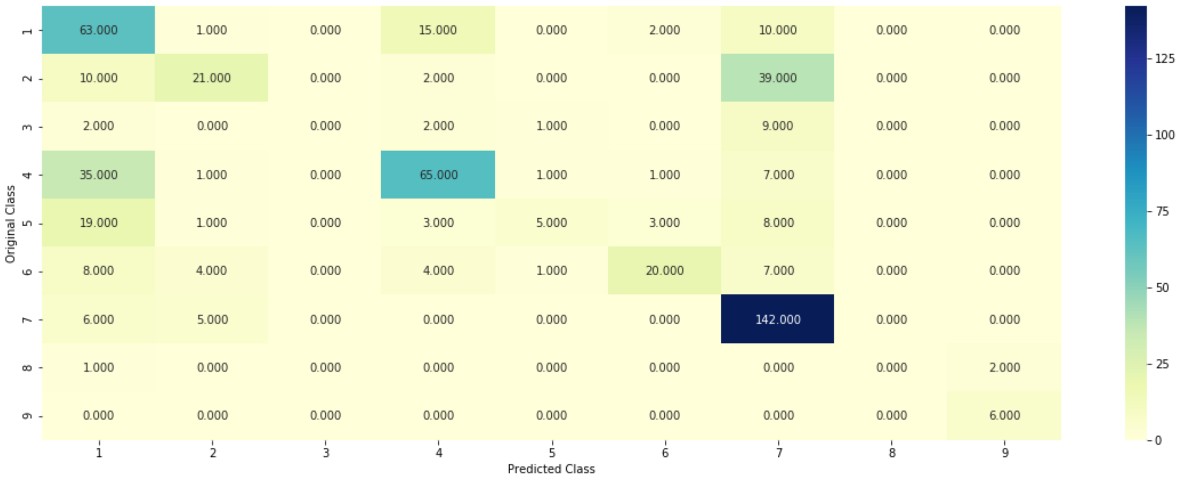
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1925444364520608
for n_estimators = 100 and max depth =  10
Log Loss : 1.2026193819502797
for n_estimators = 200 and max depth =  5
Log Loss : 1.1712440430370483
for n_estimators = 200 and max depth =  10
Log Loss : 1.1919751736720667
for n_estimators = 500 and max depth =  5
Log Loss : 1.1685213021811034
for n_estimators = 500 and max depth =  10
Log Loss : 1.1882228816403646
for n_estimators = 1000 and max depth =  5
Log Loss : 1.161564517808537
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1899310770903733
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1602611471154933
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1878498333683791
For values of best estimator =  2000 The train log loss is: 0.86553357645538
23
For values of best estimator =  2000 The cross validation log loss is: 1.160
2611471154933
For values of best estimator =  2000 The test log loss is: 1.182313085953124
1
```

```
<h3>4.5.2. Testing model with best hyper parameters (One Hot Encoding)</h3>
```

In [102]:

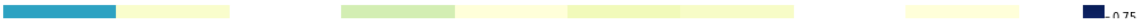
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=Non
# min_impurity_split=None, bootstrap=True, oob_score=False, 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 : 1.1602611471154933
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```

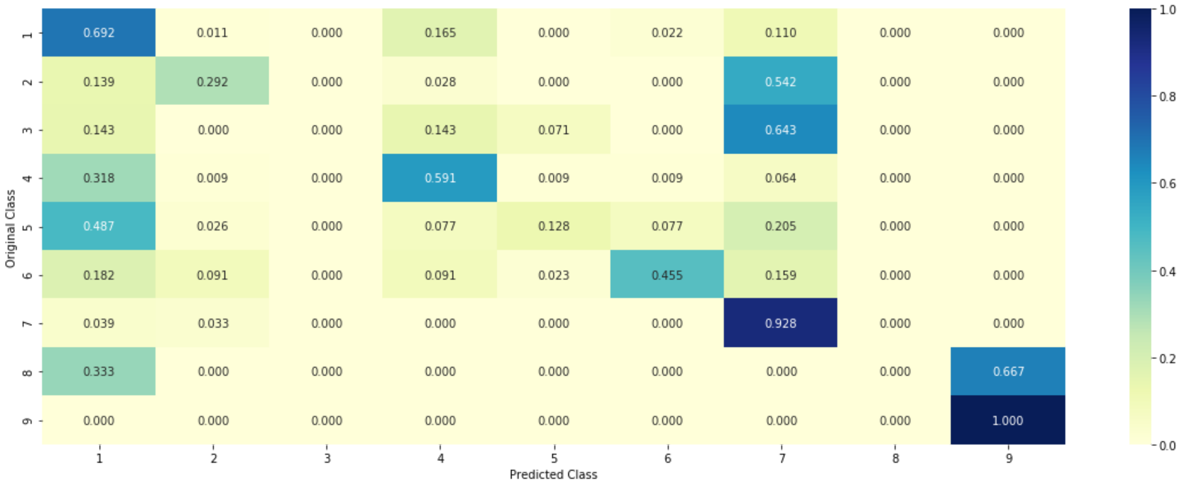


	1	2	3	4	5	6	7	8	9
1	63.000	1.000	0.000	15.000	0.000	2.000	10.000	0.000	0.000
2	10.000	21.000	0.000	2.000	0.000	0.000	39.000	0.000	0.000
3	2.000	0.000	0.000	2.000	1.000	0.000	9.000	0.000	0.000
4	35.000	1.000	0.000	65.000	1.000	1.000	7.000	0.000	0.000
5	19.000	1.000	0.000	3.000	5.000	3.000	8.000	0.000	0.000
6	8.000	4.000	0.000	4.000	1.000	20.000	7.000	0.000	0.000
7	6.000	5.000	0.000	0.000	0.000	0.000	142.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [103]:

```
# 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 : 4
Predicted Class Probabilities: [[0.1626 0.0439 0.0185 0.5363 0.0463 0.0401
0.1333 0.0094 0.0097]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [suppressor] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
8 Text feature [activation] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [variants] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [treatment] present in test data point [True]
17 Text feature [protein] present in test data point [True]
18 Text feature [deleterious] present in test data point [True]
21 Text feature [stability] present in test data point [True]
22 Text feature [pten] present in test data point [True]
23 Text feature [expression] present in test data point [True]
24 Text feature [functional] present in test data point [True]
26 Text feature [kinases] present in test data point [True]
30 Text feature [growth] present in test data point [True]
32 Text feature [therapy] present in test data point [True]
33 Text feature [months] present in test data point [True]
34 Text feature [constitutively] present in test data point [True]
36 Text feature [cells] present in test data point [True]
37 Text feature [patients] present in test data point [True]
41 Text feature [functions] present in test data point [True]
42 Text feature [defective] present in test data point [True]
43 Text feature [cell] present in test data point [True]
44 Text feature [yeast] present in test data point [True]
47 Text feature [activate] present in test data point [True]
48 Text feature [signaling] present in test data point [True]
51 Text feature [receptor] present in test data point [True]
52 Text feature [extracellular] present in test data point [True]
53 Text feature [proteins] present in test data point [True]
```

