Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25 (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 class
ified into => Multi class classification problem
```

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation (https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation)

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:
* Interpretability * Class probabilities are needed. * Penalize the errors in class probabilites => Metric is Log-loss. * No Latency constraints.

2.3. Train, CV and Test Datasets

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call d
rive.mount("/content/drive", force_remount=True).
```

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
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: Future
Warning: The sklearn.metrics.classification module is  deprecated in version 0.
22 and will be removed in version 0.24. The corresponding classes / functions s
hould instead be imported from sklearn.metrics. Anything that cannot be importe
d from sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarni
ng: The module is deprecated in version 0.21 and will be removed in version 0.2
3 since we've dropped support for Python 2.7. Please rely on the official versi
on of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: Future
Warning: The sklearn.neighbors.base module is  deprecated in version 0.22 and w
ill be removed in version 0.24. The corresponding classes / functions should in
stead be imported from sklearn.neighbors. Anything that cannot be imported from
sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [3]:
```
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/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[3]:

	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 [4]:
```
# note the seprator in this file
data_text =pd.read_csv("/content/drive/My Drive/Colab Notebooks/training_text",s
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[4]:

	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 [5]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[5]: True

In [0]:
```
# 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 [7]:
```
#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, "secon
```

```
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 : 26.294294999999998 seconds
```

In [8]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[8]:

	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 [9]:
```
result[result.isnull().any(axis=1)]
```

Out[9]:

	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 [0]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variati
```

In [11]:
```
result[result['ID']==1109]
```

Out[11]:

	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)

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

In [0]:
```
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 v
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_t
# split the train data into train and cross validation by maintaining same distri
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_tr
```

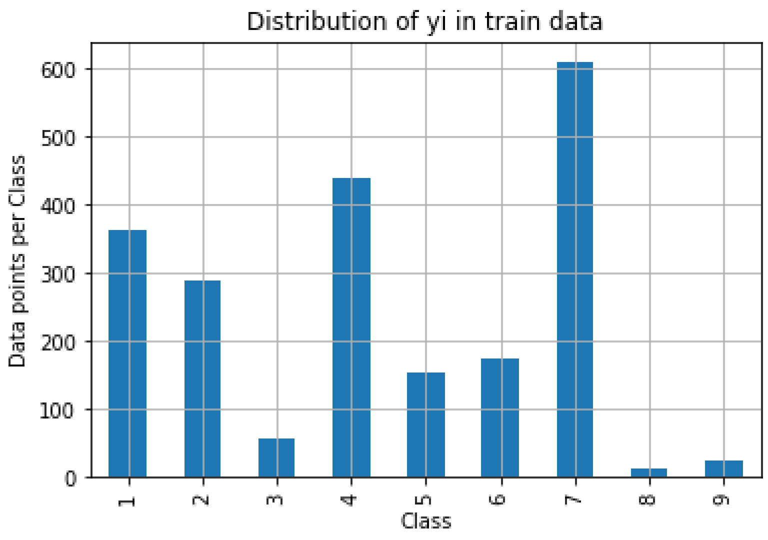
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [13]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

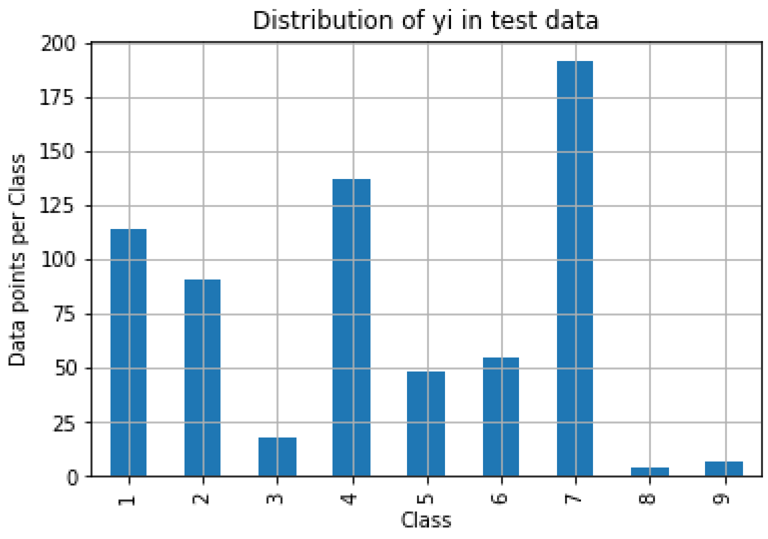
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

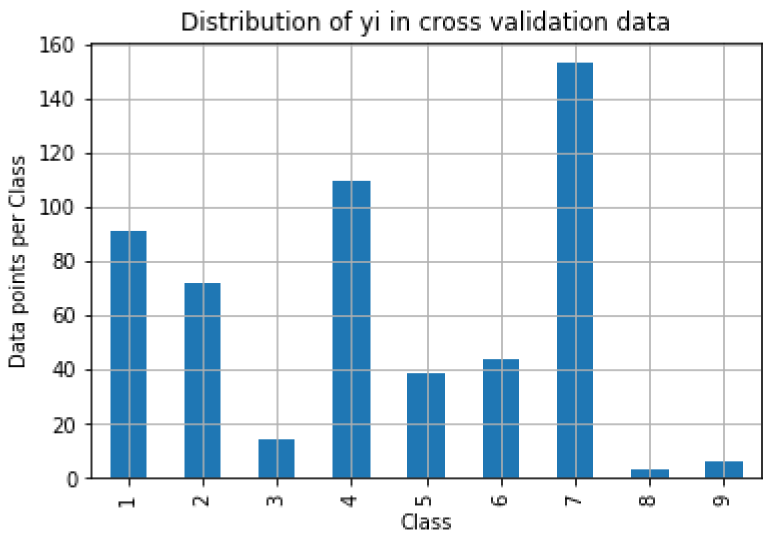
In [14]:
```
# it returns a dict, keys as class labels and values as the number of data points
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsor
# -(train_class_distribution.values): the minus sign will give us in decreasing 
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.va


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsor
# -(train_class_distribution.values): the minus sign will give us in decreasing 
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.val

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

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



```
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 %)
```

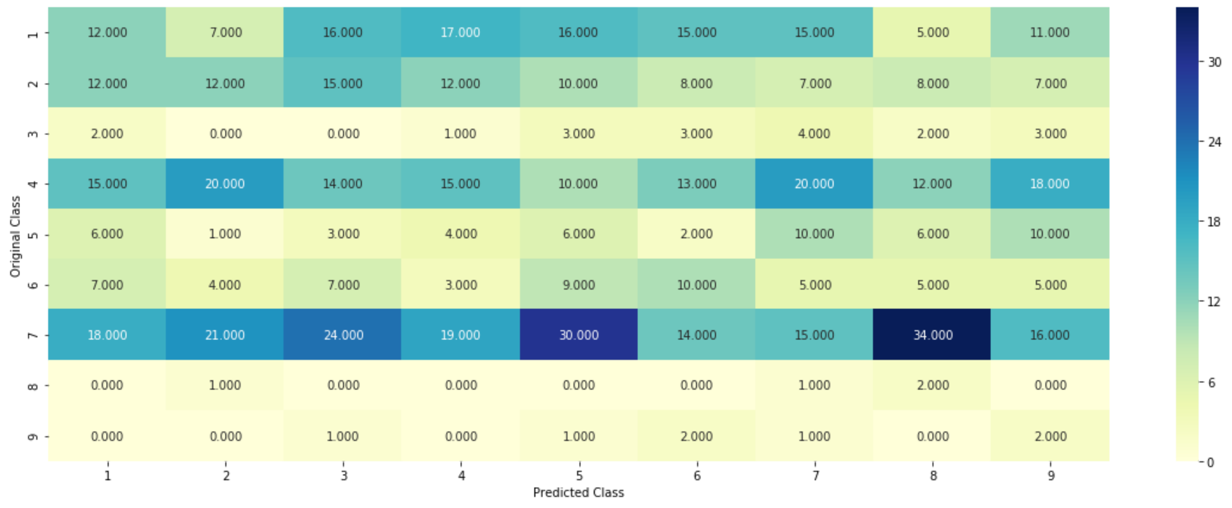
3.2 Prediction using a 'Random' Model

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

In [0]:
```
# 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

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

    # 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 rov
    # 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
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rov
    # 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, ytic
    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, ytic
    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, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

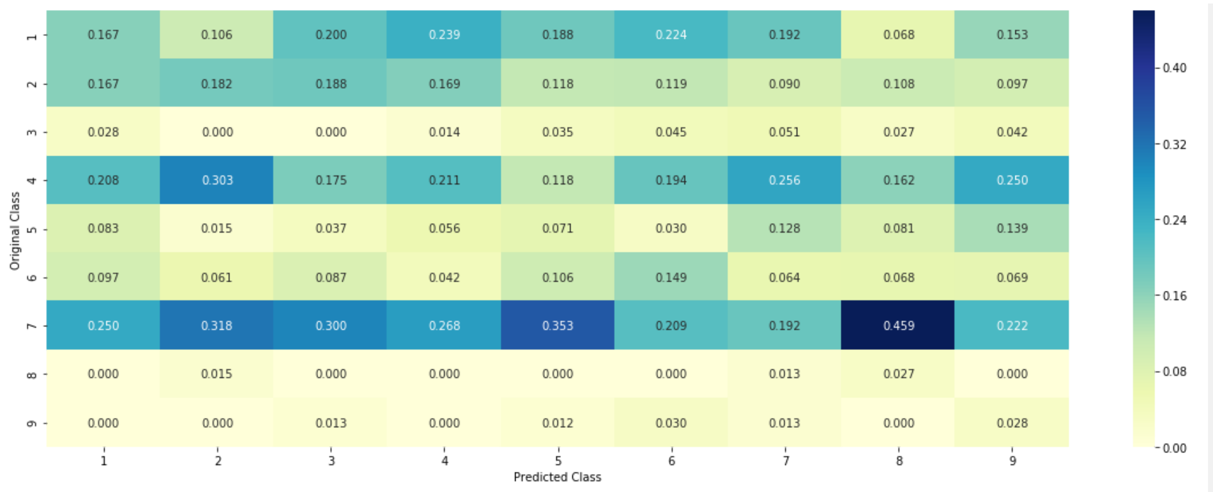
In [192]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their s
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


# 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

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 2.4266785057117146
Log loss on Test Data using Random Model 2.455515413264332
-------------------- Confusion matrix --------------------
```

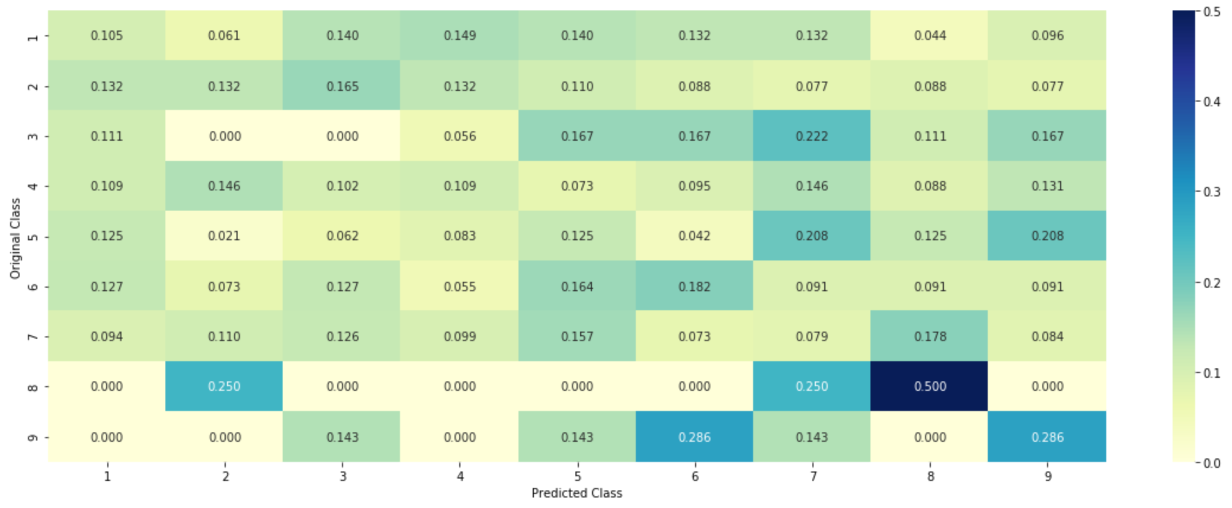


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	12.000	7.000	16.000	17.000	16.000	15.000	15.000	5.000	11.000
2	12.000	12.000	15.000	12.000	10.000	8.000	7.000	8.000	7.000
3	2.000	0.000	0.000	1.000	3.000	3.000	4.000	2.000	3.000
4	15.000	20.000	14.000	15.000	10.000	13.000	20.000	12.000	18.000
5	6.000	1.000	3.000	4.000	6.000	2.000	10.000	6.000	10.000
6	7.000	4.000	7.000	3.000	9.000	10.000	5.000	5.000	5.000
7	18.000	21.000	24.000	19.000	30.000	14.000	15.000	34.000	16.000
8	0.000	1.000	0.000	0.000	0.000	0.000	1.000	2.000	0.000
9	0.000	0.000	1.000	0.000	1.000	2.000	1.000	0.000	2.000

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



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



3.3 Univariate Analysis

In [0]:
```
# 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 tra
# build a vector (1*9) , the first element = (number of times it occured in class
# gv_dict is like a look up table, for every gene it store a (1*9) representation
# 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 eac
    gv_dict = dict()

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

            # cls_cnt.shape[0](numerator) will contain the number of time that p
            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, 0.068181818181818
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060606060606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918238993710
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920529
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333333
    #      ...
    #     }
    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 featu
    gv_fea = []
    # for every feature values in the given data frame we will check if it is th
    # 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?

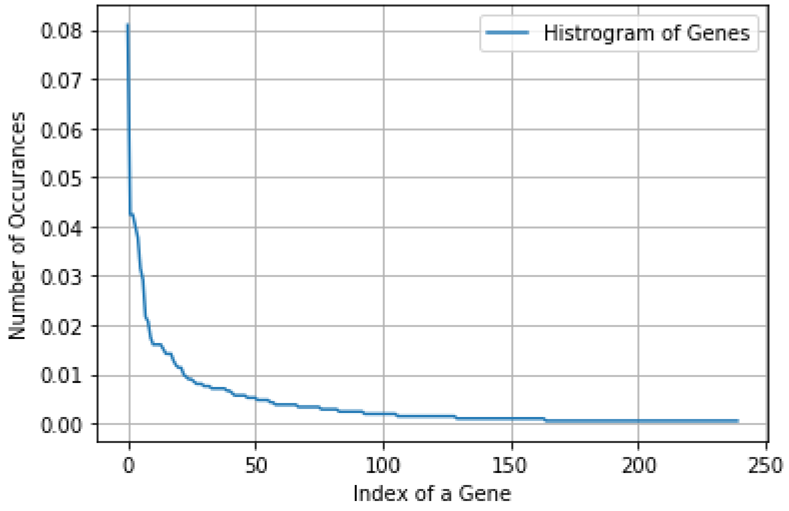
In [194]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 240
BRCA1     172
TP53       90
EGFR       90
PTEN       85
BRCA2      80
BRAF       67
KIT        62
ALK        46
ERBB2      44
PDGFRA     37
Name: Gene, dtype: int64
```

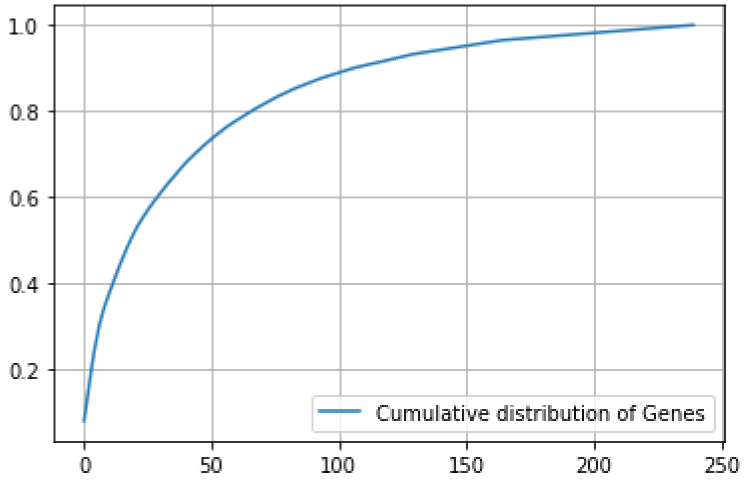
In [195]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in
```

```
Ans: There are 240 different categories of genes in the train data, and they ar
e distibuted as follows
```

In [196]:
```
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 [197]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [0]:
```
#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_
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_d
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [199]:
```
print("train_gene_feature_responseCoding is converted feature using respone codi
```

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

In [0]:
```
# tfidf encoding of Gene feature.
from sklearn.feature_extraction.text import TfidfVectorizer
gene_vectorizer =TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene']
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [201]:
```
train_df['Gene'].head()
```

Out[201]:
```
2823     BRCA2
3284       RET
1493     FGFR2
838       ABL1
3094    NOTCH1
Name: Gene, dtype: object
```

In [202]:
```
gene_vectorizer.get_feature_names()
```

Out[202]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
```