```
54 Text feature [nuclear] present in test data point [True]
55 Text feature [therapeutic] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
57 Text feature [inhibited] present in test data point [True]
58 Text feature [drug] present in test data point [True]
61 Text feature [phospho] present in test data point [True]
62 Text feature [downstream] present in test data point [True]
64 Text feature [ovarian] present in test data point [True]
65 Text feature [treated] present in test data point [True]
70 Text feature [predicted] present in test data point [True]
71 Text feature [variant] present in test data point [True]
73 Text feature [clinical] present in test data point [True]
74 Text feature [oncogene] present in test data point [True]
76 Text feature [response] present in test data point [True]
77 Text feature [null] present in test data point [True]
78 Text feature [resistance] present in test data point [True]
81 Text feature [ring] present in test data point [True]
83 Text feature [splice] present in test data point [True]
86 Text feature [combined] present in test data point [True]
87 Text feature [dna] present in test data point [True]
88 Text feature [p53] present in test data point [True]
89 Text feature [assays] present in test data point [True]
90 Text feature [atp] present in test data point [True]
91 Text feature [survival] present in test data point [True]
93 Text feature [type] present in test data point [True]
94 Text feature [mammalian] present in test data point [True]
95 Text feature [expressing] present in test data point [True]
96 Text feature [serum] present in test data point [True]
98 Text feature [conserved] present in test data point [True]
99 Text feature [harboring] present in test data point [True]
Out of the top  100  features  67 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [105]:

```python
test_point_index = 10
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 : 4
Predicted Class Probabilities: [[0.3409 0.0212 0.0336 0.3486 0.1142 0.0705
0.0548 0.0062 0.0101]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [suppressor] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
8 Text feature [activation] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [variants] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [treatment] present in test data point [True]
16 Text feature [pathogenic] present in test data point [True]
17 Text feature [protein] present in test data point [True]
18 Text feature [deleterious] present in test data point [True]
19 Text feature [brca1] present in test data point [True]
21 Text feature [stability] present in test data point [True]
23 Text feature [expression] present in test data point [True]
24 Text feature [functional] present in test data point [True]
26 Text feature [kinases] present in test data point [True]
27 Text feature [classified] present in test data point [True]
28 Text feature [neutral] present in test data point [True]
30 Text feature [growth] present in test data point [True]
35 Text feature [brca2] present in test data point [True]
36 Text feature [cells] present in test data point [True]
37 Text feature [patients] present in test data point [True]
40 Text feature [repair] present in test data point [True]
41 Text feature [functions] present in test data point [True]
42 Text feature [defective] present in test data point [True]
43 Text feature [cell] present in test data point [True]
44 Text feature [yeast] present in test data point [True]
45 Text feature [57] present in test data point [True]
46 Text feature [brct] present in test data point [True]
47 Text feature [activate] present in test data point [True]
48 Text feature [signaling] present in test data point [True]
51 Text feature [receptor] present in test data point [True]
53 Text feature [proteins] present in test data point [True]
54 Text feature [nuclear] present in test data point [True]
55 Text feature [therapeutic] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
57 Text feature [inhibited] present in test data point [True]
```

```
58 Text feature [drug] present in test data point [True]
59 Text feature [brca] present in test data point [True]
61 Text feature [phospho] present in test data point [True]
62 Text feature [downstream] present in test data point [True]
64 Text feature [ovarian] present in test data point [True]
65 Text feature [treated] present in test data point [True]
66 Text feature [f3] present in test data point [True]
67 Text feature [inactivation] present in test data point [True]
70 Text feature [predicted] present in test data point [True]
71 Text feature [variant] present in test data point [True]
72 Text feature [ic50] present in test data point [True]
73 Text feature [clinical] present in test data point [True]
75 Text feature [damage] present in test data point [True]
76 Text feature [response] present in test data point [True]
77 Text feature [null] present in test data point [True]
78 Text feature [resistance] present in test data point [True]
81 Text feature [ring] present in test data point [True]
82 Text feature [vus] present in test data point [True]
83 Text feature [splice] present in test data point [True]
84 Text feature [history] present in test data point [True]
86 Text feature [combined] present in test data point [True]
87 Text feature [dna] present in test data point [True]
88 Text feature [p53] present in test data point [True]
89 Text feature [assays] present in test data point [True]
91 Text feature [survival] present in test data point [True]
93 Text feature [type] present in test data point [True]
94 Text feature [mammalian] present in test data point [True]
95 Text feature [expressing] present in test data point [True]
98 Text feature [conserved] present in test data point [True]
99 Text feature [harboring] present in test data point [True]
Out of the top  100  features  73 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [106]:

```
# --------------------------------
# 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_c
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 : 1.9323176465474496
for n_estimators = 10 and max depth =  3
Log Loss : 1.6781556201998327
for n_estimators = 10 and max depth =  5
Log Loss : 1.4125539251954122
for n_estimators = 10 and max depth =  10
Log Loss : 1.8379554230380377
for n_estimators = 50 and max depth =  2
Log Loss : 1.641626286148054
for n_estimators = 50 and max depth =  3
Log Loss : 1.4113421484895443
for n_estimators = 50 and max depth =  5
Log Loss : 1.297575561679436
for n_estimators = 50 and max depth =  10
Log Loss : 1.7233244053060508
for n_estimators = 100 and max depth =  2
Log Loss : 1.5305785802709828
for n_estimators = 100 and max depth =  3
Log Loss : 1.454201598311224
for n_estimators = 100 and max depth =  5
Log Loss : 1.28381044531529
for n_estimators = 100 and max depth =  10
Log Loss : 1.7124458955252593
for n_estimators = 200 and max depth =  2
Log Loss : 1.596516459984317
for n_estimators = 200 and max depth =  3
Log Loss : 1.4738364913126907
for n_estimators = 200 and max depth =  5
Log Loss : 1.3501930625276162
for n_estimators = 200 and max depth =  10
Log Loss : 1.7004980924389954
for n_estimators = 500 and max depth =  2
Log Loss : 1.6136850935827143
for n_estimators = 500 and max depth =  3
Log Loss : 1.5165888289823763
for n_estimators = 500 and max depth =  5
Log Loss : 1.3651682436511272
for n_estimators = 500 and max depth =  10
Log Loss : 1.720534011758849
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6013502447604868
for n_estimators = 1000 and max depth =  3
Log Loss : 1.520348458340779
for n_estimators = 1000 and max depth =  5
```

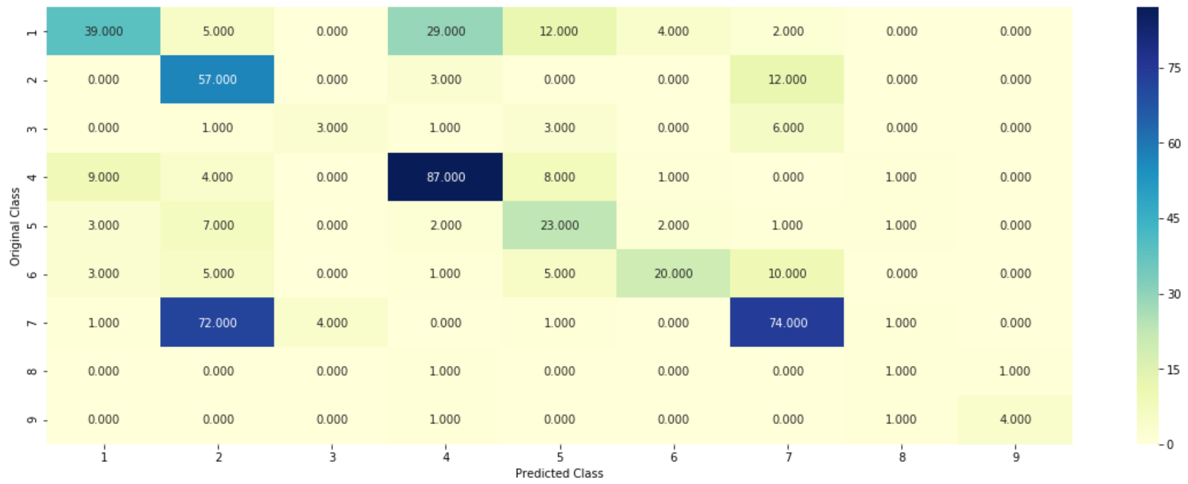
```
Log Loss : 1.3576546966082363
for n_estimators = 1000 and max depth =  10
Log Loss : 1.695849824948636
For values of best alpha =  100 The train log loss is: 0.06595209736587128
For values of best alpha =  100 The cross validation log loss is: 1.28381044
531529
For values of best alpha =  100 The test log loss is: 1.2739586167684207
```

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

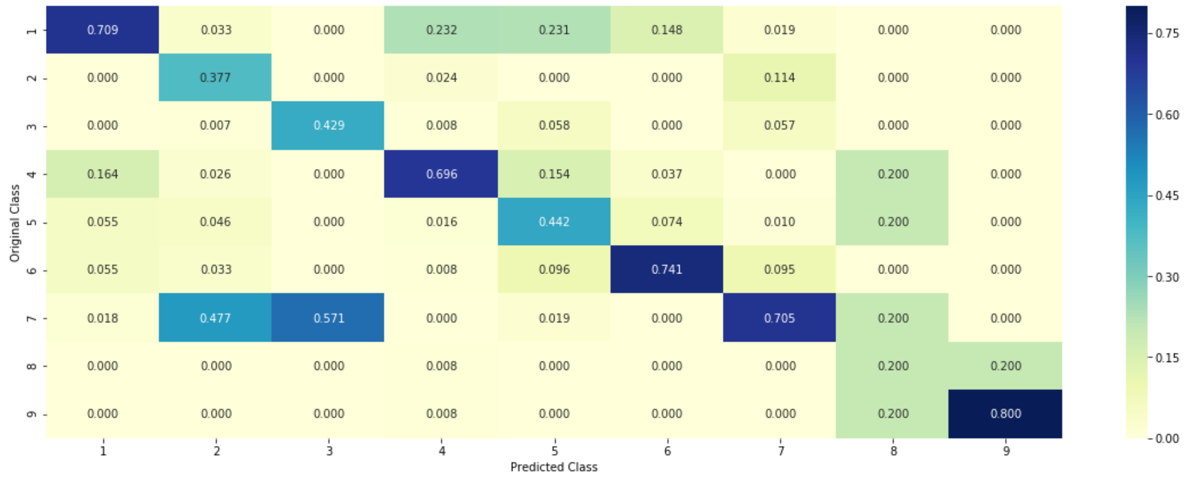
In [107]:

```
# --------------------------------
# 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,
```

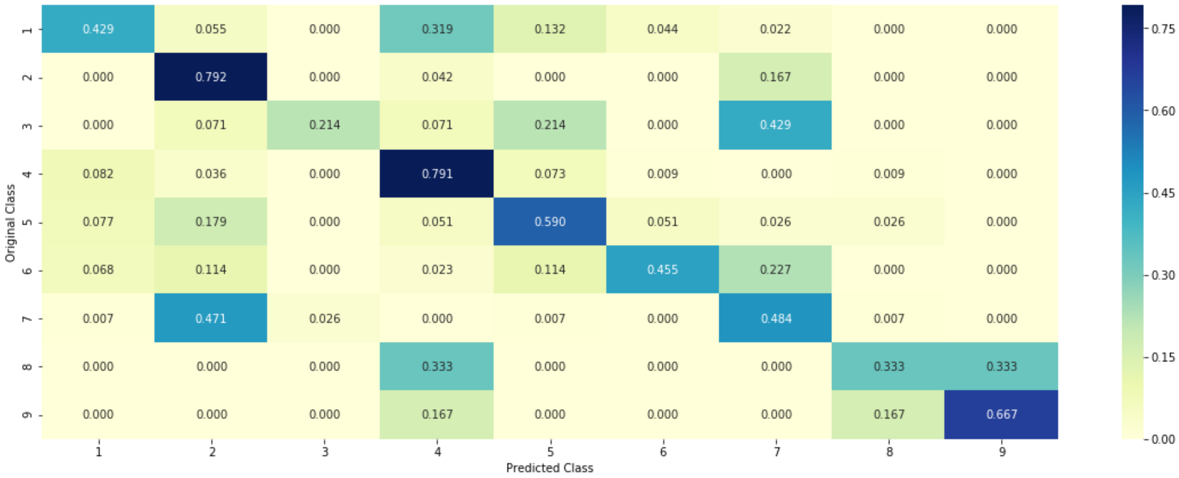
```
Log loss : 1.28381044531529
Number of mis-classified points : 0.42105263157894735
-------------------- Confusion matrix --------------------
```



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



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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.429	0.055	0.000	0.319	0.132	0.044	0.022	0.000	0.000
2	0.000	0.792	0.000	0.042	0.000	0.000	0.167	0.000	0.000
3	0.000	0.071	0.214	0.071	0.214	0.000	0.429	0.000	0.000
4	0.082	0.036	0.000	0.791	0.073	0.009	0.000	0.009	0.000
5	0.077	0.179	0.000	0.051	0.590	0.051	0.026	0.026	0.000
6	0.068	0.114	0.000	0.023	0.114	0.455	0.227	0.000	0.000
7	0.007	0.471	0.026	0.000	0.007	0.000	0.484	0.007	0.000
8	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.333	0.333
9	0.000	0.000	0.000	0.167	0.000	0.000	0.000	0.167	0.667

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [108]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1868 0.0259 0.1223 0.5143 0.0293 0.0531
0.0105 0.0288 0.029 ]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
```

4.5.5.2. Incorrectly Classified point

In [109]:

```
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 : 4
Predicted Class Probabilities: [[0.2496 0.0286 0.0892 0.4832 0.0311 0.0529
0.0105 0.0285 0.0262]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [110]:

```
r() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.h
--

penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
ilon=0.1, n_jobs=1, random_state=None, learning_rate=’optimal’, eta0=0.0, power_t=0.5,
t=False, average=False, n_iter=None)


ept_init, …]) Fit linear model with Stochastic Gradient Descent.
labels for samples in X.

--
iedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
-



or machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklear
----

ree=3, gamma=’auto’, coef0=0.0, shrinking=True, probability=False, tol=0.001,
=None, verbose=False, max_iter=-1, decision_function_shape=’ovr’, random_state=None)


  Fit the SVM model according to the given training data.
fication on samples in X.
----
iedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
----


or machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklear
----

tClassifier(n_estimators=10, criterion=’gini’, max_depth=None, min_samples_split=2,
nt_fraction_leaf=0.0, max_features=’auto’, max_leaf_nodes=None, min_impurity_decrease=0.0,
tstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,


estClassifier()
  Fit the SVM model according to the given training data.
fication on samples in X.
assification on samples in X.

mForestClassifier()
 of shape = [n_features]
 higher, the more important the feature).