In [203]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encodi
```

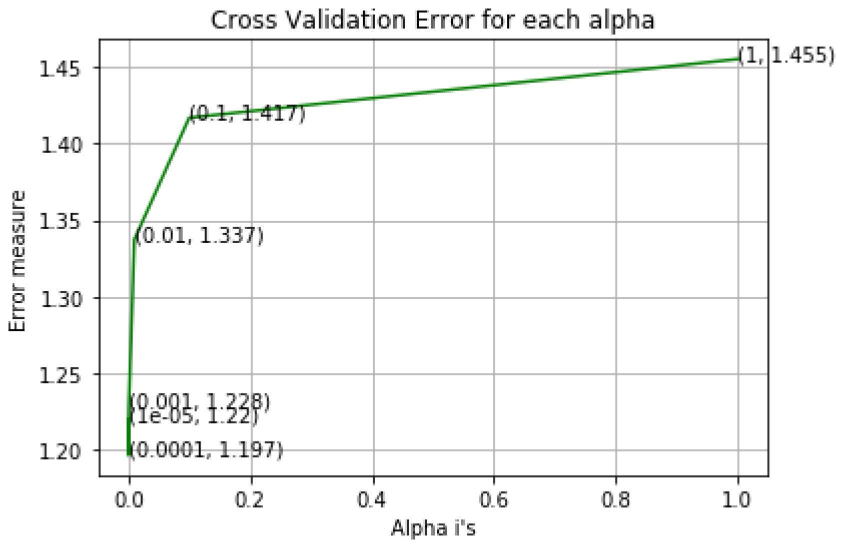
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding met
hod. The shape of gene feature: (2124, 240)
```

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

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

In [204]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_ra
# 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 Gra
# 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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

```
For values of alpha =  1e-05 The log loss is: 1.2196862811575042
For values of alpha =  0.0001 The log loss is: 1.196641798358647
For values of alpha =  0.001 The log loss is: 1.2280511960589267
For values of alpha =  0.01 The log loss is: 1.3372997135932572
For values of alpha =  0.1 The log loss is: 1.416848537161832
For values of alpha =  1 The log loss is: 1.4551437846180135
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9616445599819275
For values of best alpha =  0.0001 The cross validation log loss is: 1.19664179
8358647
For values of best alpha =  0.0001 The test log loss is: 1.2583283581324405
```

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 [205]:
```python
print("Q6. How many data points in Test and CV datasets are covered by the ", un

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

```
Q6. How many data points in Test and CV datasets are covered by the  240  genes
in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 521 out of  532 : 97.93233082706767
```

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?

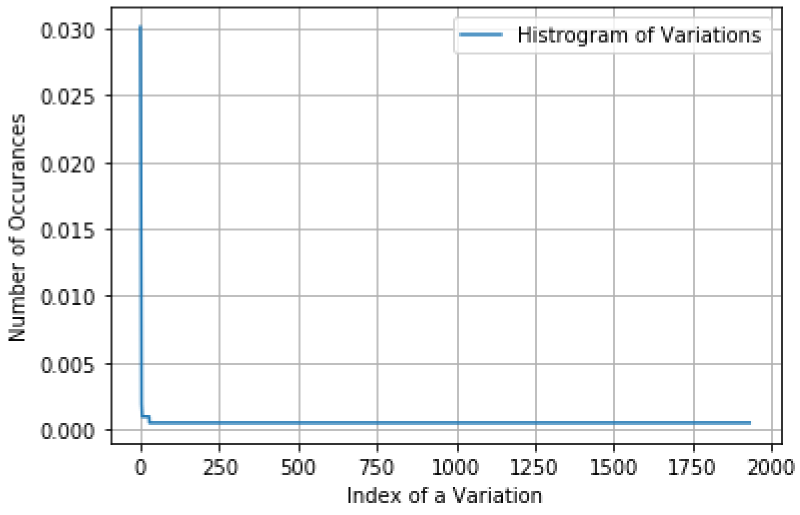
In [206]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1932
Truncating_Mutations    64
Deletion                45
Amplification           38
Fusions                 21
G12V                     4
T58I                     3
Y42C                     2
M1R                      2
Q61R                     2
G67R                     2
Name: Variation, dtype: int64
```

In [207]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of var
```

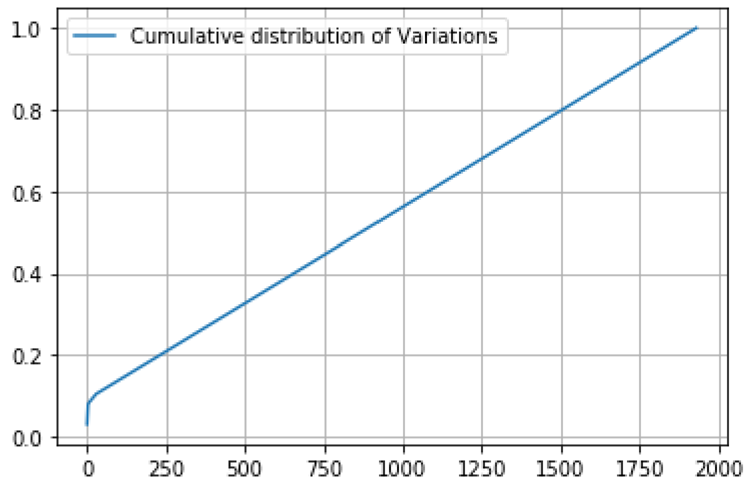
```
Ans: There are 1932 different categories of variations in the train data, and t
hey are distibuted as follows
```

In [208]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [209]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.03013183 0.05131827 0.06920904 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [0]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variati
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation"
```

In [211]:
```
print("train_variation_feature_responseCoding is a converted feature using the re
```

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

In [0]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Va
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variat
```

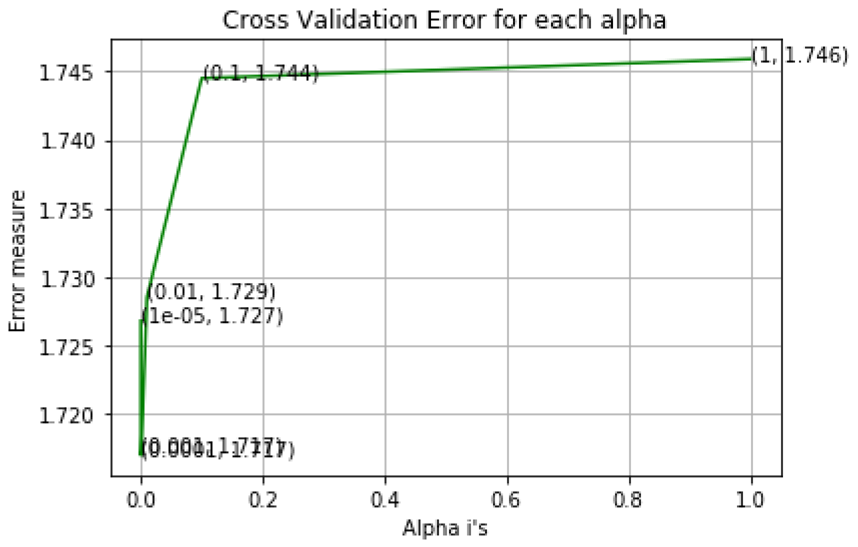
In [213]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot e
ncoding method. The shape of Variation feature: (2124, 1962)
```

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

Let's build a model just like the earlier!

In [214]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_ra
# 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 Gra
# 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
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_st
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:"
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
For values of alpha =  1e-05 The log loss is: 1.7268113070391837
For values of alpha =  0.0001 The log loss is: 1.7169873046873894
For values of alpha =  0.001 The log loss is: 1.7173715562297662
For values of alpha =  0.01 The log loss is: 1.7285280214950953
For values of alpha =  0.1 The log loss is: 1.7444983395011235
For values of alpha =  1 The log loss is: 1.7458978788999457
```



```
For values of best alpha =  0.0001 The train log loss is: 0.666138849940641
For values of best alpha =  0.0001 The cross validation log loss is: 1.71698730
46873894
For values of best alpha =  0.0001 The test log loss is: 1.7095908167434941
```

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 [215]:
```python
print("Q12. How many data points are covered by total ", unique_variations.shape
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation']))
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].sha
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(
```

```
Q12. How many data points are covered by total  1932  genes in test and cross v
alidation data sets?
Ans
1. In test data 78 out of 665 : 11.729323308270677
2. In cross validation data 49 out of  532 : 9.210526315789473
```

3.2.3 Univariate Analysis on Text Feature

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

In [0]:
```
# 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 [0]:
```
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
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(ro
            row_index += 1
    return text_feature_responseCoding
```

In [218]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in t
text_vectorizer = CountVectorizer(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
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 ti
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 [0]:
```
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 [0]:
```
#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 [0]:
```
# 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_te
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_featur
```

In [0]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis

# 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=(

# 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)
```

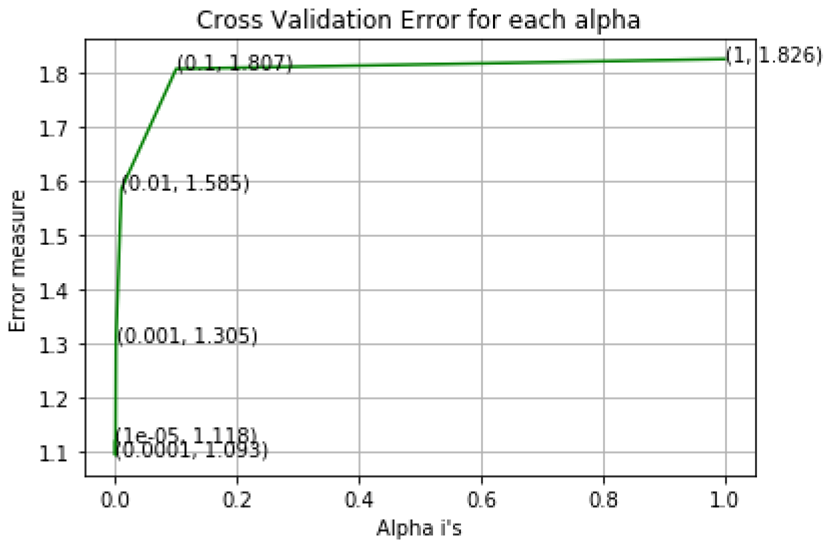
In [0]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , re
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [224]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3516: 3, 3242: 3, 2947: 3, 2851: 3, 2663: 3, 2646: 3, 9744: 2, 8276:
2, 8169: 2, 8079: 2, 6024: 2, 5997: 2, 5475: 2, 5438: 2, 5321: 2, 5285: 2, 478
5: 2, 4542: 2, 4432: 2, 4361: 2, 4194: 2, 4054: 2, 4023: 2, 3995: 2, 3853: 2, 3
821: 2, 3819: 2, 3718: 2, 3696: 2, 3590: 2, 3540: 2, 3527: 2, 3471: 2, 3442: 2,
3435: 2, 3390: 2, 3372: 2, 3368: 2, 3339: 2, 3277: 2, 3228: 2, 3220: 2, 3104:
2, 2853: 2, 2712: 2, 2708: 2, 2688: 2, 2679: 2, 2664: 2, 2649: 2, 2614: 2, 1562
54: 1, 119335: 1, 79775: 1, 68849: 1, 68432: 1, 68414: 1, 67340: 1, 64430: 1, 6
3845: 1, 56703: 1, 54721: 1, 50009: 1, 49626: 1, 47017: 1, 46396: 1, 44287: 1,
42877: 1, 42723: 1, 42377: 1, 42072: 1, 40889: 1, 40792: 1, 39979: 1, 39166: 1,
38853: 1, 38182: 1, 36492: 1, 36127: 1, 35957: 1, 35597: 1, 35027: 1, 34475: 1,
33849: 1, 33259: 1, 32816: 1, 31506: 1, 30344: 1, 29572: 1, 28174: 1, 26612: 1,
26489: 1, 26463: 1, 26292: 1, 25486: 1, 24859: 1, 24831: 1, 24715: 1, 24555: 1,
24369: 1, 24189: 1, 23988: 1, 23590: 1, 23021: 1, 22483: 1, 22444: 1, 22318: 1,
22182: 1, 21707: 1, 21556: 1, 21055: 1, 21040: 1, 20884: 1, 20616: 1, 20475: 1,
20021: 1, 19721: 1, 19706: 1, 19560: 1, 19510: 1, 19459: 1, 19326: 1, 19065: 1,
18907: 1, 18873: 1, 18662: 1, 18621: 1, 18291: 1, 18263: 1, 18168: 1, 18054: 1,
18022: 1, 17959: 1, 17740: 1, 17715: 1, 17623: 1, 17616: 1, 17538: 1, 17306: 1,
17221: 1, 17205: 1, 17196: 1, 17183: 1, 17163: 1, 17115: 1, 16999: 1, 16630: 1,
16266: 1, 16242: 1, 16228: 1, 16058: 1, 15896: 1, 15863: 1, 15846: 1, 15828: 1,
15751: 1, 15649: 1, 15615: 1, 15418: 1, 15296: 1, 15057: 1, 14985: 1, 14934: 1,
14834: 1, 14736: 1, 14702: 1, 14641: 1, 14602: 1, 14553: 1, 14525: 1, 14288: 1,
14251: 1, 14197: 1, 13948: 1, 13714: 1, 13669: 1, 13498: 1, 13481: 1, 13330: 1,
13326: 1, 13310: 1, 13288: 1, 13179: 1, 13168: 1, 13123: 1, 13110: 1, 13050: 1,
12956: 1, 12921: 1, 12799: 1, 12765: 1, 12715: 1, 12669: 1, 12621: 1, 12620: 1,
12581: 1, 12515: 1, 12466: 1, 12442: 1, 12437: 1, 12425: 1, 12392: 1, 12383: 1,
12381: 1, 12352: 1, 12350: 1, 12332: 1, 12302: 1, 12249: 1, 12136: 1, 12095: 1,
12032: 1, 12016: 1, 12001: 1, 11991: 1, 11972: 1, 11967: 1, 11961: 1, 11942: 1,
11915: 1, 11765: 1, 11762: 1, 11722: 1, 11709: 1, 11644: 1, 11626: 1, 11503: 1,
11473: 1, 11414: 1, 11159: 1, 11145: 1, 11103: 1, 11057: 1, 11033: 1, 11022: 1,
10882: 1, 10875: 1, 10854: 1, 10833: 1, 10827: 1, 10807: 1, 10657: 1, 10622: 1,
10516: 1, 10430: 1, 10428: 1, 10341: 1, 10340: 1, 10251: 1, 10239: 1, 10111: 1,
10080: 1, 10024: 1, 10019: 1, 9992: 1, 9949: 1, 9941: 1, 9910: 1, 9892: 1, 989
1: 1, 9872: 1, 9797: 1, 9795: 1, 9559: 1, 9545: 1, 9518: 1, 9423: 1, 9410: 1, 9
404: 1, 9383: 1, 9354: 1, 9302: 1, 9300: 1, 9193: 1, 9191: 1, 9180: 1, 9158: 1,
9140: 1, 9124: 1, 9075: 1, 9036: 1, 9000: 1, 8991: 1, 8964: 1, 8945: 1, 8914:
1, 8899: 1, 8878: 1, 8854: 1, 8838: 1, 8708: 1, 8688: 1, 8659: 1, 8623: 1, 860
6: 1, 8548: 1, 8525: 1, 8487: 1, 8484: 1, 8444: 1, 8431: 1, 8430: 1, 8425: 1, 8
361: 1, 8338: 1, 8321: 1, 8303: 1, 8299: 1, 8296: 1, 8256: 1, 8240: 1, 8232: 1,
8220: 1, 8178: 1, 8175: 1, 8081: 1, 8057: 1, 8037: 1, 8020: 1, 7989: 1, 7927:
1, 7922: 1, 7905: 1, 7897: 1, 7892: 1, 7877: 1, 7863: 1, 7814: 1, 7810: 1, 780
4: 1, 7789: 1, 7776: 1, 7753: 1, 7744: 1, 7741: 1, 7732: 1, 7726: 1, 7723: 1, 7
667: 1, 7622: 1, 7591: 1, 7561: 1, 7559: 1, 7548: 1, 7527: 1, 7512: 1, 7487: 1,
7418: 1, 7369: 1, 7360: 1, 7358: 1, 7357: 1, 7355: 1, 7299: 1, 7290: 1, 7288:
1, 7277: 1, 7226: 1, 7219: 1, 7215: 1, 7214: 1, 7197: 1, 7165: 1, 7144: 1, 713
6: 1, 7129: 1, 7123: 1, 7117: 1, 7110: 1, 7097: 1, 7079: 1, 7070: 1, 7060: 1, 7
018: 1, 6988: 1, 6969: 1, 6968: 1, 6954: 1, 6941: 1, 6887: 1, 6876: 1, 6851: 1,
6842: 1, 6828: 1, 6826: 1, 6818: 1, 6802: 1, 6798: 1, 6783: 1, 6767: 1, 6756:
1, 6745: 1, 6731: 1, 6729: 1, 6710: 1, 6695: 1, 6693: 1, 6686: 1, 6668: 1, 664
9: 1, 6636: 1, 6635: 1, 6617: 1, 6574: 1, 6565: 1, 6521: 1, 6504: 1, 6500: 1, 6
495: 1, 6447: 1, 6429: 1, 6421: 1, 6419: 1, 6409: 1, 6407: 1, 6402: 1, 6365: 1,
6347: 1, 6326: 1, 6313: 1, 6304: 1, 6302: 1, 6301: 1, 6279: 1, 6258: 1, 6255:
1, 6244: 1, 6241: 1, 6230: 1, 6222: 1, 6218: 1, 6211: 1, 6210: 1, 6204: 1, 619
6: 1, 6179: 1, 6162: 1, 6137: 1, 6100: 1, 6058: 1, 6048: 1, 6042: 1, 6036: 1, 6
034: 1, 6030: 1, 6004: 1, 6002: 1, 5999: 1, 5995: 1, 5993: 1, 5962: 1, 5949: 1,
5939: 1, 5911: 1, 5907: 1, 5905: 1, 5904: 1, 5894: 1, 5871: 1, 5836: 1, 5821:
1, 5816: 1, 5779: 1, 5763: 1, 5735: 1, 5721: 1, 5717: 1, 5711: 1, 5666: 1, 565
6: 1, 5642: 1, 5615: 1, 5611: 1, 5595: 1, 5593: 1, 5585: 1, 5581: 1, 5559: 1, 5
552: 1, 5549: 1, 5547: 1, 5542: 1, 5536: 1, 5520: 1, 5518: 1, 5513: 1, 5493: 1,
5486: 1, 5453: 1, 5450: 1, 5441: 1, 5420: 1, 5401: 1, 5399: 1, 5353: 1, 5313:
1, 5310: 1, 5299: 1, 5278: 1, 5254: 1, 5245: 1, 5234: 1, 5233: 1, 5217: 1, 519
5: 1, 5182: 1, 5166: 1, 5146: 1, 5143: 1, 5136: 1, 5135: 1, 5126: 1, 5124: 1, 5
119: 1, 5104: 1, 5094: 1, 5074: 1, 5071: 1, 5063: 1, 5050: 1, 5043: 1, 5036: 1,
5006: 1, 5001: 1, 4977: 1, 4976: 1, 4972: 1, 4964: 1, 4958: 1, 4953: 1, 4949:
1, 4940: 1, 4935: 1, 4927: 1, 4923: 1, 4918: 1, 4901: 1, 4886: 1, 4882: 1, 487
6: 1, 4865: 1, 4846: 1, 4843: 1, 4834: 1, 4827: 1, 4807: 1, 4801: 1, 4798: 1, 4
795: 1, 4787: 1, 4778: 1, 4775: 1, 4739: 1, 4737: 1, 4734: 1, 4733: 1, 4710: 1,
4688: 1, 4680: 1, 4670: 1, 4669: 1, 4656: 1, 4653: 1, 4627: 1, 4611: 1, 4608:
1, 4594: 1, 4592: 1, 4573: 1, 4563: 1, 4560: 1, 4546: 1, 4519: 1, 4517: 1, 451
6: 1, 4511: 1, 4509: 1, 4505: 1, 4501: 1, 4496: 1, 4495: 1, 4483: 1, 4479: 1, 4
471: 1, 4470: 1, 4468: 1, 4464: 1, 4461: 1, 4421: 1, 4416: 1, 4392: 1, 4390: 1,
4383: 1, 4379: 1, 4376: 1, 4374: 1, 4359: 1, 4354: 1, 4327: 1, 4325: 1, 4313:
1, 4306: 1, 4305: 1, 4300: 1, 4299: 1, 4291: 1, 4283: 1, 4277: 1, 4270: 1, 426
9: 1, 4261: 1, 4258: 1, 4246: 1, 4235: 1, 4234: 1, 4229: 1, 4226: 1, 4224: 1, 4
216: 1, 4213: 1, 4201: 1, 4198: 1, 4196: 1, 4179: 1, 4168: 1, 4164: 1, 4154: 1,
4149: 1, 4134: 1, 4128: 1, 4127: 1, 4126: 1, 4119: 1, 4115: 1, 4114: 1, 4111:
1, 4108: 1, 4093: 1, 4092: 1, 4090: 1, 4087: 1, 4074: 1, 4073: 1, 4067: 1, 404
8: 1, 4046: 1, 4044: 1, 4042: 1, 4033: 1, 4025: 1, 4024: 1, 4021: 1, 4019: 1, 4
011: 1, 3997: 1, 3994: 1, 3987: 1, 3981: 1, 3980: 1, 3933: 1, 3926: 1, 3925: 1,
3921: 1, 3917: 1, 3907: 1, 3904: 1, 3903: 1, 3899: 1, 3897: 1, 3892: 1, 3891:
1, 3879: 1, 3867: 1, 3864: 1, 3862: 1, 3861: 1, 3851: 1, 3850: 1, 3848: 1, 384
7: 1, 3845: 1, 3839: 1, 3830: 1, 3828: 1, 3824: 1, 3814: 1, 3806: 1, 3804: 1, 3
803: 1, 3802: 1, 3793: 1, 3777: 1, 3768: 1, 3766: 1, 3760: 1, 3752: 1, 3749: 1,
3747: 1, 3746: 1, 3738: 1, 3726: 1, 3725: 1, 3720: 1, 3719: 1, 3714: 1, 3707:
1, 3704: 1, 3702: 1, 3701: 1, 3692: 1, 3684: 1, 3678: 1, 3668: 1, 3666: 1, 366
3: 1, 3654: 1, 3652: 1, 3647: 1, 3627: 1, 3621: 1, 3609: 1, 3599: 1, 3593: 1, 3
591: 1, 3589: 1, 3588: 1, 3584: 1, 3579: 1, 3573: 1, 3564: 1, 3562: 1, 3556: 1,
3553: 1, 3548: 1, 3533: 1, 3532: 1, 3531: 1, 3530: 1, 3526: 1, 3520: 1, 3518:
1, 3517: 1, 3513: 1, 3509: 1, 3507: 1, 3504: 1, 3497: 1, 3483: 1, 3476: 1, 347
3: 1, 3466: 1, 3462: 1, 3459: 1, 3456: 1, 3452: 1, 3451: 1, 3450: 1, 3438: 1, 3
436: 1, 3428: 1, 3423: 1, 3422: 1, 3416: 1, 3398: 1, 3389: 1, 3388: 1, 3387: 1,
3385: 1, 3384: 1, 3381: 1, 3380: 1, 3378: 1, 3374: 1, 3371: 1, 3366: 1, 3358:
1, 3357: 1, 3351: 1, 3348: 1, 3330: 1, 3328: 1, 3323: 1, 3322: 1, 3317: 1, 330
6: 1, 3303: 1, 3301: 1, 3295: 1, 3293: 1, 3292: 1, 3290: 1, 3289: 1, 3288: 1, 3
287: 1, 3282: 1, 3278: 1, 3274: 1, 3272: 1, 3263: 1, 3252: 1, 3236: 1, 3221: 1,
3215: 1, 3206: 1, 3202: 1, 3201: 1, 3198: 1, 3197: 1, 3195: 1, 3194: 1, 3193:
1, 3190: 1, 3187: 1, 3184: 1, 3182: 1, 3181: 1, 3179: 1, 3173: 1, 3159: 1, 315
5: 1, 3150: 1, 3142: 1, 3141: 1, 3137: 1, 3127: 1, 3124: 1, 3121: 1, 3116: 1, 3
115: 1, 3111: 1, 3110: 1, 3107: 1, 3105: 1, 3092: 1, 3087: 1, 3078: 1, 3073: 1,
3072: 1, 3070: 1, 3067: 1, 3066: 1, 3065: 1, 3064: 1, 3063: 1, 3061: 1, 3053:
1, 3052: 1, 3046: 1, 3045: 1, 3044: 1, 3043: 1, 3035: 1, 3034: 1, 3030: 1, 302
6: 1, 3023: 1, 3021: 1, 3017: 1, 3011: 1, 2999: 1, 2997: 1, 2981: 1, 2976: 1, 2
974: 1, 2973: 1, 2967: 1, 2964: 1, 2959: 1, 2958: 1, 2955: 1, 2945: 1, 2944: 1,
2940: 1, 2939: 1, 2922: 1, 2921: 1, 2918: 1, 2915: 1, 2895: 1, 2894: 1, 2867:
1, 2856: 1, 2854: 1, 2846: 1, 2845: 1, 2838: 1, 2836: 1, 2834: 1, 2833: 1, 282
0: 1, 2819: 1, 2814: 1, 2811: 1, 2796: 1, 2791: 1, 2782: 1, 2779: 1, 2778: 1, 2
771: 1, 2767: 1, 2765: 1, 2760: 1, 2759: 1, 2751: 1, 2745: 1, 2739: 1, 2738: 1,
2737: 1, 2733: 1, 2730: 1, 2727: 1, 2725: 1, 2722: 1, 2720: 1, 2715: 1, 2707:
1, 2702: 1, 2701: 1, 2689: 1, 2687: 1, 2685: 1, 2676: 1, 2675: 1, 2674: 1, 267
3: 1, 2671: 1, 2670: 1, 2669: 1, 2662: 1, 2659: 1, 2652: 1, 2645: 1, 2641: 1, 2
627: 1, 2623: 1, 2618: 1, 2617: 1, 2616: 1, 2615: 1, 2612: 1, 2611: 1, 2606: 1,
2604: 1})
```