----
iedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction
----


01, penalty='l2', loss='log', class_weight='balanced', random_state=0)
 train_y)
rCV(clf1, method="sigmoid")

penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
 train_y)
rCV(clf2, method="sigmoid")
```

```
01)
 train_y)
rCV(clf3, method="sigmoid")

ing, train_y)
Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
ing, train_y)
 : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
ing, train_y)
 %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))

,1,10]


i)
classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
ing, train_y)
: for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_o
sclf.predict_proba(cv_x_onehotCoding))
```

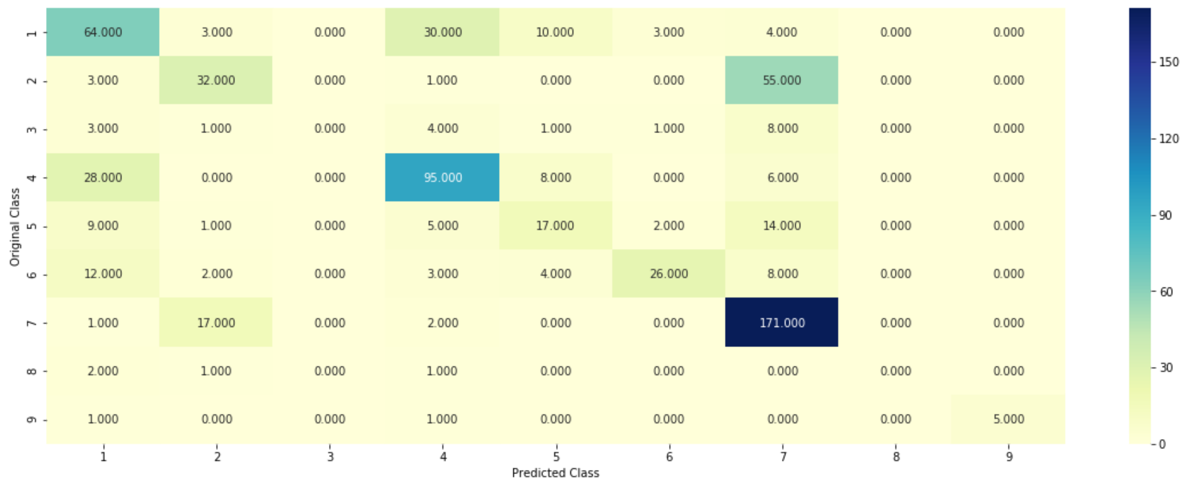
```
Logistic Regression :  Log Loss: 1.00
Support vector machines : Log Loss: 1.84
Naive Bayes : Log Loss: 1.18
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.710
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.307
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.241
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.621
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 2.048
```

4.7.2 testing the model with the best hyper parameters

In [111]:

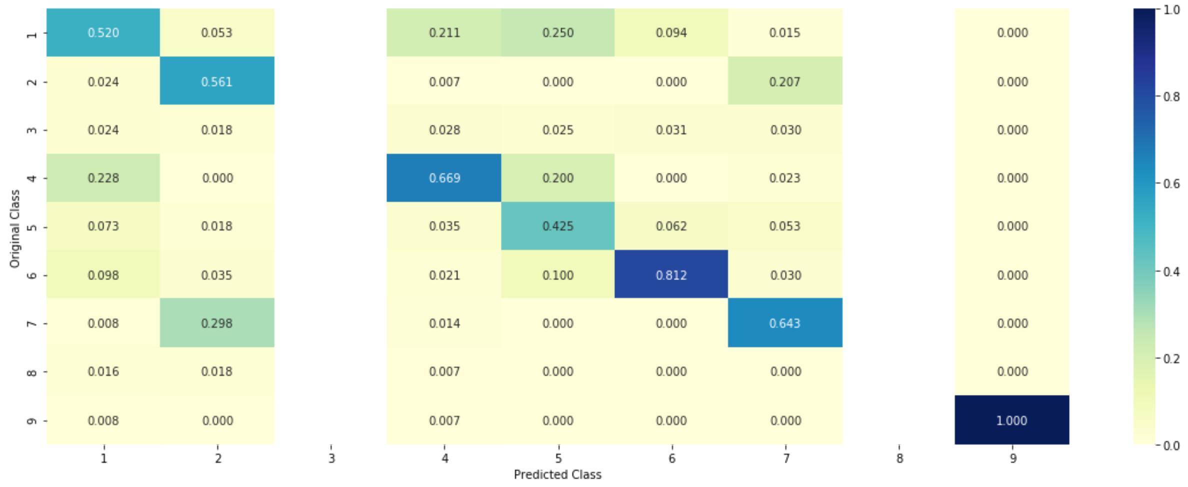
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, u
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.34561813798628266
Log loss (CV) on the stacking classifier : 1.2407645456242287
Log loss (test) on the stacking classifier : 1.2343230082514793
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```



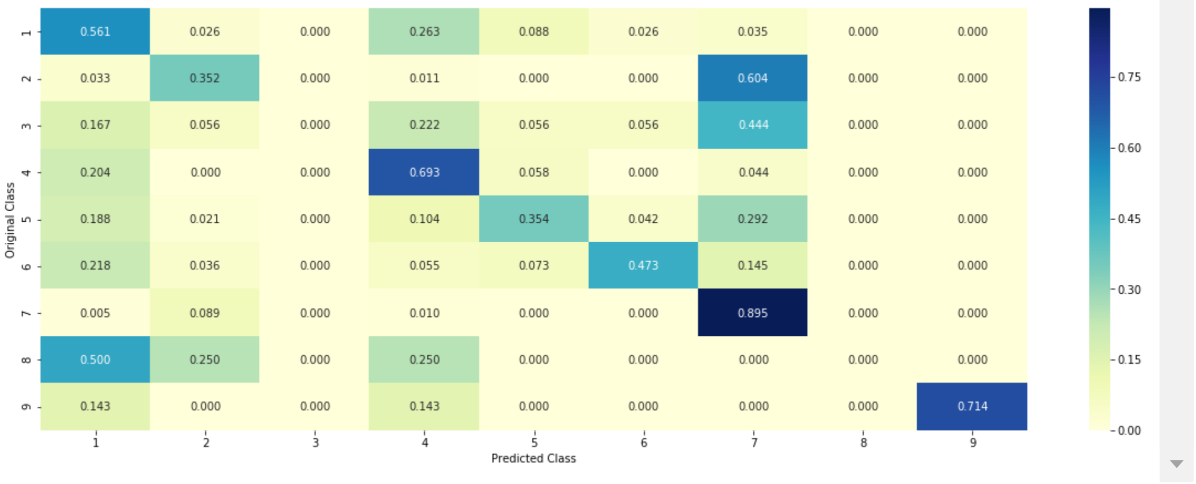
	1	2	3	4	5	6	7	8	9
1	64.000	3.000	0.000	30.000	10.000	3.000	4.000	0.000	0.000
2	3.000	32.000	0.000	1.000	0.000	0.000	55.000	0.000	0.000
3	3.000	1.000	0.000	4.000	1.000	1.000	8.000	0.000	0.000
4	28.000	0.000	0.000	95.000	8.000	0.000	6.000	0.000	0.000
5	9.000	1.000	0.000	5.000	17.000	2.000	14.000	0.000	0.000
6	12.000	2.000	0.000	3.000	4.000	26.000	8.000	0.000	0.000
7	1.000	17.000	0.000	2.000	0.000	0.000	171.000	0.000	0.000
8	2.000	1.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
9	1.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.520	0.053		0.211	0.250	0.094	0.015		0.000
2	0.024	0.561		0.007	0.000	0.000	0.207		0.000
3	0.024	0.018		0.028	0.025	0.031	0.030		0.000
4	0.228	0.000		0.669	0.200	0.000	0.023		0.000
5	0.073	0.018		0.035	0.425	0.062	0.053		0.000
6	0.098	0.035		0.021	0.100	0.812	0.030		0.000
7	0.008	0.298		0.014	0.000	0.000	0.643		0.000
8	0.016	0.018		0.007	0.000	0.000	0.000		0.000
9	0.008	0.000		0.007	0.000	0.000	0.000		1.000

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



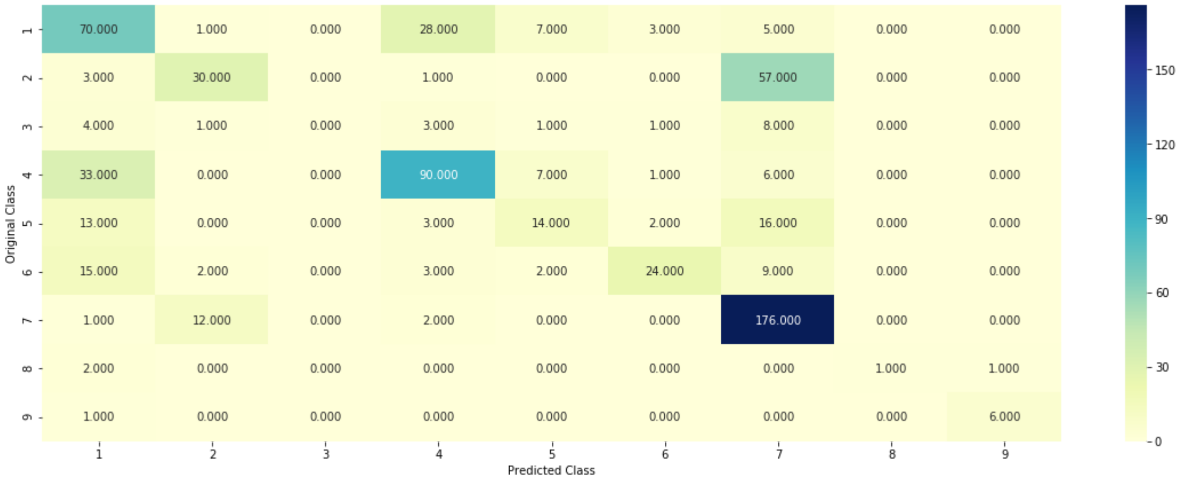
	1	2	3	4	5	6	7	8	9
1	0.561	0.026	0.000	0.263	0.088	0.026	0.035	0.000	0.000
2	0.033	0.352	0.000	0.011	0.000	0.000	0.604	0.000	0.000
3	0.167	0.056	0.000	0.222	0.056	0.056	0.444	0.000	0.000
4	0.204	0.000	0.000	0.693	0.058	0.000	0.044	0.000	0.000
5	0.188	0.021	0.000	0.104	0.354	0.042	0.292	0.000	0.000
6	0.218	0.036	0.000	0.055	0.073	0.473	0.145	0.000	0.000
7	0.005	0.089	0.000	0.010	0.000	0.000	0.895	0.000	0.000
8	0.500	0.250	0.000	0.250	0.000	0.000	0.000	0.000	0.000
9	0.143	0.000	0.000	0.143	0.000	0.000	0.000	0.000	0.714

4.7.3 Maximum Voting classifier

In [112]:

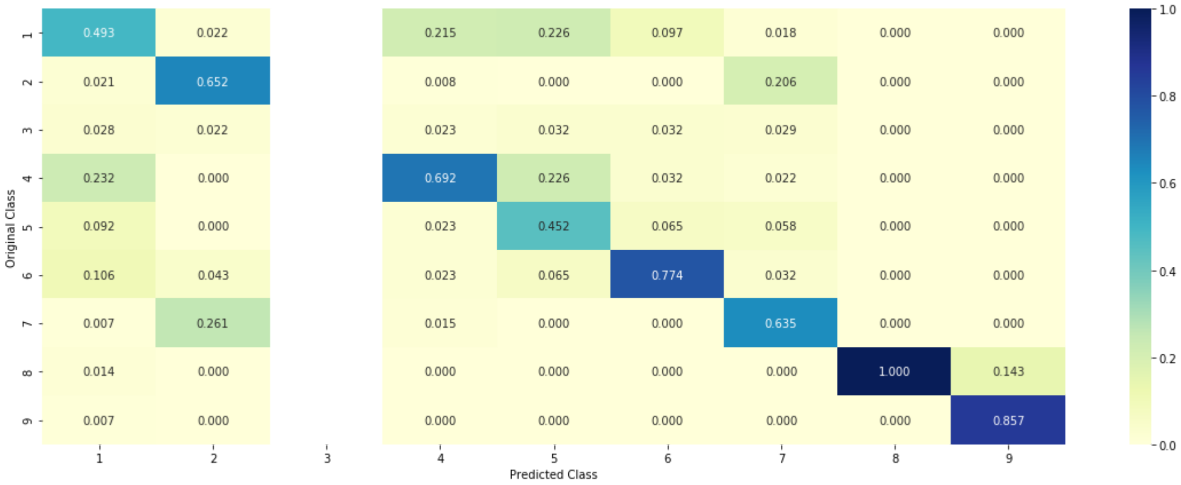
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.h
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)],
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on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