In [225]:
```
# Train a Logistic regression+Calibration model using text features whicha re on
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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

```
For values of alpha =  1e-05 The log loss is: 1.1180902361138372
For values of alpha =  0.0001 The log loss is: 1.092857045159537
For values of alpha =  0.001 The log loss is: 1.3046340776271625
For values of alpha =  0.01 The log loss is: 1.5850821003235362
For values of alpha =  0.1 The log loss is: 1.807413747288979
For values of alpha =  1 The log loss is: 1.825519846312339
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9420862854495291
For values of best alpha =  0.0001 The cross validation log loss is: 1.09285704
5159537
For values of best alpha =  0.0001 The test log loss is: 1.2697506177586746
```

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

Ans. Yes, it seems like!

In [0]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [227]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train da
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in t
```

```
3.504 % of word of test data appeared in train data
4.028 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [0]:
```
#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 belong
    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)
    plot_confusion_matrix(test_y, pred_y)
```

In [0]:
```
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 [0]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".for
        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 [{}]
        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 [{}]".for

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

Stacking the three types of features

In [0]:
```
# 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_vari
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,tra
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variat

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_fe
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_featu
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_res
```

In [232]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_o
print("(number of data points * number of features) in test data = ", test_x_one
print("(number of data points * number of features) in cross validation data =",
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3202)
(number of data points * number of features) in test data =  (665, 3202)
(number of data points * number of features) in cross validation data = (532, 3
202)
```

In [233]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_r
print("(number of data points * number of features) in test data = ", test_x_res
print("(number of data points * number of features) in cross validation data =",
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 2
7)
```

4.1. Base Line Model

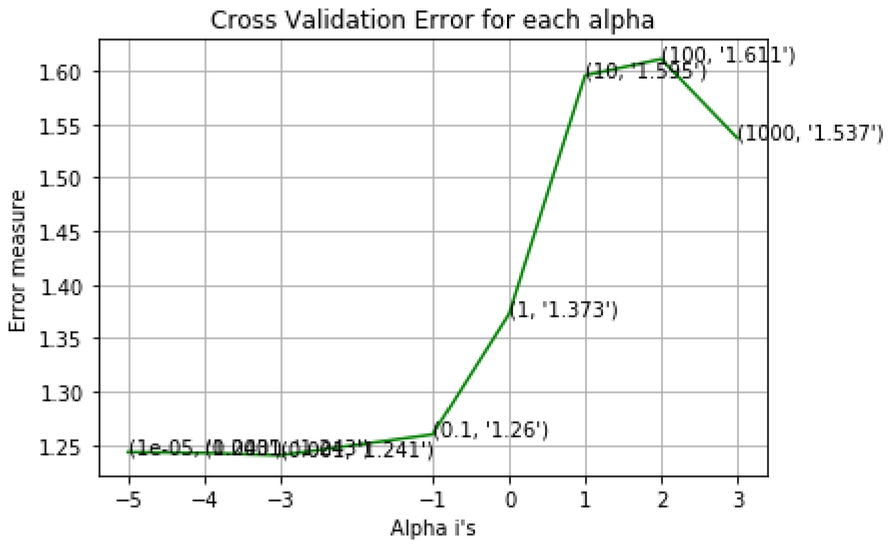
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [234]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/st
# -------------------------
# 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/les
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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/les
# -----------------------


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

```
for alpha = 1e-05
Log Loss : 1.2434369280653783
for alpha = 0.0001
Log Loss : 1.242646740693285
for alpha = 0.001
Log Loss : 1.2405020810469456
for alpha = 0.1
Log Loss : 1.259943941894815
for alpha = 1
Log Loss : 1.3727355285780591
for alpha = 10
Log Loss : 1.595450541683481
for alpha = 100
Log Loss : 1.6107506046092936
for alpha = 1000
Log Loss : 1.5368193367112561
```

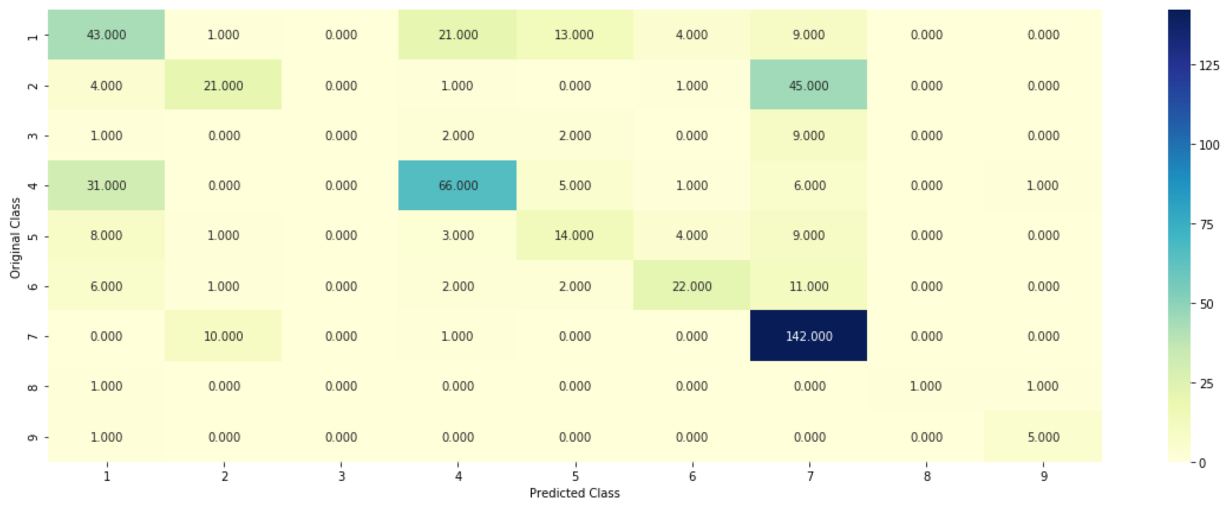


```
For values of best alpha =  0.001 The train log loss is: 0.4446663236298083
For values of best alpha =  0.001 The cross validation log loss is: 1.240502081
0469456
For values of best alpha =  0.001 The test log loss is: 1.2911328494660967
```

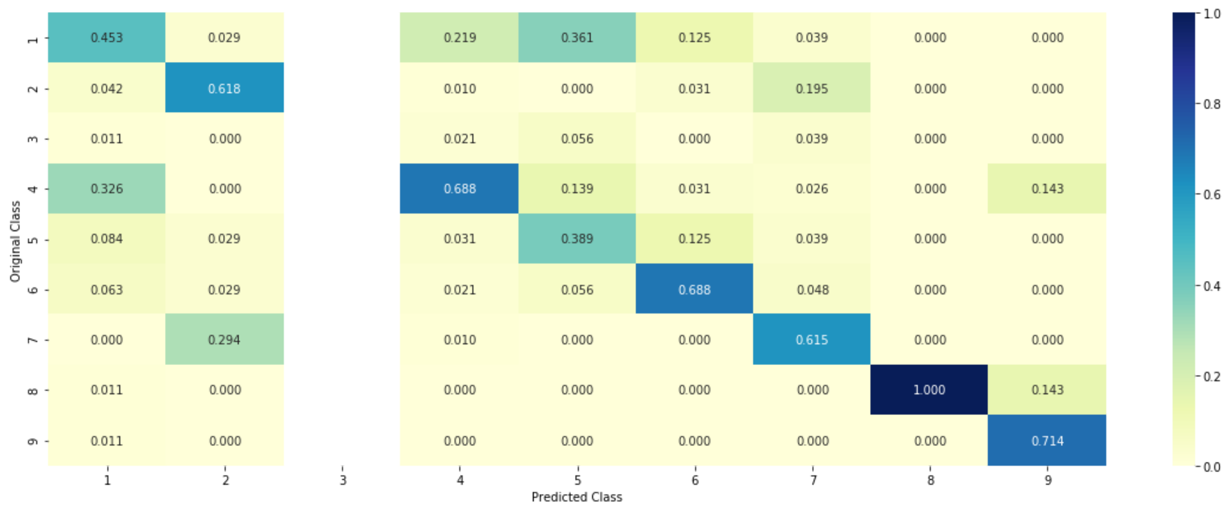
4.1.1.2. Testing the model with best hyper paramters

In [235]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/s
# -------------------------
# 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/le
# -----------------------


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

```
Log Loss : 1.2405020810469456
Number of missclassified point : 0.40977443609022557
-------------------- Confusion matrix --------------------
```

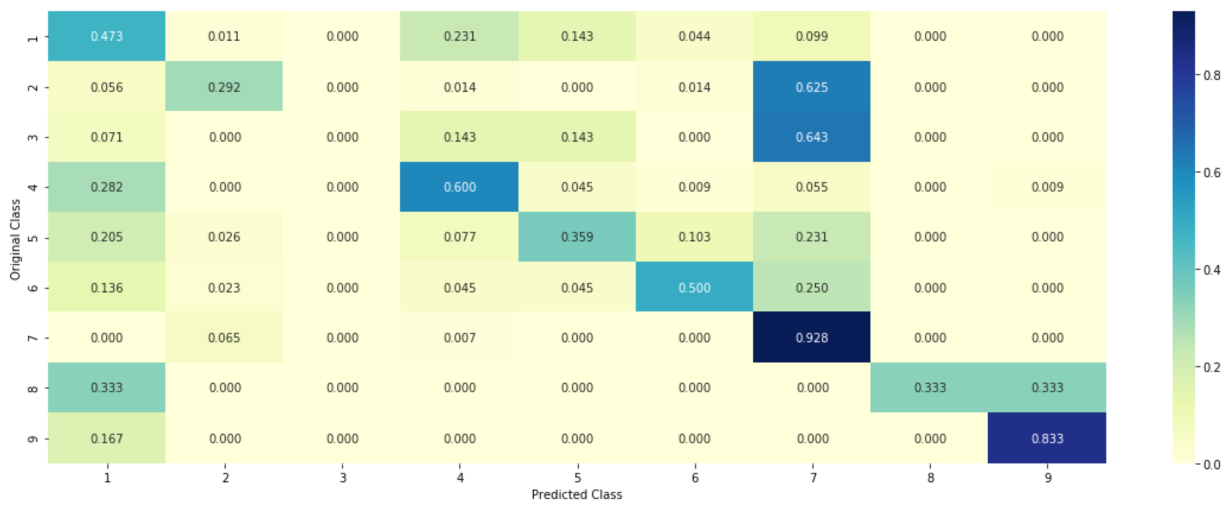


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.453	0.029		0.219	0.361	0.125	0.039	0.000	0.000
2	0.042	0.618		0.010	0.000	0.031	0.195	0.000	0.000
3	0.011	0.000		0.021	0.056	0.000	0.039	0.000	0.000
4	0.326	0.000		0.688	0.139	0.031	0.026	0.000	0.143
5	0.084	0.029		0.031	0.389	0.125	0.039	0.000	0.000
6	0.063	0.029		0.021	0.056	0.688	0.048	0.000	0.000
7	0.000	0.294		0.010	0.000	0.000	0.615	0.000	0.000
8	0.011	0.000		0.000	0.000	0.000	0.000	1.000	0.143
9	0.011	0.000		0.000	0.000	0.000	0.000	0.000	0.714

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



	1	2	3	4	5	6	7	8	9
1	0.473	0.011	0.000	0.231	0.143	0.044	0.099	0.000	0.000
2	0.056	0.292	0.000	0.014	0.000	0.014	0.625	0.000	0.000
3	0.071	0.000	0.000	0.143	0.143	0.000	0.643	0.000	0.000
4	0.282	0.000	0.000	0.600	0.045	0.009	0.055	0.000	0.009
5	0.205	0.026	0.000	0.077	0.359	0.103	0.231	0.000	0.000
6	0.136	0.023	0.000	0.045	0.045	0.500	0.250	0.000	0.000
7	0.000	0.065	0.000	0.007	0.000	0.000	0.928	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.333
9	0.167	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.833

4.1.1.3. Feature Importance, Correctly classified point

In [236]:
```
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_one
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[
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0615 0.6733 0.0126 0.0708 0.0372 0.0278 0.11
07 0.0031 0.003 ]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [237]:
```
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_one
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[
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.056  0.0837 0.0118 0.0645 0.0341 0.0252 0.71
91 0.0028 0.0027]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

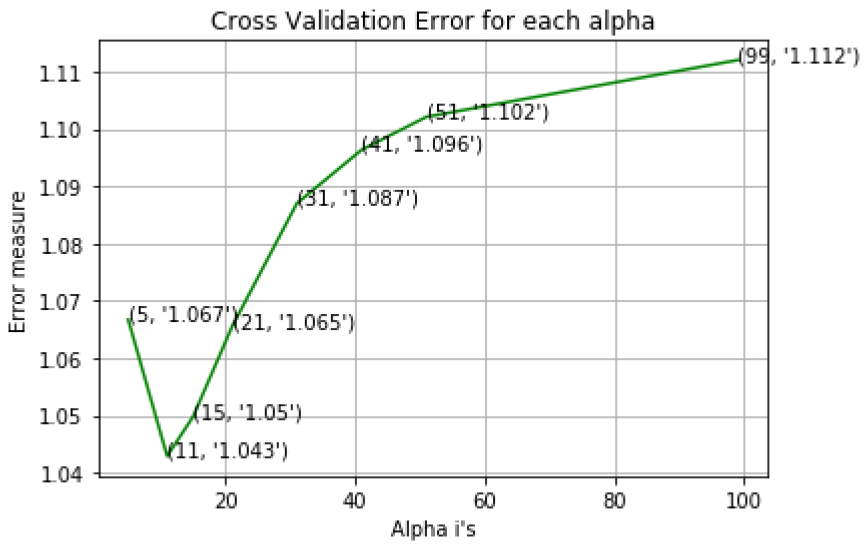
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [238]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# 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/les
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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_,
    # to avoid rounding error while multiplying probabilites we use log-probabil
    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:",
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation lo
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",
```

```
for alpha = 5
Log Loss : 1.0667280502334866
for alpha = 11
Log Loss : 1.042914009281993
for alpha = 15
Log Loss : 1.0497939647357155
for alpha = 21
Log Loss : 1.0654753018649226
for alpha = 31
Log Loss : 1.0869915420692449
for alpha = 41
Log Loss : 1.0963424982920769
for alpha = 51
Log Loss : 1.102095675318619
for alpha = 99
Log Loss : 1.1119909570876292
```

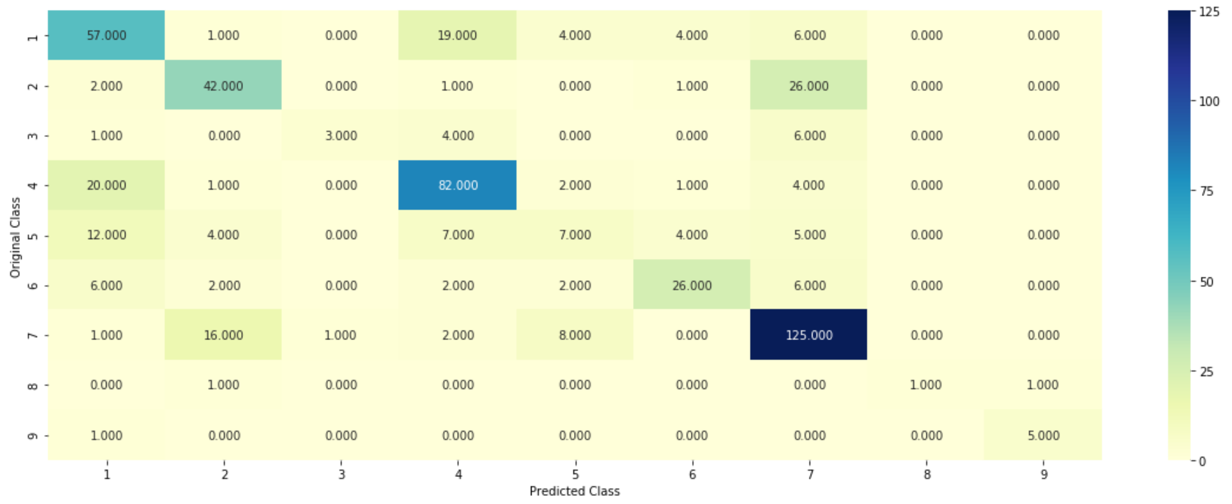


```
For values of best alpha =  11 The train log loss is: 0.5722617567923619
For values of best alpha =  11 The cross validation log loss is: 1.0429140092
81993
For values of best alpha =  11 The test log loss is: 1.167874383009327
```

4.2.2. Testing the model with best hyper paramters

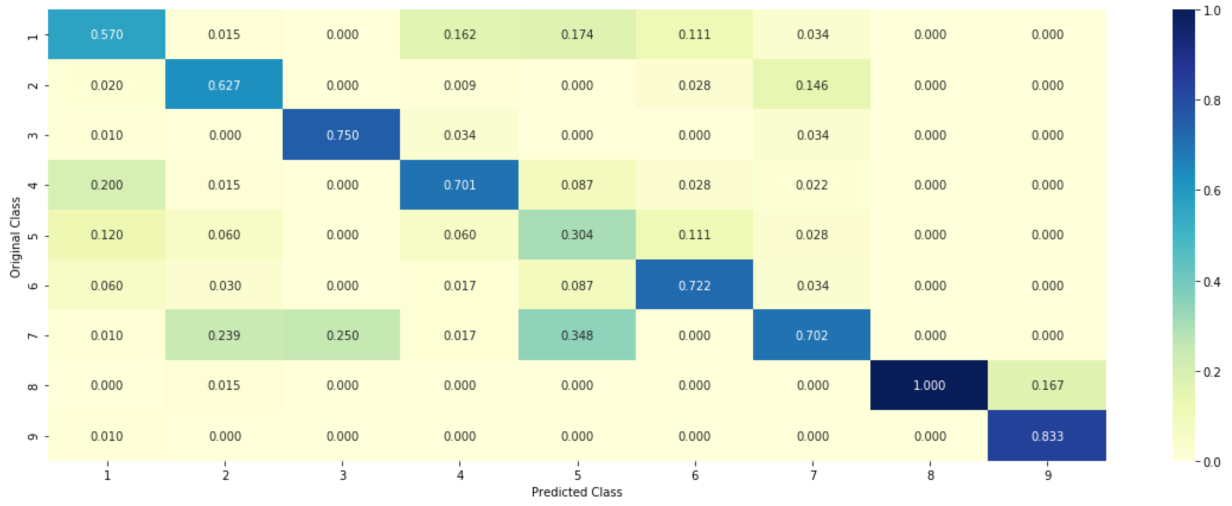
In [239]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.042914009281993
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```



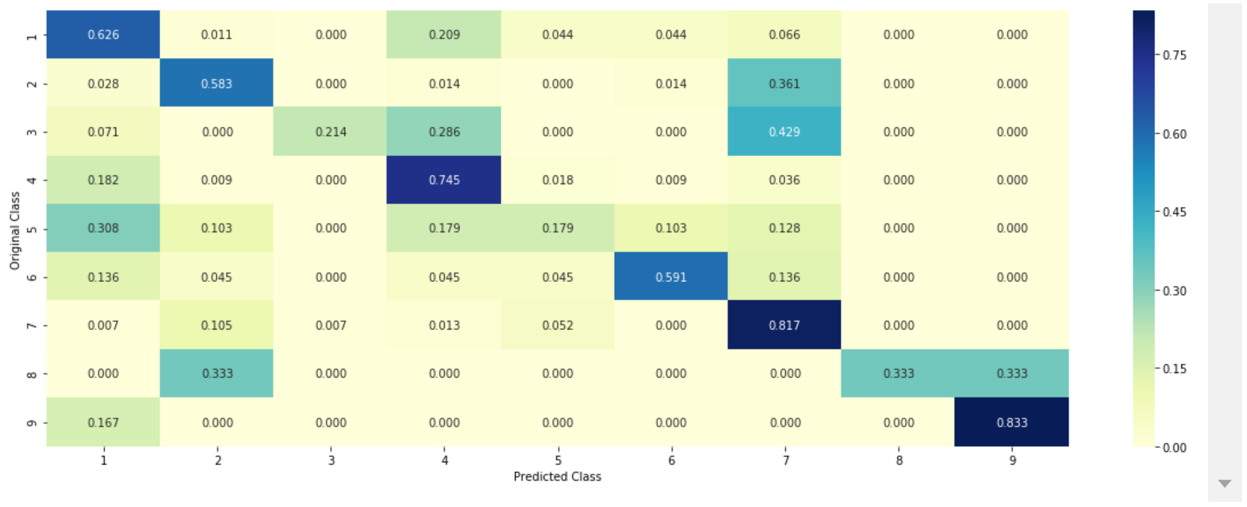
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	57.000	1.000	0.000	19.000	4.000	4.000	6.000	0.000	0.000
2	2.000	42.000	0.000	1.000	0.000	1.000	26.000	0.000	0.000
3	1.000	0.000	3.000	4.000	0.000	0.000	6.000	0.000	0.000
4	20.000	1.000	0.000	82.000	2.000	1.000	4.000	0.000	0.000
5	12.000	4.000	0.000	7.000	7.000	4.000	5.000	0.000	0.000
6	6.000	2.000	0.000	2.000	2.000	26.000	6.000	0.000	0.000
7	1.000	16.000	1.000	2.000	8.000	0.000	125.000	0.000	0.000
8	0.000	1.000	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	5.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.570	0.015	0.000	0.162	0.174	0.111	0.034	0.000	0.000
2	0.020	0.627	0.000	0.009	0.000	0.028	0.146	0.000	0.000
3	0.010	0.000	0.750	0.034	0.000	0.000	0.034	0.000	0.000
4	0.200	0.015	0.000	0.701	0.087	0.028	0.022	0.000	0.000
5	0.120	0.060	0.000	0.060	0.304	0.111	0.028	0.000	0.000
6	0.060	0.030	0.000	0.017	0.087	0.722	0.034	0.000	0.000
7	0.010	0.239	0.250	0.017	0.348	0.000	0.702	0.000	0.000
8	0.000	0.015	0.000	0.000	0.000	0.000	0.000	1.000	0.167
9	0.010	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.833

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



4.2.3.Sample Query point -1

In [240]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 2
Actual Class : 2
The  11  nearest neighbours of the test points belongs to classes [2 2 2 2 2 7
6 7 7 2 2]
Fequency of nearest points : Counter({2: 7, 7: 3, 6: 1})
```

4.2.4. Sample Query Point-2

```
In [241]: 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(
          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
          print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of 
          print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 2
the k value for knn is 11 and the nearest neighbours of the test points belongs
to classes [7 7 7 2 7 7 7 2 7 2 7]
Fequency of nearest points : Counter({7: 8, 2: 3})
```

4.3. Logistic Regression

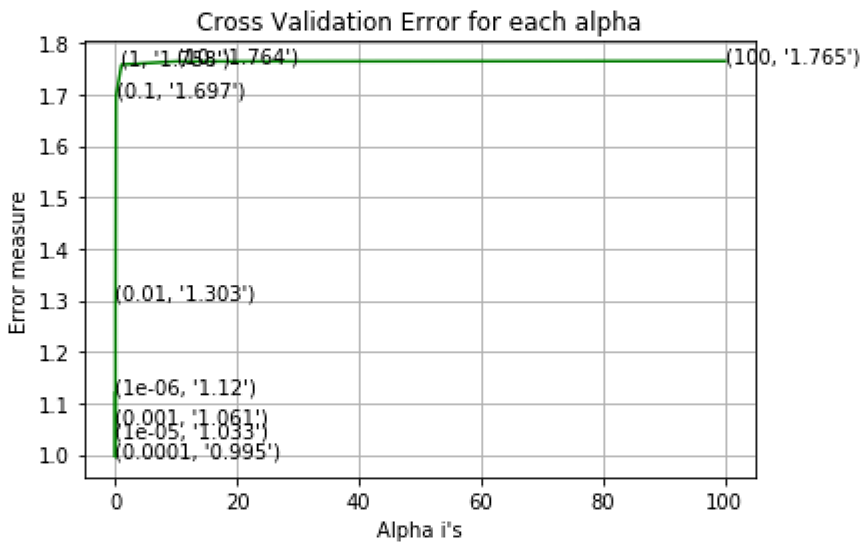
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [242]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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



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

```
for alpha = 1e-06
Log Loss : 1.1196885392067795
for alpha = 1e-05
Log Loss : 1.0334671073269235
for alpha = 0.0001
Log Loss : 0.9949046441610732
for alpha = 0.001
Log Loss : 1.0606835537447814
for alpha = 0.01
Log Loss : 1.3026234324919055
for alpha = 0.1
Log Loss : 1.6967828939993292
for alpha = 1
Log Loss : 1.7579991217024282
for alpha = 10
Log Loss : 1.7642823631430968
for alpha = 100
Log Loss : 1.7649934471556046
```