```
Log loss (train) on the VotingClassifier : 0.7943868924402231
Log loss (CV) on the VotingClassifier : 1.1721372280667581
Log loss (test) on the VotingClassifier : 1.1944763181067921
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```



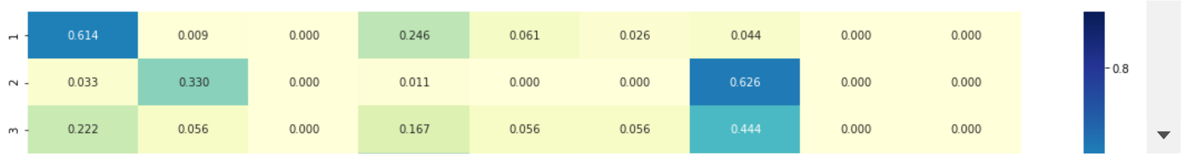
	1	2	3	4	5	6	7	8	9
1	70.000	1.000	0.000	28.000	7.000	3.000	5.000	0.000	0.000
2	3.000	30.000	0.000	1.000	0.000	0.000	57.000	0.000	0.000
3	4.000	1.000	0.000	3.000	1.000	1.000	8.000	0.000	0.000
4	33.000	0.000	0.000	90.000	7.000	1.000	6.000	0.000	0.000
5	13.000	0.000	0.000	3.000	14.000	2.000	16.000	0.000	0.000
6	15.000	2.000	0.000	3.000	2.000	24.000	9.000	0.000	0.000
7	1.000	12.000	0.000	2.000	0.000	0.000	176.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.493	0.022		0.215	0.226	0.097	0.018	0.000	0.000
2	0.021	0.652		0.008	0.000	0.000	0.206	0.000	0.000
3	0.028	0.022		0.023	0.032	0.032	0.029	0.000	0.000
4	0.232	0.000		0.692	0.226	0.032	0.022	0.000	0.000
5	0.092	0.000		0.023	0.452	0.065	0.058	0.000	0.000
6	0.106	0.043		0.023	0.065	0.774	0.032	0.000	0.000
7	0.007	0.261		0.015	0.000	0.000	0.635	0.000	0.000
8	0.014	0.000		0.000	0.000	0.000	0.000	1.000	0.143
9	0.007	0.000		0.000	0.000	0.000	0.000	0.000	0.857

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



5. Assignments

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

5.1 CountVectorization on Text Features, including both unigrams and bigrams

In [113]:

```
# 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_bow = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occu
fea_dict = dict(zip(list(train_features),train_fea_counts))

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

```
Total number of unique words in train data : 771041
```

In [114]:

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

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

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

In [115]:

```
print("Shape of bow encoded text features(train) is ",train_text_feature_bow.shape)
print("Shape of bow encoded text features(test) is ",test_text_feature_bow.shape)
print("Shape of bow encoded text features(cv) is ",cv_text_feature_bow.shape)
```

```
Shape of bow encoded text features(train) is  (2124, 771041)
Shape of bow encoded text features(test) is  (665, 771041)
Shape of bow encoded text features(cv) is  (532, 771041)
```

In [116]:

```
#stacking the features
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_bow)).tocsr(
train_y = np.array(list(train_df['Class']))

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

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

In [117]:

```
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, 773236)
(number of data points * number of features) in test data =  (665, 773236)
(number of data points * number of features) in cross validation data = (53
2, 773236)
```