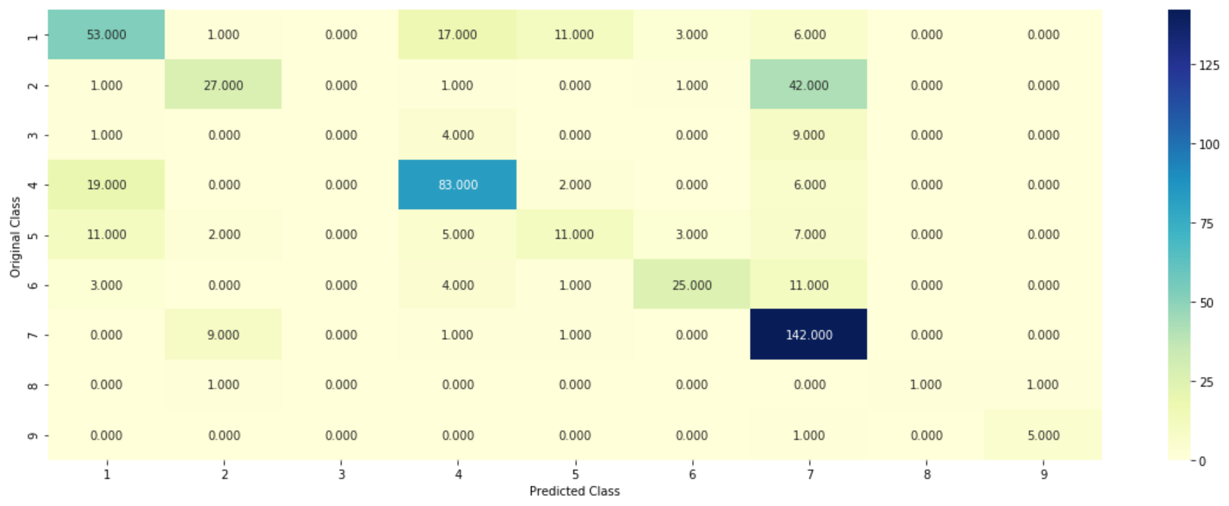


```
For values of best alpha =  0.0001 The train log loss is: 0.3820179887209377
For values of best alpha =  0.0001 The cross validation log loss is: 0.99490464
41610732
For values of best alpha =  0.0001 The test log loss is: 1.094996784841714
```

4.3.1.2. Testing the model with best hyper paramters

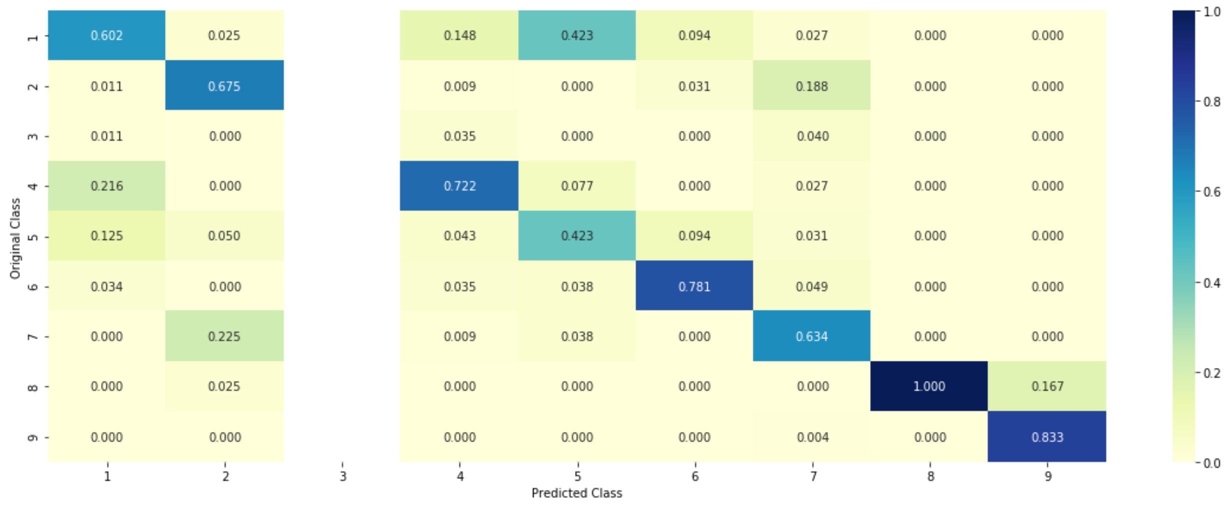
In [243]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/les
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l1
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodin
```

```
Log loss : 0.9949046441610732
Number of mis-classified points : 0.347744360902255563
-------------------- Confusion matrix --------------------
```



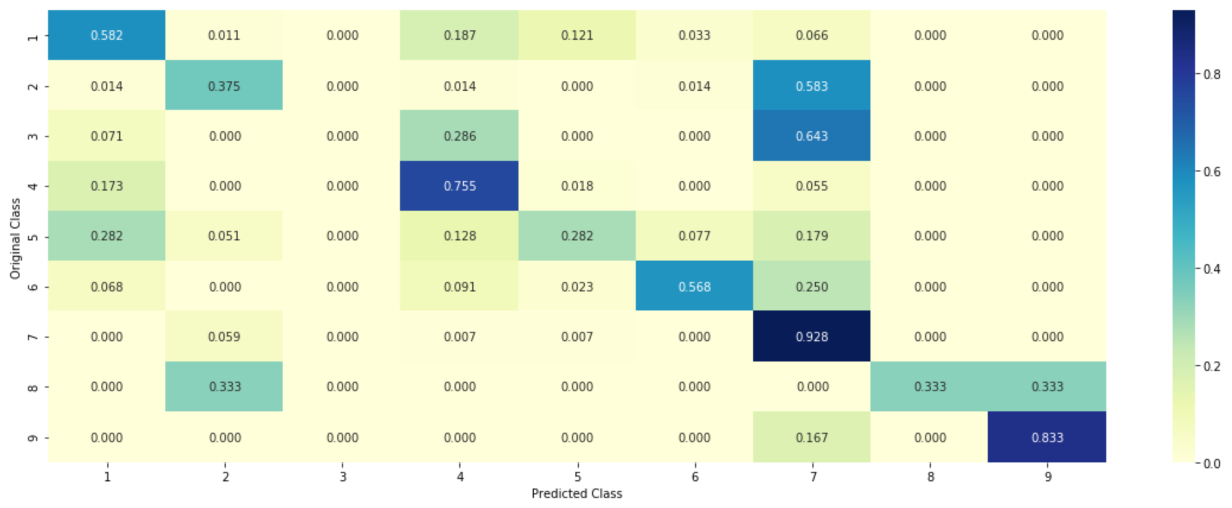
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	53.000	1.000	0.000	17.000	11.000	3.000	6.000	0.000	0.000
2	1.000	27.000	0.000	1.000	0.000	1.000	42.000	0.000	0.000
3	1.000	0.000	0.000	4.000	0.000	0.000	9.000	0.000	0.000
4	19.000	0.000	0.000	83.000	2.000	0.000	6.000	0.000	0.000
5	11.000	2.000	0.000	5.000	11.000	3.000	7.000	0.000	0.000
6	3.000	0.000	0.000	4.000	1.000	25.000	11.000	0.000	0.000
7	0.000	9.000	0.000	1.000	1.000	0.000	142.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.602	0.025		0.148	0.423	0.094	0.027	0.000	0.000
2	0.011	0.675		0.009	0.000	0.031	0.188	0.000	0.000
3	0.011	0.000		0.035	0.000	0.000	0.040	0.000	0.000
4	0.216	0.000		0.722	0.077	0.000	0.027	0.000	0.000
5	0.125	0.050		0.043	0.423	0.094	0.031	0.000	0.000
6	0.034	0.000		0.035	0.038	0.781	0.049	0.000	0.000
7	0.000	0.225		0.009	0.038	0.000	0.634	0.000	0.000
8	0.000	0.025		0.000	0.000	0.000	0.000	1.000	0.167
9	0.000	0.000		0.000	0.000	0.000	0.004	0.000	0.833

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



4.3.1.3. Feature Importance

```
In [0]: 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_
                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]," clas
            print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or No
```

4.3.1.3.1. Correctly Classified point

In [245]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l1
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_one
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[
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0733 0.767  0.016  0.0216 0.0217 0.0311 0.05
49 0.0074 0.0071]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [246]:
```
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_one
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[
```

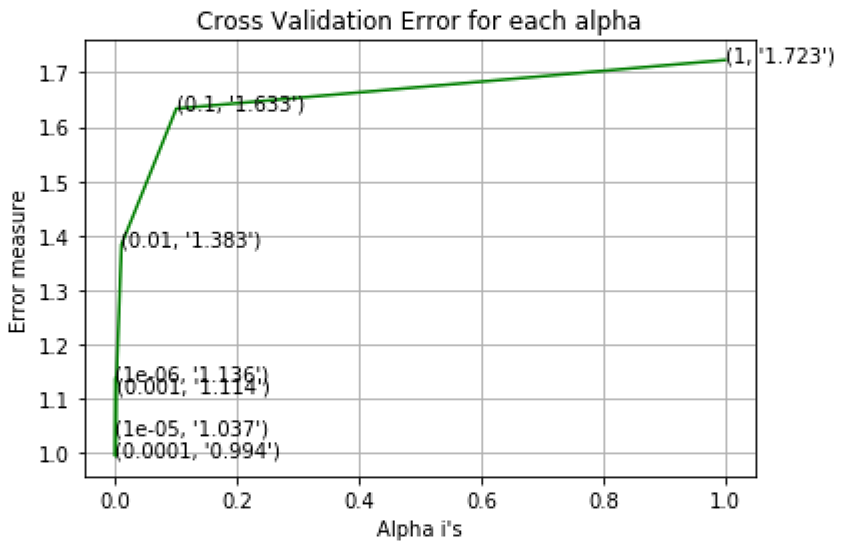
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0119 0.1657 0.0404 0.0097 0.0606 0.0045 0.70
14 0.0031 0.0029]]
Actual Class : 2
--------------------------------------------------
66 Text feature [10] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [247]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_ra
# 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 Gra
# predict(X)    Predict class labels for samples in X.

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



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

```
for alpha = 1e-06
Log Loss : 1.136482140857177
for alpha = 1e-05
Log Loss : 1.0368376251084792
for alpha = 0.0001
Log Loss : 0.9938234479054255
for alpha = 0.001
Log Loss : 1.1143680745282043
for alpha = 0.01
Log Loss : 1.3829721405841202
for alpha = 0.1
Log Loss : 1.6334785650563497
for alpha = 1
Log Loss : 1.722956626558638
```

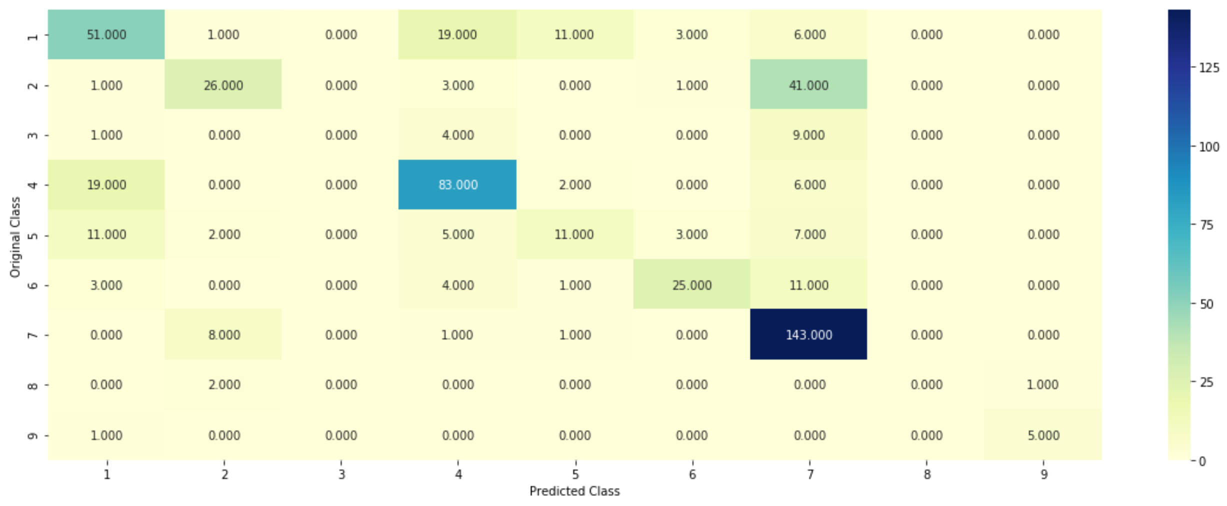


```
For values of best alpha =  0.0001 The train log loss is: 0.37658416351640145
For values of best alpha =  0.0001 The cross validation log loss is: 0.99382344
79054255
For values of best alpha =  0.0001 The test log loss is: 1.0852118785249474
```

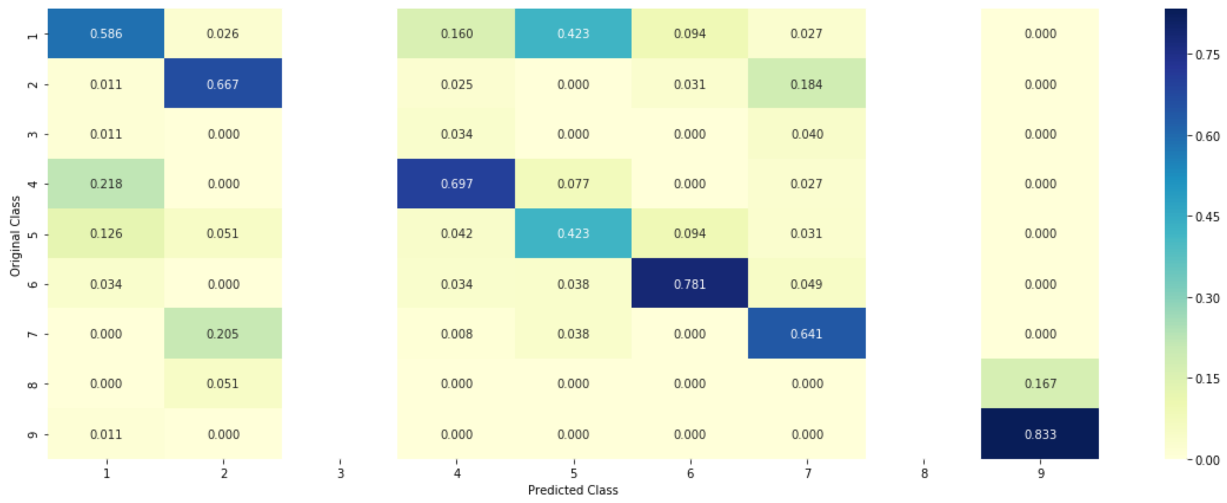
4.3.2.2. Testing model with best hyper parameters

In [248]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_sta
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
```

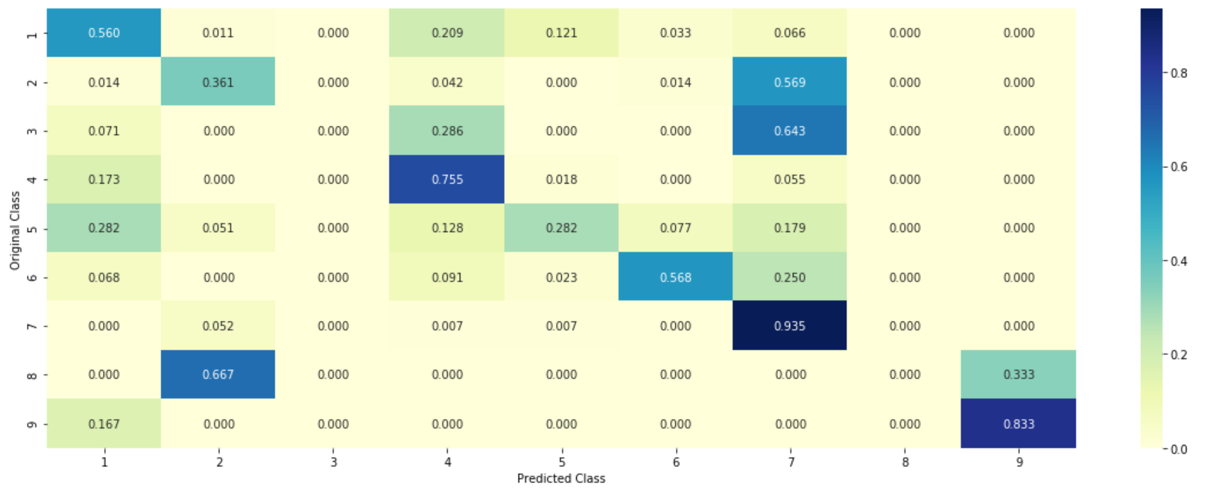
```
Log loss : 0.9938234479054255
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.560	0.011	0.000	0.209	0.121	0.033	0.066	0.000	0.000
2	0.014	0.361	0.000	0.042	0.000	0.014	0.569	0.000	0.000
3	0.071	0.000	0.000	0.286	0.000	0.000	0.643	0.000	0.000
4	0.173	0.000	0.000	0.755	0.018	0.000	0.055	0.000	0.000
5	0.282	0.051	0.000	0.128	0.282	0.077	0.179	0.000	0.000
6	0.068	0.000	0.000	0.091	0.023	0.568	0.250	0.000	0.000
7	0.000	0.052	0.000	0.007	0.007	0.000	0.935	0.000	0.000
8	0.000	0.667	0.000	0.000	0.000	0.000	0.000	0.000	0.333
9	0.167	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.833

4.3.2.3. Feature Importance, Correctly Classified point

In [249]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_st
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_one
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[
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0699 0.7681 0.0158 0.0219 0.0228 0.0342 0.05
19 0.0083 0.0072]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [250]:
```
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_on
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[
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0119 0.1755 0.0479 0.0108 0.0613 0.0049 0.68
19 0.0032 0.0025]]
Actual Class : 2
--------------------------------------------------
92 Text feature [10] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

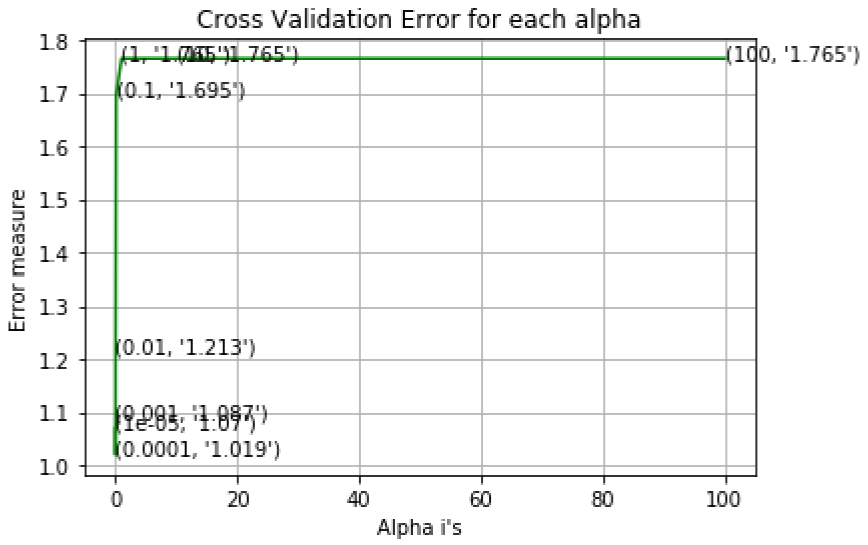
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [251]:
```
# read more about support vector machines with linear kernals here http://scikit

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

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




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

```
for C = 1e-05
Log Loss : 1.0698822969097057
for C = 0.0001
Log Loss : 1.0193858139473322
for C = 0.001
Log Loss : 1.0872484058054963
for C = 0.01
Log Loss : 1.2134336253066649
for C = 0.1
Log Loss : 1.6951515535703288
for C = 1
Log Loss : 1.7651671192013214
for C = 10
Log Loss : 1.765165253959348
for C = 100
Log Loss : 1.765166614339717
```

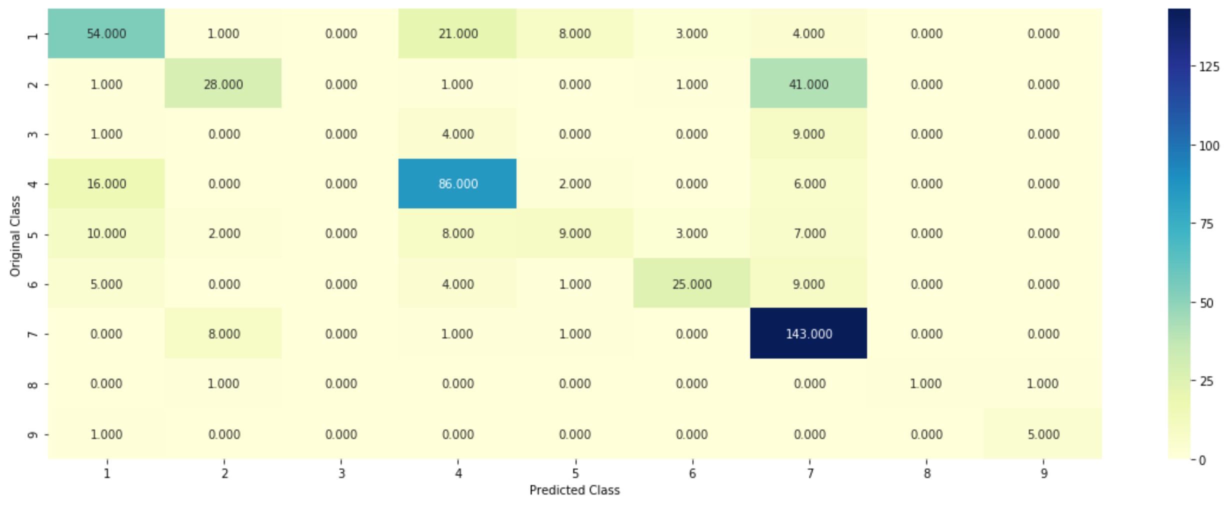


```
For values of best alpha =  0.0001 The train log loss is: 0.30631083324303 62
For values of best alpha =  0.0001 The cross validation log loss is: 1.01938581
39473322
For values of best alpha =  0.0001 The test log loss is: 1.1181703217284822
```

4.4.2. Testing model with best hyper parameters

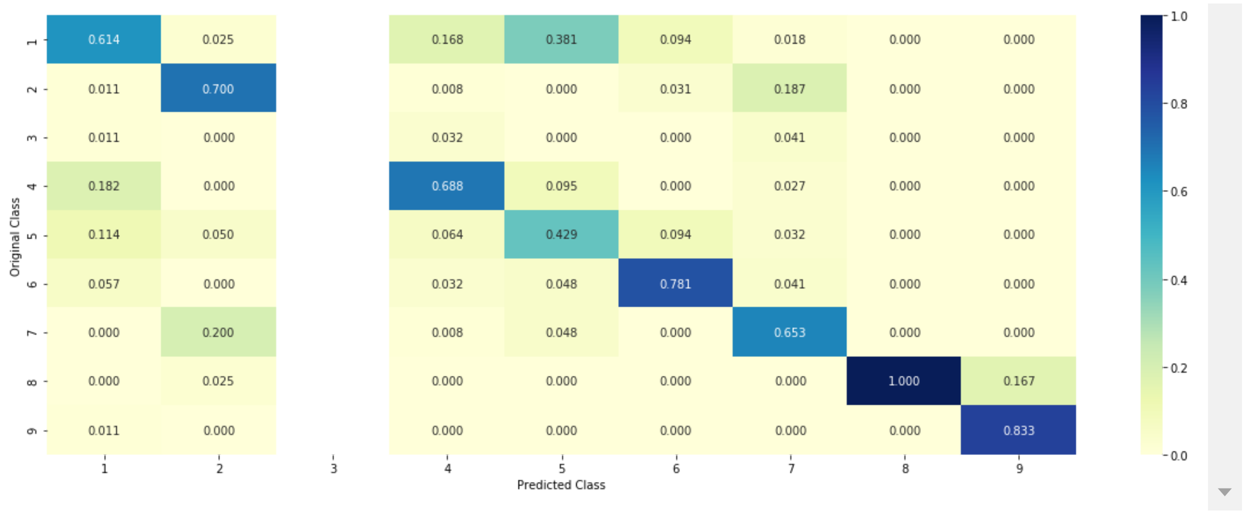
In [252]:
```
# read more about support vector machines with linear kernals here http://scikit

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

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


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

```
Log loss : 1.0193858139473322
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```

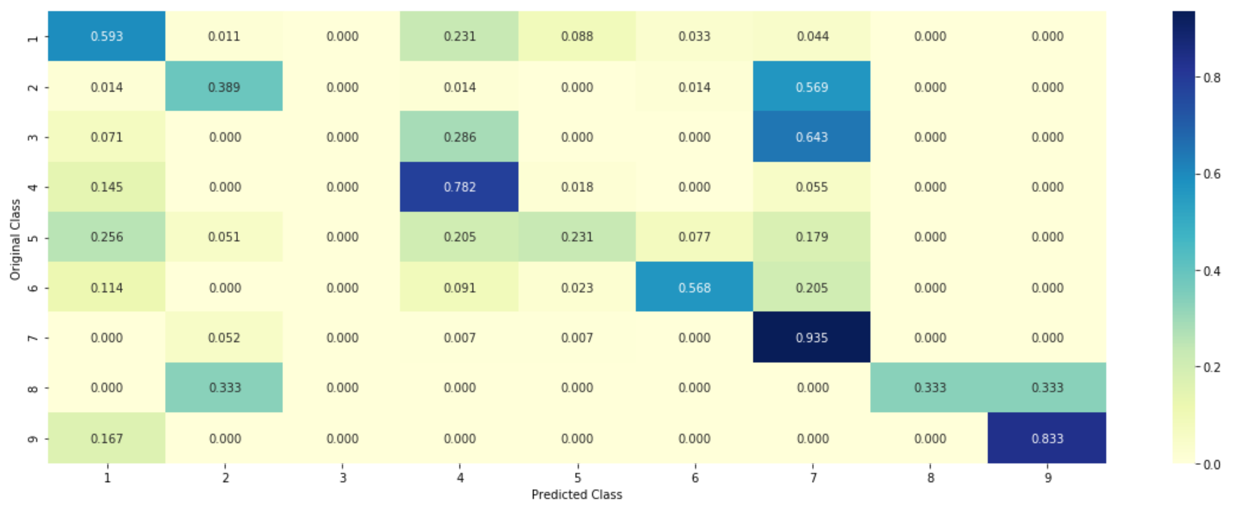


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	54.000	1.000	0.000	21.000	8.000	3.000	4.000	0.000	0.000
2	1.000	28.000	0.000	1.000	0.000	1.000	41.000	0.000	0.000
3	1.000	0.000	0.000	4.000	0.000	0.000	9.000	0.000	0.000
4	16.000	0.000	0.000	86.000	2.000	0.000	6.000	0.000	0.000
5	10.000	2.000	0.000	8.000	9.000	3.000	7.000	0.000	0.000
6	5.000	0.000	0.000	4.000	1.000	25.000	9.000	0.000	0.000
7	0.000	8.000	0.000	1.000	1.000	0.000	143.000	0.000	0.000
8	0.000	1.000	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	5.000

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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [253]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_
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_on
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[
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0418 0.7849 0.0179 0.0223 0.0472 0.021  0.05
17 0.0076 0.0057]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [254]:
```
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_on
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[
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0299 0.144  0.0357 0.0376 0.0621 0.0041 0.68
01 0.0037 0.0028]]
Actual Class : 2
--------------------------------------------------
313 Text feature [10] present in test data point [True]
464 Text feature [11] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

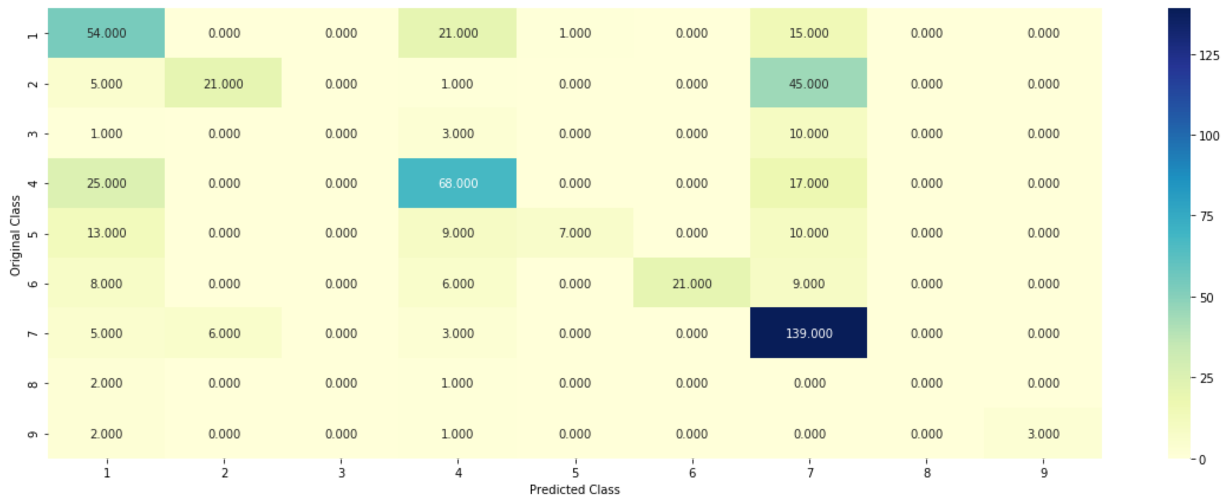
In [255]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_lea
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
# class_weight=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
#
# 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
        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.class
        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_
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='g
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 lo
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross va
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2123898639923436
for n_estimators = 100 and max depth =  10
Log Loss : 1.2079599488392754
for n_estimators = 200 and max depth =  5
Log Loss : 1.198475205032409
for n_estimators = 200 and max depth =  10
Log Loss : 1.2010852612150646
for n_estimators = 500 and max depth =  5
Log Loss : 1.1872142079564816
for n_estimators = 500 and max depth =  10
Log Loss : 1.196300214483638
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1841425171896085
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1919204507234764
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1837999271630917
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1905664396534934
For values of best estimator =  2000 The train log loss is: 0.8939627894440503
For values of best estimator =  2000 The cross validation log loss is: 1.183799
9271630917
For values of best estimator =  2000 The test log loss is: 1.2623306496209639
```

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

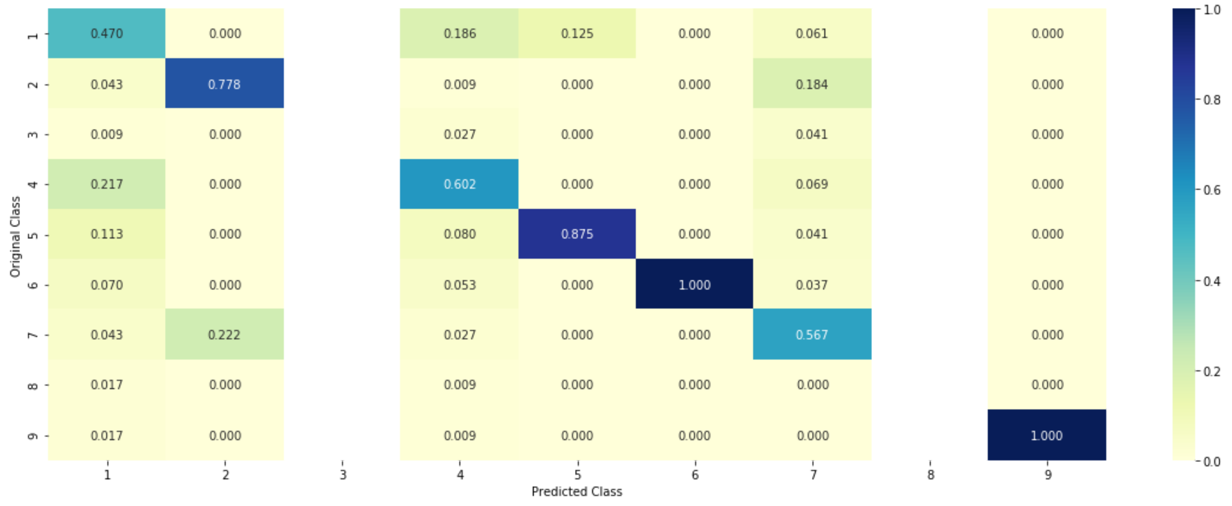
In [256]:
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_lea
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
# class_weight=None)

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='g
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCodin
```

```
Log loss : 1.1837999271630917
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```

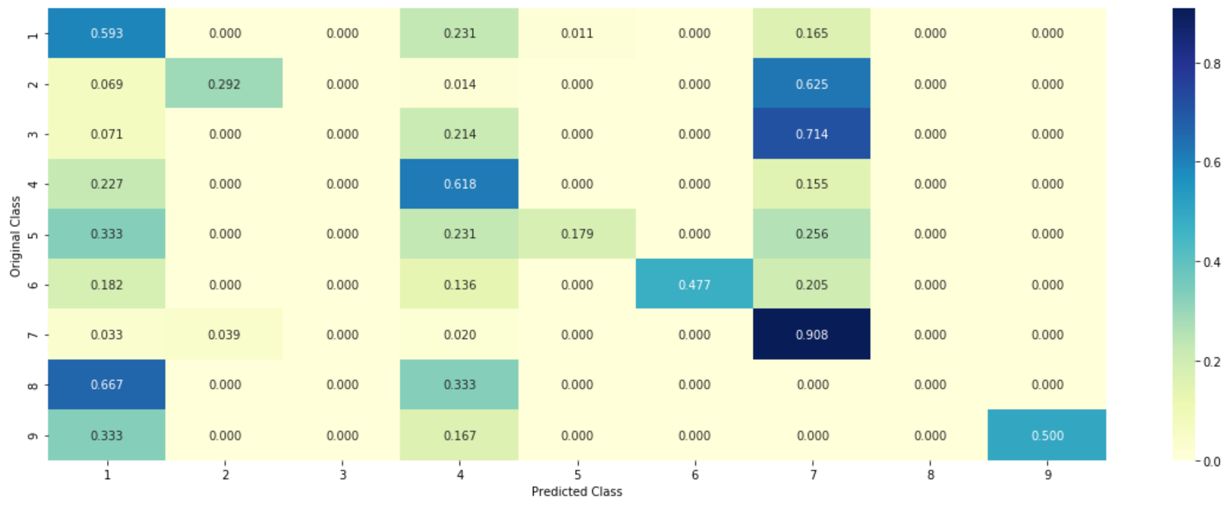


	1	2	3	4	5	6	7	8	9
1	54.000	0.000	0.000	21.000	1.000	0.000	15.000	0.000	0.000
2	5.000	21.000	0.000	1.000	0.000	0.000	45.000	0.000	0.000
3	1.000	0.000	0.000	3.000	0.000	0.000	10.000	0.000	0.000
4	25.000	0.000	0.000	68.000	0.000	0.000	17.000	0.000	0.000
5	13.000	0.000	0.000	9.000	7.000	0.000	10.000	0.000	0.000
6	8.000	0.000	0.000	6.000	0.000	21.000	9.000	0.000	0.000
7	5.000	6.000	0.000	3.000	0.000	0.000	139.000	0.000	0.000
8	2.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
9	2.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	3.000

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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [257]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='g
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_on
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
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0492 0.3968 0.0301 0.0857 0.0569 0.042  0.32
61 0.009  0.0041]]
Actual Class : 2
--------------------------------------------------
12 Text feature [109] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [258]:
```
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_on
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
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0138 0.5763 0.0149 0.0202 0.0332 0.026  0.30
98 0.0028 0.0029]]
Actuall Class : 2
--------------------------------------------------
42 Text feature [104] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [259]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_lea
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
#
# 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
        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.classe
        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_l
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='g
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 lo
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross valida
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log los
```

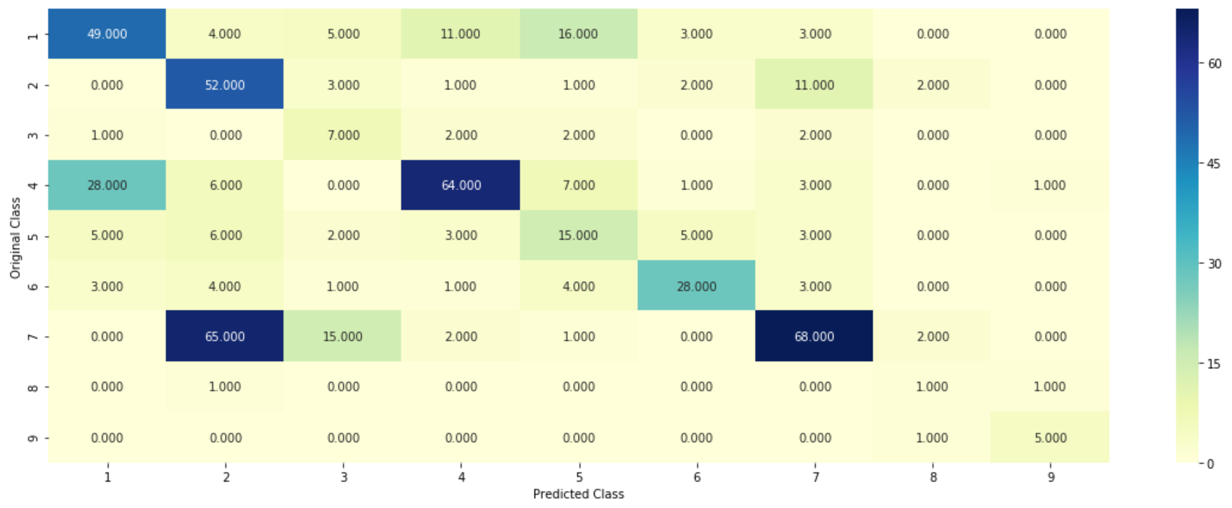
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.086875645592438
for n_estimators = 10 and max depth =  3
Log Loss : 1.657743511320897
for n_estimators = 10 and max depth =  5
Log Loss : 1.5346621771119413
for n_estimators = 10 and max depth =  10
Log Loss : 1.7465073792602968
for n_estimators = 50 and max depth =  2
Log Loss : 1.6660379753142083
for n_estimators = 50 and max depth =  3
Log Loss : 1.4642083057356599
for n_estimators = 50 and max depth =  5
Log Loss : 1.33386874914904
for n_estimators = 50 and max depth =  10
Log Loss : 1.6129944300444556
for n_estimators = 100 and max depth =  2
Log Loss : 1.5067387343673515
for n_estimators = 100 and max depth =  3
Log Loss : 1.4431070930095704
for n_estimators = 100 and max depth =  5
Log Loss : 1.2990703795958765
for n_estimators = 100 and max depth =  10
Log Loss : 1.5681175089890185
for n_estimators = 200 and max depth =  2
Log Loss : 1.5458662668211995
for n_estimators = 200 and max depth =  3
Log Loss : 1.441997159468693
for n_estimators = 200 and max depth =  5
Log Loss : 1.3882248733842144
for n_estimators = 200 and max depth =  10
Log Loss : 1.6077148634769651
for n_estimators = 500 and max depth =  2
Log Loss : 1.5837982259371914
for n_estimators = 500 and max depth =  3
Log Loss : 1.4756831763741975
for n_estimators = 500 and max depth =  5
Log Loss : 1.418354940420824
```

```
for n_estimators = 500 and max depth =  10
Log Loss : 1.6487332065239972
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5641483252214516
for n_estimators = 1000 and max depth =  3
Log Loss : 1.47874883985345
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4049997063196757
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6377662246080875
For values of best alpha =  100 The train log loss is: 0.0660813739997366
For values of best alpha =  100 The cross validation log loss is: 1.299070379
5958758
For values of best alpha =  100 The test log loss is: 1.3516019428735186
```