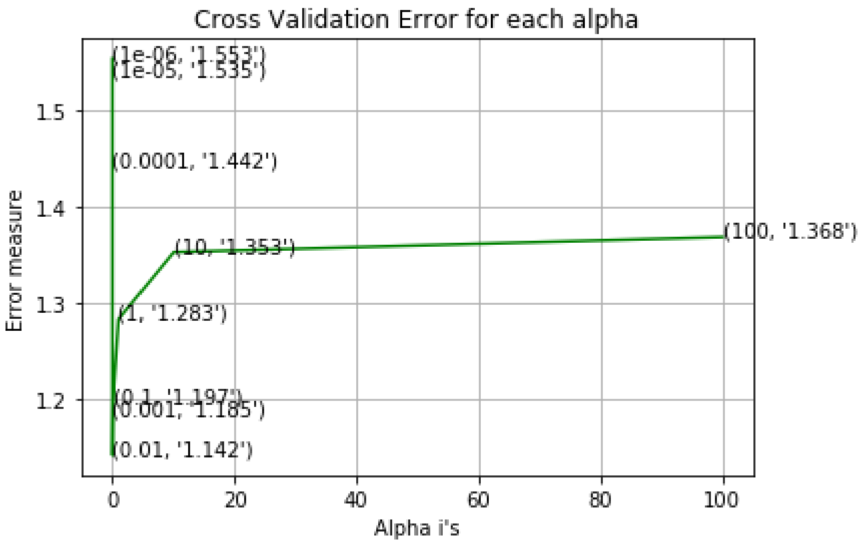
5.2. Logistic Regression with class balancing

Hyperparameter tuning

In [118]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.5534839502803033
for alpha = 1e-05
Log Loss : 1.5352894875202445
for alpha = 0.0001
Log Loss : 1.4419748000966093
for alpha = 0.001
Log Loss : 1.184621761644552
for alpha = 0.01
Log Loss : 1.1418982681276748
for alpha = 0.1
Log Loss : 1.196854834811157
for alpha = 1
Log Loss : 1.2830425320357504
for alpha = 10
Log Loss : 1.3526292867023395
for alpha = 100
Log Loss : 1.3683567180405047
```

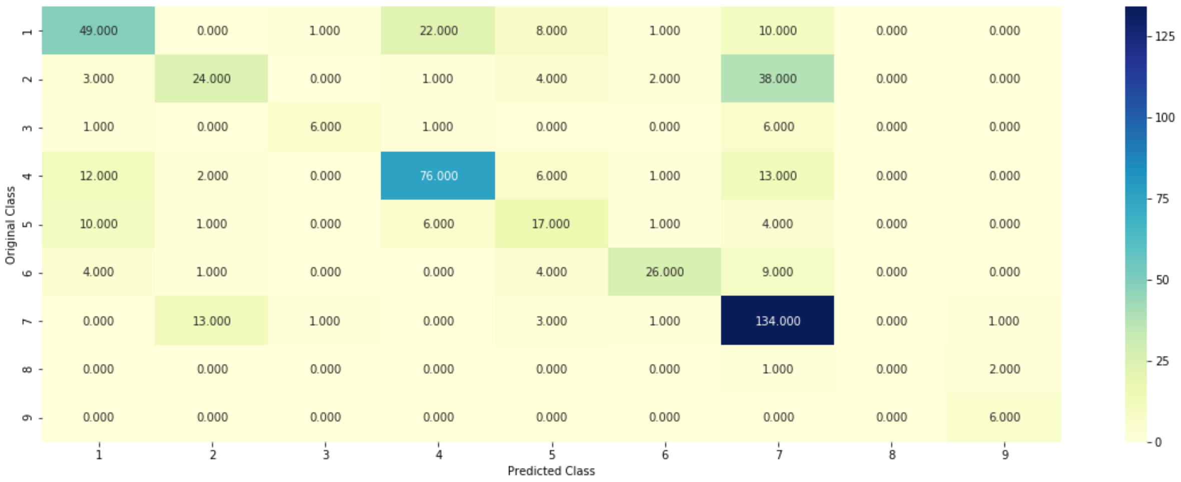


```
For values of best alpha =  0.01 The train log loss is: 0.6815998874490794
For values of best alpha =  0.01 The cross validation log loss is: 1.1418982
681276748
For values of best alpha =  0.01 The test log loss is: 1.0747719782343605
```

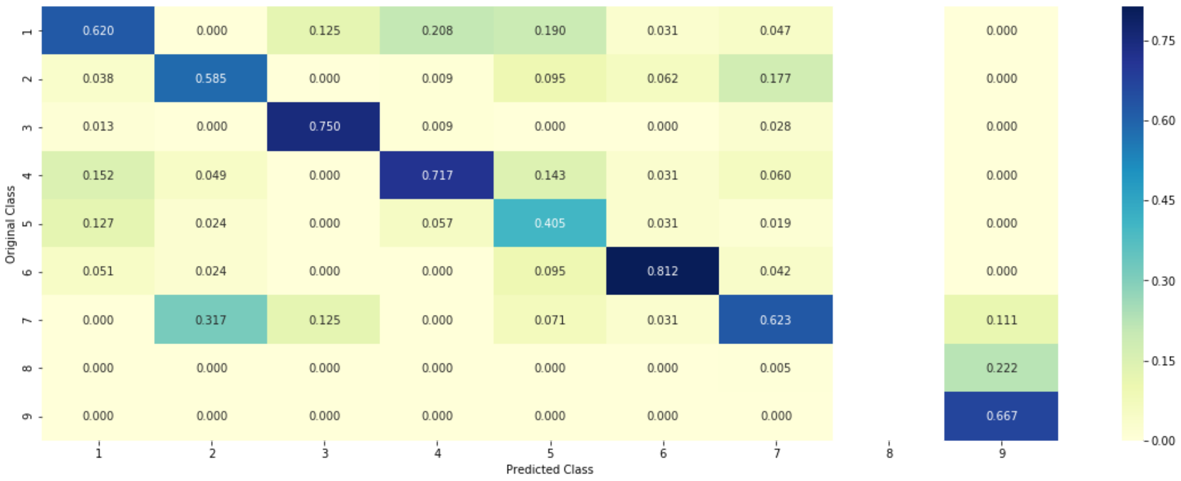
In [119]:

```
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
```

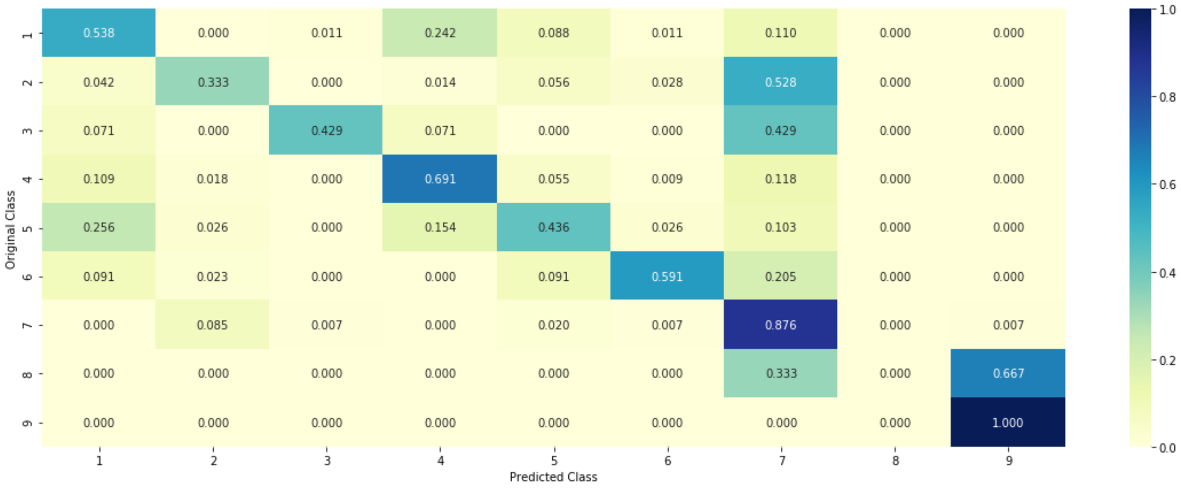
```
Log loss : 1.1418982681276748
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```