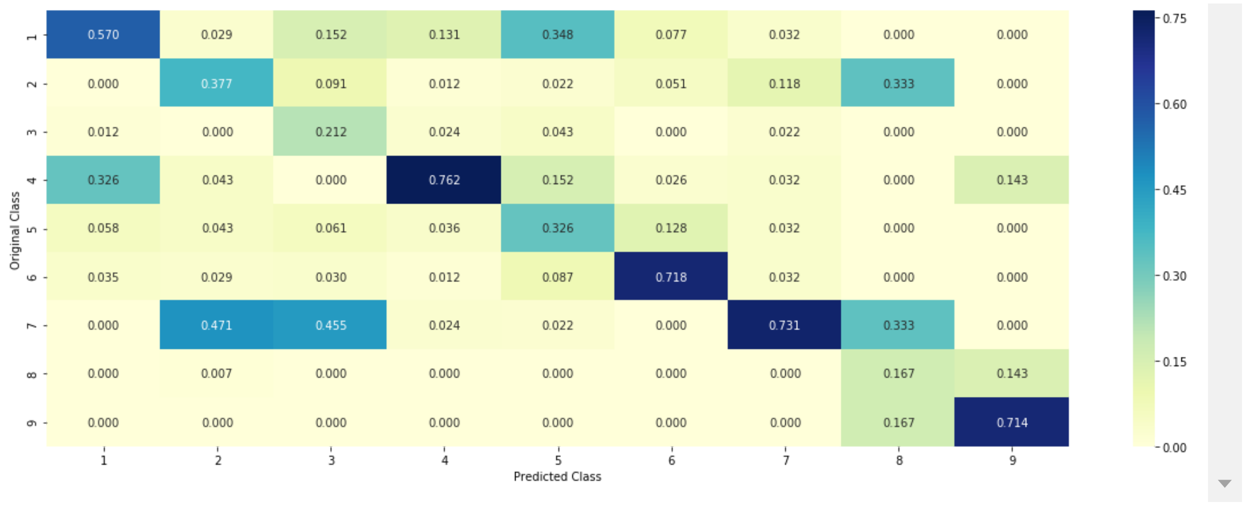
4.5.4. Testing model with best hyper parameters (Response Coding)

In [260]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_lea
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
# class_weight=None)

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimator
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseC
```

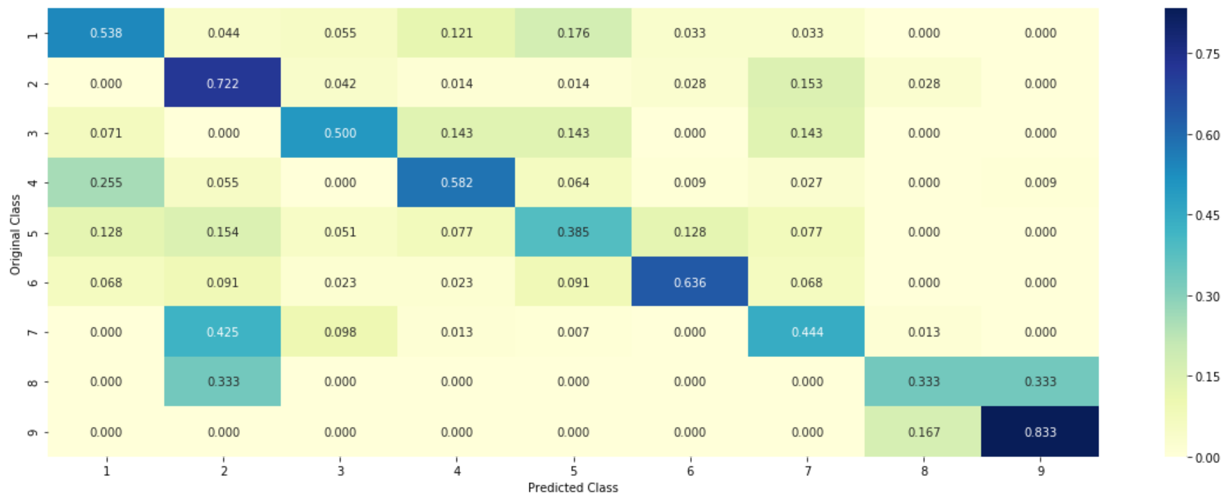
```
Log loss : 1.299070379595876
Number of mis-classified points : 0.4567669172932331
-------------------- Confusion matrix --------------------
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [261]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='g
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(
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_re
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0537 0.6091 0.0948 0.0424 0.0226 0.0391 0.04
49 0.0747 0.0188]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [262]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_res
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0135 0.3142 0.3264 0.0183 0.0231 0.0274 0.23
89 0.0275 0.0108]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [263]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_ra
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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



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

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



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

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



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

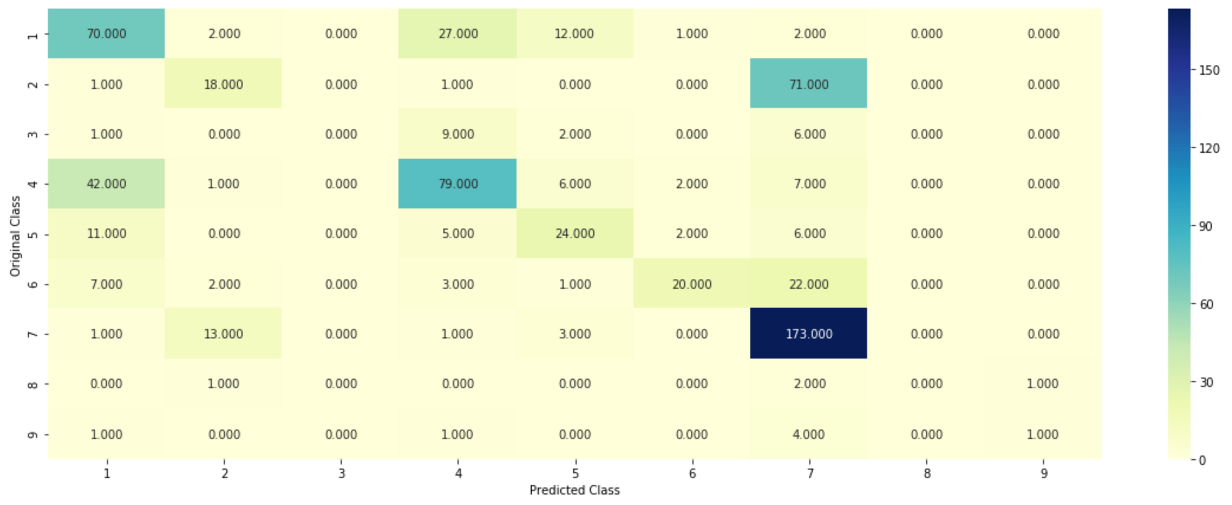
```
clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

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

```
Logistic Regression :  Log Loss: 1.06
Support vector machines : Log Loss: 1.77
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.714
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.344
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.289
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.656
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 2.036
```

4.7.2 testing the model with the best hyper parameters

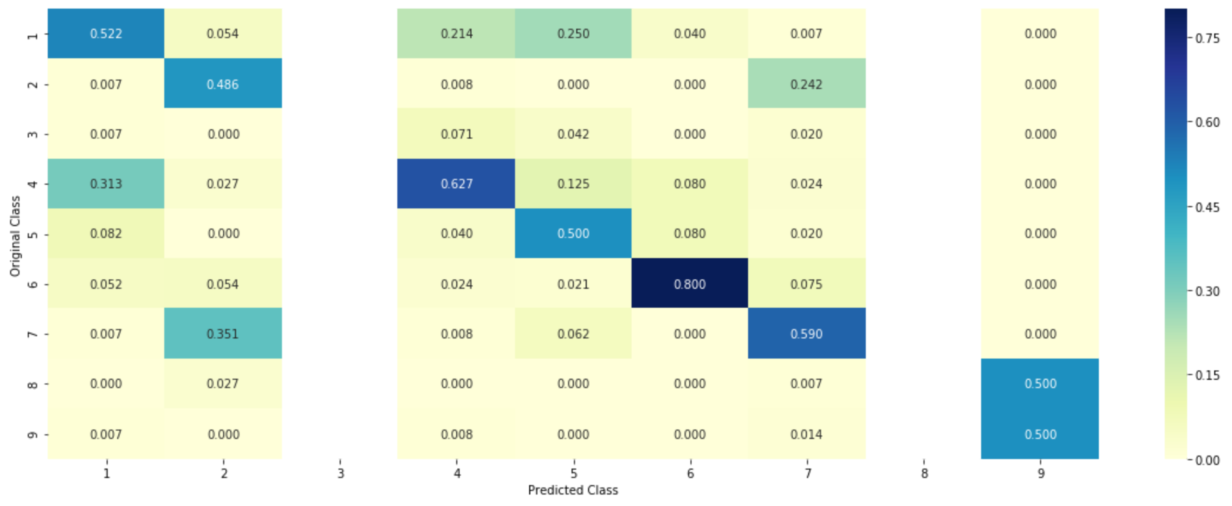
In [264]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_class
sclf.fit(train_x_onehotCoding, train_y)

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

```
Log loss (train) on the stacking classifier : 0.3329931046985514
Log loss (CV) on the stacking classifier : 1.2893915049964633
Log loss (test) on the stacking classifier : 1.3508757312562063
Number of missclassified point : 0.42105263157894735
-------------------- Confusion matrix --------------------
```



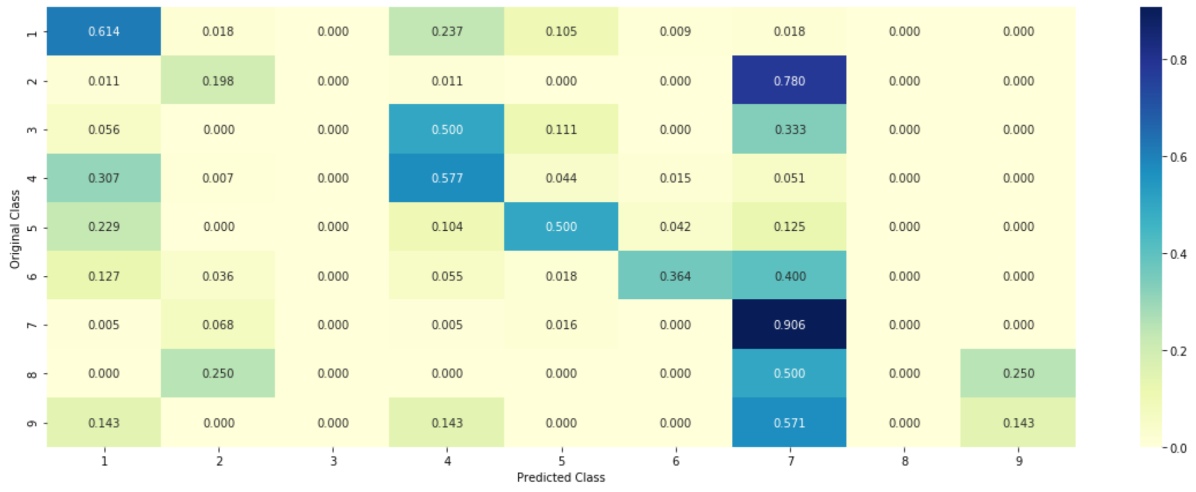
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	70.000	2.000	0.000	27.000	12.000	1.000	2.000	0.000	0.000
2	1.000	18.000	0.000	1.000	0.000	0.000	71.000	0.000	0.000
3	1.000	0.000	0.000	9.000	2.000	0.000	6.000	0.000	0.000
4	42.000	1.000	0.000	79.000	6.000	2.000	7.000	0.000	0.000
5	11.000	0.000	0.000	5.000	24.000	2.000	6.000	0.000	0.000
6	7.000	2.000	0.000	3.000	1.000	20.000	22.000	0.000	0.000
7	1.000	13.000	0.000	1.000	3.000	0.000	173.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000
9	1.000	0.000	0.000	1.000	0.000	0.000	4.000	0.000	1.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.522	0.054		0.214	0.250	0.040	0.007		0.000
2	0.007	0.486		0.008	0.000	0.000	0.242		0.000
3	0.007	0.000		0.071	0.042	0.000	0.020		0.000
4	0.313	0.027		0.627	0.125	0.080	0.024		0.000
5	0.082	0.000		0.040	0.500	0.080	0.020		0.000
6	0.052	0.054		0.024	0.021	0.800	0.075		0.000
7	0.007	0.351		0.008	0.062	0.000	0.590		0.000
8	0.000	0.027		0.000	0.000	0.000	0.007		0.500
9	0.007	0.000		0.008	0.000	0.000	0.014		0.500

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

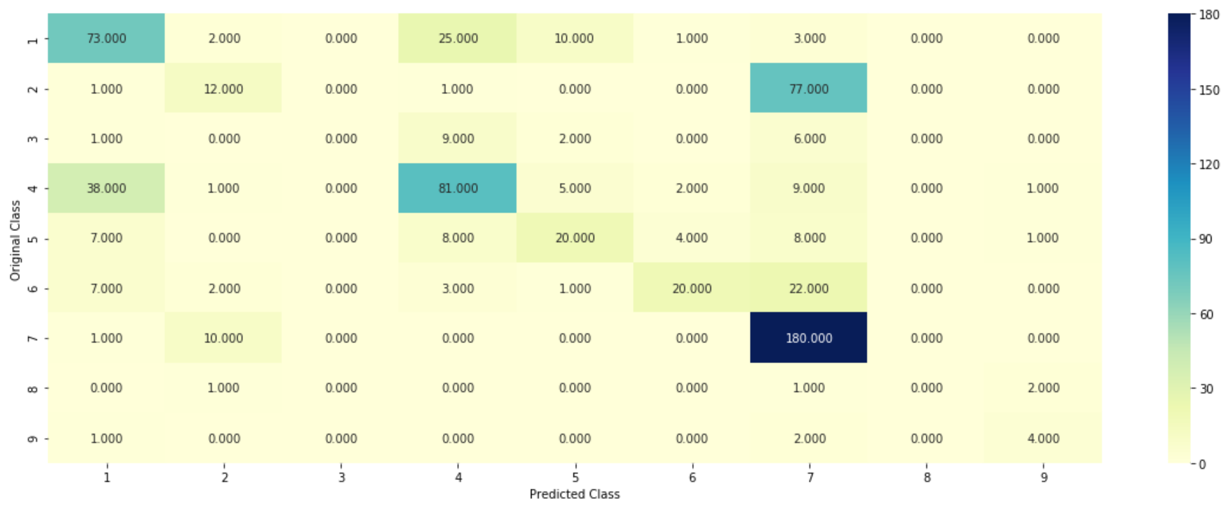


	1	2	3	4	5	6	7	8	9
1	0.614	0.018	0.000	0.237	0.105	0.009	0.018	0.000	0.000
2	0.011	0.198	0.000	0.011	0.000	0.000	0.780	0.000	0.000
3	0.056	0.000	0.000	0.500	0.111	0.000	0.333	0.000	0.000
4	0.307	0.007	0.000	0.577	0.044	0.015	0.051	0.000	0.000
5	0.229	0.000	0.000	0.104	0.500	0.042	0.125	0.000	0.000
6	0.127	0.036	0.000	0.055	0.018	0.364	0.400	0.000	0.000
7	0.005	0.068	0.000	0.005	0.016	0.000	0.906	0.000	0.000
8	0.000	0.250	0.000	0.000	0.000	0.000	0.500	0.000	0.250
9	0.143	0.000	0.000	0.143	0.000	0.000	0.571	0.000	0.143

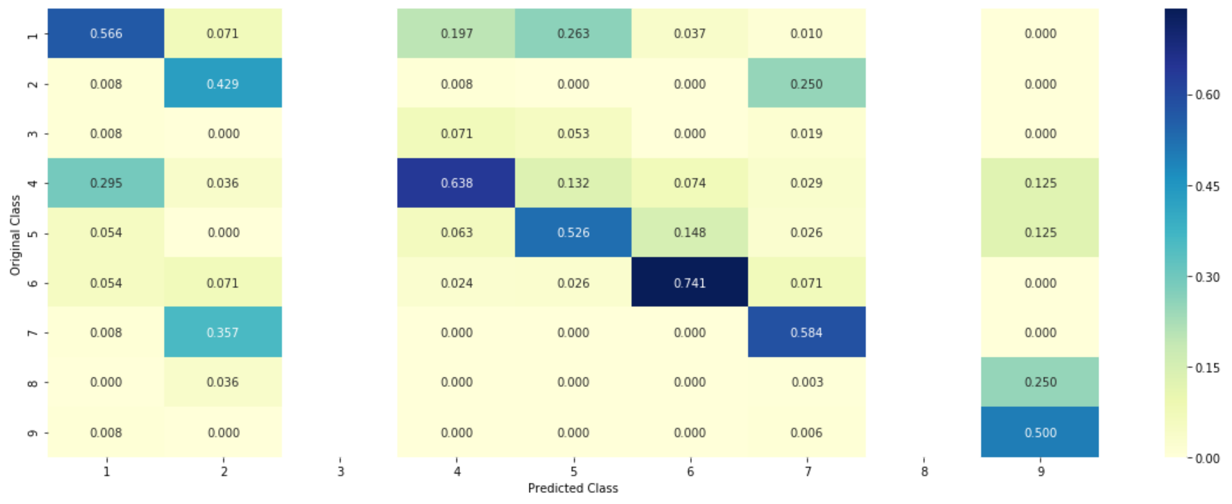
4.7.3 Maximum Voting classifier