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



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



5.3 Feature engineering techniques to reduce the CV and test log-loss to a value less than 1.0

5.3.1 Using Tfidf vectorization(bigram) with Text features

In [120]:

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

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occu
fea_dict = dict(zip(list(train_features_tfidf),train_fea_counts))

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

```
Total number of unique words in train data : 717528
```

In [121]:

```
test_text_feature_tfidf = text_vectorizer.transform(test_df['TEXT'])
cv_text_feature_tfidf = text_vectorizer.transform(cv_df['TEXT'])
```

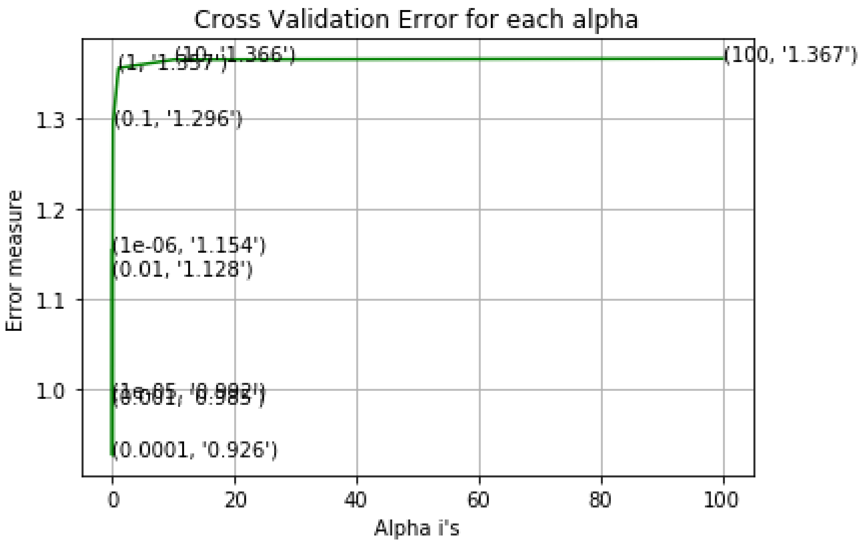
In [126]:

```
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_tfidf = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf)).tocsr()
train_y = np.array(list(train_df['Class']))

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

cv_x_tfidf = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [127]:

```
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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15)
    # to avoid rounding error while multiplying probabilites we use log-probability 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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1544254309164548
for alpha = 1e-05
Log Loss : 0.9916464621286171
for alpha = 0.0001
Log Loss : 0.9263323472562257
for alpha = 0.001
Log Loss : 0.9846788069087555
for alpha = 0.01
Log Loss : 1.1275904704493311
for alpha = 0.1
Log Loss : 1.295547419734905
for alpha = 1
Log Loss : 1.3568904993599318
for alpha = 10
Log Loss : 1.36622325237505
for alpha = 100
Log Loss : 1.3673948942707213
```

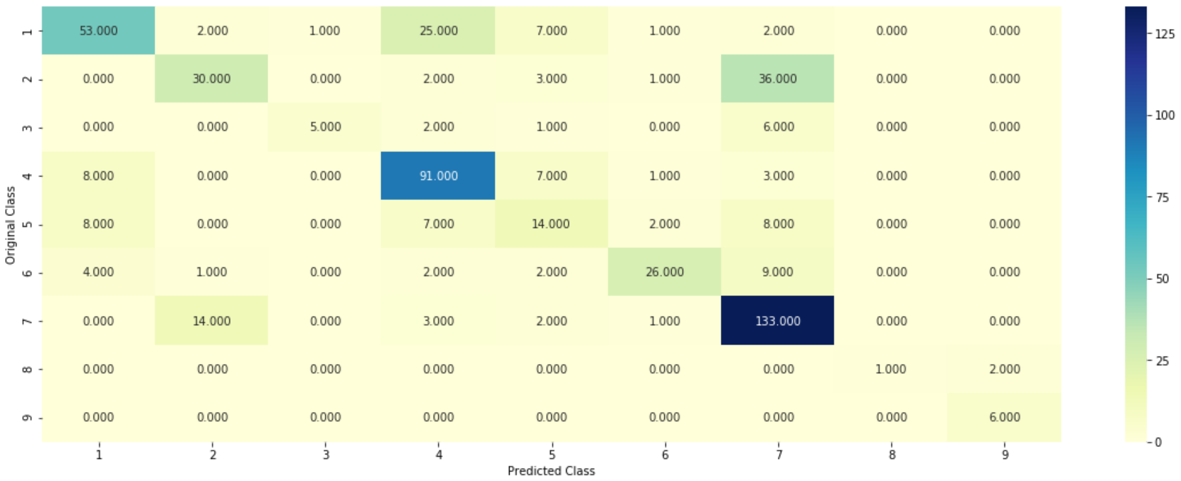


```
For values of best alpha =  0.0001 The train log loss is: 0.3609490113967570
6
For values of best alpha =  0.0001 The cross validation log loss is: 0.92633
23472562257
For values of best alpha =  0.0001 The test log loss is: 0.928954133892105
```

In [128]:

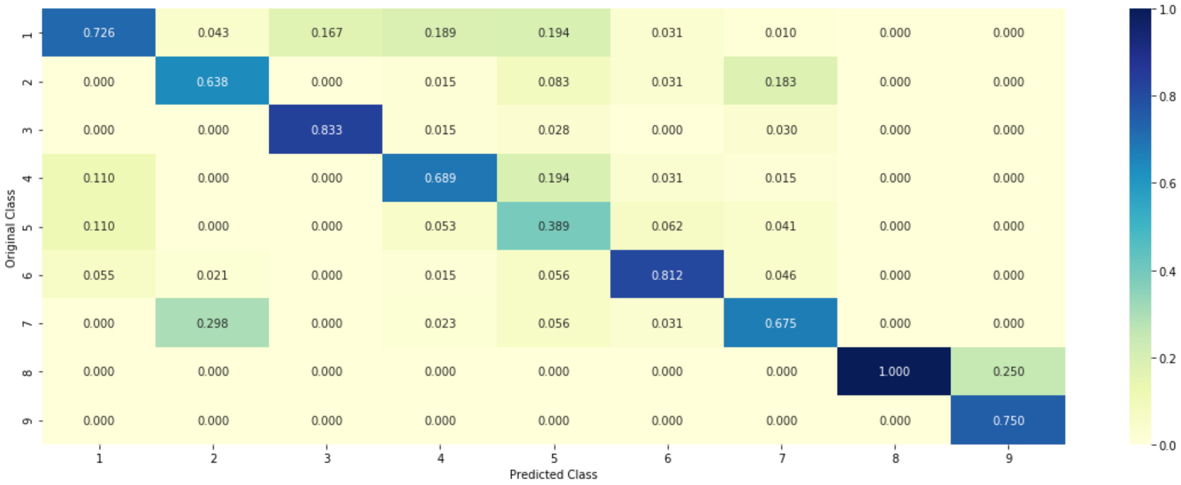
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

```
Log loss : 0.9263323472562257
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```



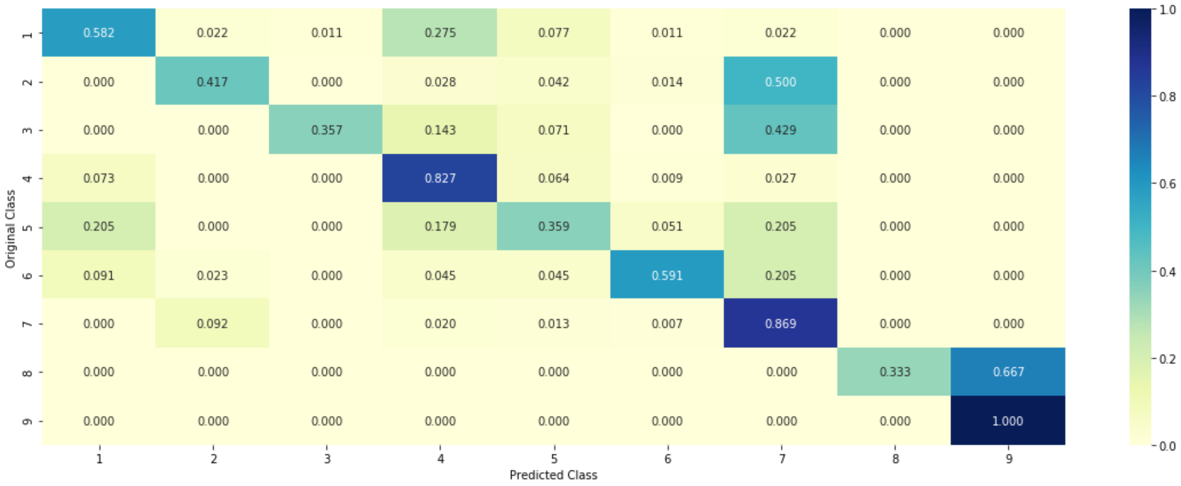
	1	2	3	4	5	6	7	8	9
1	53.000	2.000	1.000	25.000	7.000	1.000	2.000	0.000	0.000
2	0.000	30.000	0.000	2.000	3.000	1.000	36.000	0.000	0.000
3	0.000	0.000	5.000	2.000	1.000	0.000	6.000	0.000	0.000
4	8.000	0.000	0.000	91.000	7.000	1.000	3.000	0.000	0.000
5	8.000	0.000	0.000	7.000	14.000	2.000	8.000	0.000	0.000
6	4.000	1.000	0.000	2.000	2.000	26.000	9.000	0.000	0.000
7	0.000	14.000	0.000	3.000	2.000	1.000	133.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.726	0.043	0.167	0.189	0.194	0.031	0.010	0.000	0.000
2	0.000	0.638	0.000	0.015	0.083	0.031	0.183	0.000	0.000
3	0.000	0.000	0.833	0.015	0.028	0.000	0.030	0.000	0.000
4	0.110	0.000	0.000	0.689	0.194	0.031	0.015	0.000	0.000
5	0.110	0.000	0.000	0.053	0.389	0.062	0.041	0.000	0.000
6	0.055	0.021	0.000	0.015	0.056	0.812	0.046	0.000	0.000
7	0.000	0.298	0.000	0.023	0.056	0.031	0.675	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.250
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.750

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



	1	2	3	4	5	6	7	8	9
1	0.582	0.022	0.011	0.275	0.077	0.011	0.022	0.000	0.000
2	0.000	0.417	0.000	0.028	0.042	0.014	0.500	0.000	0.000
3	0.000	0.000	0.357	0.143	0.071	0.000	0.429	0.000	0.000
4	0.073	0.000	0.000	0.827	0.064	0.009	0.027	0.000	0.000
5	0.205	0.000	0.000	0.179	0.359	0.051	0.205	0.000	0.000
6	0.091	0.023	0.000	0.045	0.045	0.591	0.205	0.000	0.000
7	0.000	0.092	0.000	0.020	0.013	0.007	0.869	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

In [9]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Model", "Train Log Loss","CV Log Loss", "Test Log Loss","Mis
x.add_row(["-","Random_Model", "-",2.507, 2.509, "-"])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","Naive Bayes", 0.453,1.184, 1.182, 39.097])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["ResponseCoding","KNN", 0.669, 1.045, 1.057, 36.090])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","Logistic_Regression(Balanced)",0.408, 0.965, 1.004, 33.835])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","Logistic_Regression(ImBalanced)", 0.399, 0.997,1.026, 33.835])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","Linear_SVM", 0.334, 0.984, 1.037,32.706])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","Random_Forest", 0.865, 1.160,1.182, 39.473])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["ResponseCoding","Random_Forest", 0.066, 1.284,1.274, 42.105])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","StackingClassifier", 0.346, 1.241, 1.234, 38.346])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["OHE","Max_Voting", 0.794, 1.172, 1.194, 38.195])
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["BoW(unigram-bigram)","Logistic_Regression(Balanced)", 0.682, 1.142,1.075, 36.46
x.add_row(["------------------","-----------------------","---------------","-----------","
x.add_row(["Tfidf(bigram)","Logistic_Regression(Balanced)", 0.361, 0.926, 0.929, 32.519])

print(x)
```

```
+---------------------+---------------------------------+-----------------
+-------------+---------------+--------------------+
|      Vectorizer     |              Model              |  Train Log Loss 
| CV Log Loss | Test Log Loss | Misclassified %age |
+---------------------+---------------------------------+-----------------
+-------------+---------------+--------------------+
|          -          |           Random_Model          |        -        
|    2.507    |     2.509     |         -          |
|  ------------------ |     -----------------------     | --------------- 
| ----------- |  -----------  |    -----------     |
|         OHE         |           Naive Bayes           |      0.453      
|    1.184    |     1.182     |       39.097       |
|  ------------------ |     -----------------------     | --------------- 
| ----------- |  -----------  |    -----------     |
|    ResponseCoding   |               KNN               |      0.669      
|    1.045    |     1.057     |       36.09        |
|  ------------------ |     -----------------------     | --------------- 
| ----------- |  -----------  |    -----------     |
|         OHE         |  Logistic_Regression(Balanced)   |      0.408      
|    0.965    |     1.004     |       33.835       |
|  ------------------ |     -----------------------     | --------------- 
| ----------- |  -----------  |    -----------     |
|         OHE         | Logistic_Regression(ImBalanced) |      0.399      
|    0.997    |     1.026     |       33.835       |
|  ------------------ |     -----------------------     | --------------- 
| ----------- |  -----------  |    -----------     |
|         OHE         |            Linear_SVM           |      0.334      
|    0.984    |     1.037     |       32.706       |
|  ------------------ |     -----------------------     | ---------------
| ----------- |  -----------  |    -----------     |
|         OHE         |          Random_Forest           |      0.865
|     1.16    |     1.182     |        39.473      |
|  ------------------ |     -----------------------     | ---------------
| ----------- |  -----------  |    -----------     |
|    ResponseCoding   |          Random_Forest           |      0.066
|    1.284    |     1.274     |        42.105      |
|  ------------------ |     -----------------------     | ---------------
| ----------- |  -----------  |    -----------     |
|         OHE         |        StackingClassifier        |      0.346
|    1.241    |     1.234     |        38.346      |
|  ------------------ |     -----------------------     | ---------------
| ----------- |  -----------  |    -----------     |
|         OHE         |            Max_Voting            |      0.794
|    1.172    |     1.194     |        38.195      |
|  ------------------ |     -----------------------     | ---------------
| ----------- |  -----------  |    -----------     |
| BoW(unigram-bigram) |  Logistic_Regression(Balanced)  |      0.682
|    1.142    |     1.075     |        36.466      |
|  ------------------ |     -----------------------     | ---------------
| ----------- |  -----------  |    -----------     |
|    Tfidf(bigram)    |  Logistic_Regression(Balanced)  |      0.361
|    0.926    |     0.929     |        32.519      |
+---------------------+----------------------------------+-----------------
+-------------+---------------+--------------------+
```