```
In [265]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingCl
          from sklearn.ensemble import VotingClassifier
          vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', s
          vclf.fit(train_x_onehotCoding,train_y)
          print("Log loss (train) on the VotingClassifier:", log_loss(train_y, vclf.predict
          print("Log loss (CV) on the VotingClassifier:", log_loss(cv_y, vclf.predict_proba
          print("Log loss (test) on the VotingClassifier:", log_loss(test_y, vclf.predict_p
          print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_o
          plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding)
```

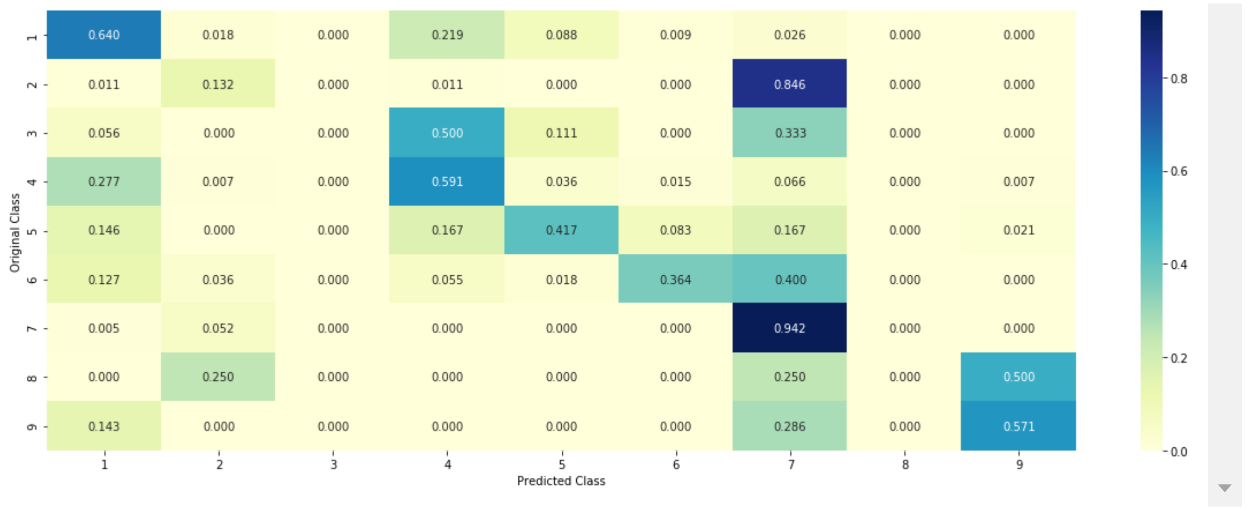
```
Log loss (train) on the VotingClassifier: 0.7916048327656435
Log loss (CV) on the VotingClassifier: 1.2176324976602917
Log loss (test) on the VotingClassifier: 1.2826968479830854
Number of missclassified point : 0.41353383458646614
-------------------- Confusion matrix --------------------
```



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



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



5. Assignments

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

WE WILL BE TRAINING 2 MODELS OF LOGISTIC REGRESSION WITHout BALANCED CLASS WEIGHT

1.MODEL TRAINED WITH FEATURES WITHOUT PERFORMING ANY FEATURE ENGINEERING

2.MODEL TRAINED WITH FEATURES BY PERFORMING FEATURE ENGINEERING TECHNIQUES

2.1-TEXT_DATA-TFIDF VECTORIZATION

2.2-MEAN ENCODING FOR OF GENE AND VARIATION features

In [0]:
```
train_df.columns
var_train=train_df['Variation'].values;
var_test=test_df['Variation'].values;
var_cv=cv_df['Variation'].values

gene_train=train_df['Gene'].values;
gene_test=test_df['Gene'].values;
gene_cv=cv_df['Gene'].values

text_train=train_df['TEXT'].values;
text_test=test_df['TEXT'].values;
text_cv=cv_df['TEXT'].values
```

In [0]:
```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer=CountVectorizer(min_df=10,ngram_range=(1, 2))
var_train=vectorizer.fit_transform(var_train)
var_test=vectorizer.transform(var_test)
var_cv=vectorizer.transform(var_cv)

gene_train=vectorizer.fit_transform(gene_train)
gene_test=vectorizer.transform(gene_test)
gene_cv=vectorizer.transform(gene_cv)


text_train=vectorizer.fit_transform(text_train)
text_test=vectorizer.transform(text_test)
text_cv=vectorizer.transform(text_cv)
```

In [0]:
```
from scipy.sparse import hstack
data_train=hstack([var_train,gene_train,text_train]).tocsr()
data_test=hstack([var_test,gene_test,text_test]).tocsr()
data_cv=hstack([var_cv,gene_cv,text_cv]).tocsr()
```

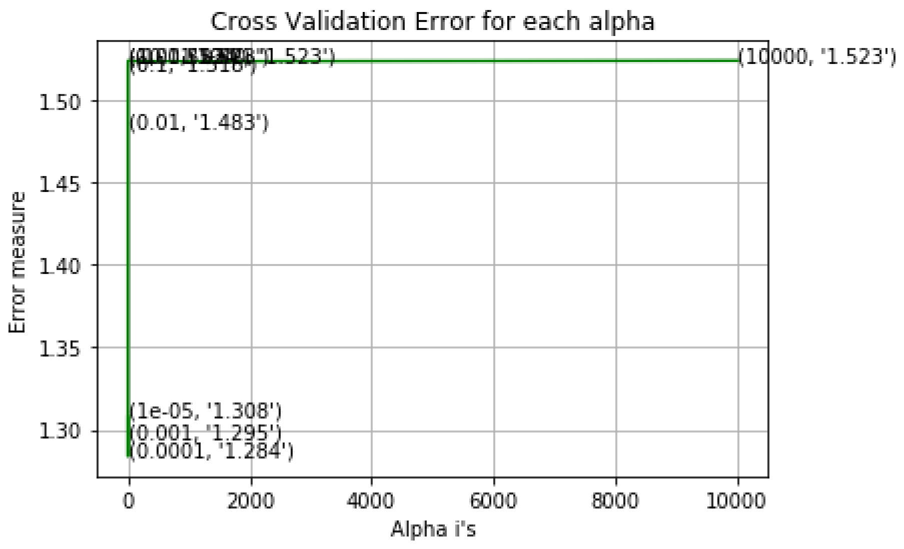
In [19]:
```
print(data_train.shape)
print(data_test.shape)
print(data_cv.shape)
```

```
(2124, 235934)
(665, 235934)
(532, 235934)
```

In [172]:
```
cv_values=[]
alpha=[10 ** x for x in range(-5, 5)]
for c in tqdm(alpha):
  print("for alpha =", c)
  lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_jobs=-1)
  clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
  clf.fit(data_train,y_train)
  pre_cv=clf.predict_proba(data_cv)
  print(c,log_loss(y_cv, pre_cv))
  cv_values.append(log_loss(y_cv, pre_cv))
print(alpha,cv_values)
print(len(alpha),len(cv_values))
print(type(cv_values))
fig, ax = plt.subplots()
ax.plot(alpha, cv_values,c='g')
for i, t in enumerate(np.round(cv_values,3)):
  ax.annotate((alpha[i],str(t)), (alpha[i],cv_values[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

for alpha = 1e-05

 10%|█         | 1/10 [04:18<38:46, 258.53s/it]

1e-05 1.307704760189509
for alpha = 0.0001

 20%|██        | 2/10 [08:33<34:19, 257.42s/it]

0.0001 1.2837278442535625
for alpha = 0.001

 30%|███       | 3/10 [12:43<29:46, 255.15s/it]

0.001 1.2948793444371294
for alpha = 0.01

 40%|████      | 4/10 [16:50<25:16, 252.72s/it]

0.01 1.482630497950825
for alpha = 0.1

 50%|█████     | 5/10 [20:54<20:50, 250.06s/it]

0.1 1.5182725411359548
for alpha = 1
```

```
 60%|██████    | 6/10 [24:58<16:33, 248.46s/it]

1 1.5230694403732323
for alpha = 10


 70%|███████   | 7/10 [29:06<12:24, 248.14s/it]

10 1.5235605057949146
for alpha = 100


 80%|████████  | 8/10 [33:12<08:14, 247.45s/it]

100 1.5231432546278938
for alpha = 1000


 90%|█████████ | 9/10 [37:16<04:06, 246.55s/it]

1000 1.5230428598845918
for alpha = 10000


100%|██████████| 10/10 [41:20<00:00, 245.72s/it]

10000 1.523460007982243
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000] [1.307704760189509,
1.2837278442535625, 1.2948793444371294, 1.482630497950825, 1.5182725411359548,
1.5230694403732323, 1.5235605057949146, 1.5231432546278938, 1.5230428598845918,
1.523460007982243]
10 10
<class 'list'>
```



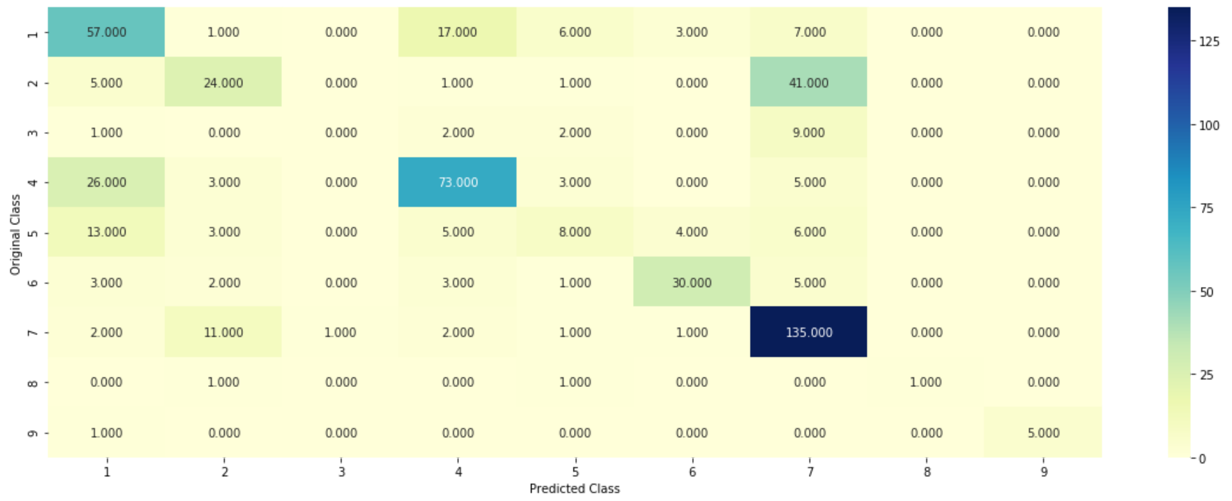
from above graph best value for C=0.001

In [271]:
```
l = LogisticRegression(random_state=0, C=0.001,class_weight='balanced',n_jobs=-1
clf=CalibratedClassifierCV(base_estimator=l,method='sigmoid')
clf.fit(data_train,y_train)
pre_test=clf.predict_proba(data_test)
print("Log Loss value for the test data is ",log_loss(y_test, pre_test))
```

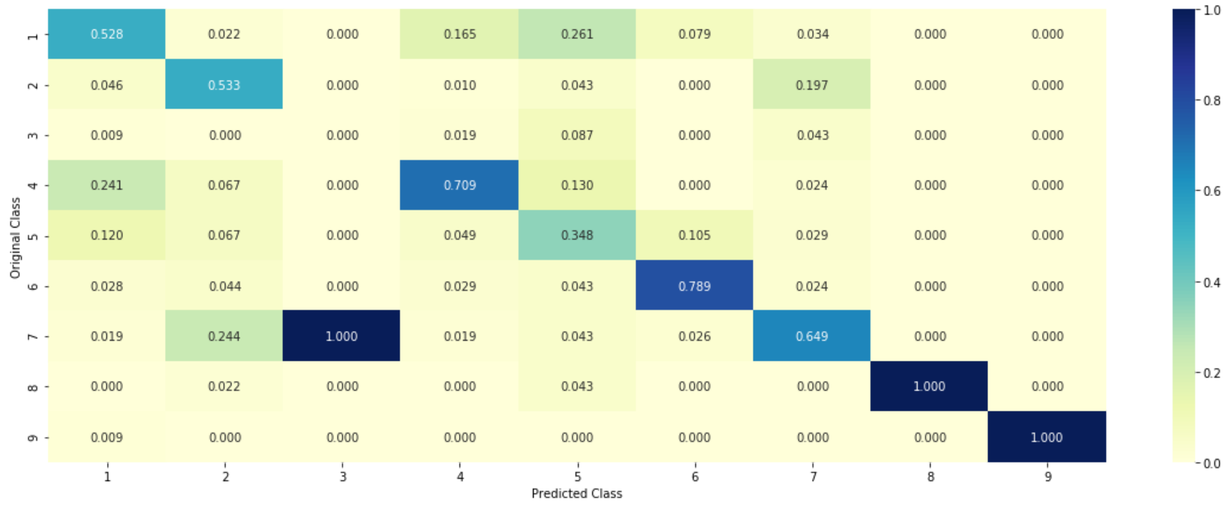
```
Log Loss value for the test data is  1.3722961595858407
```

```
In [274]: predict_and_plot_confusion_matrix(data_train, y_train,data_cv,y_cv, clf)
```

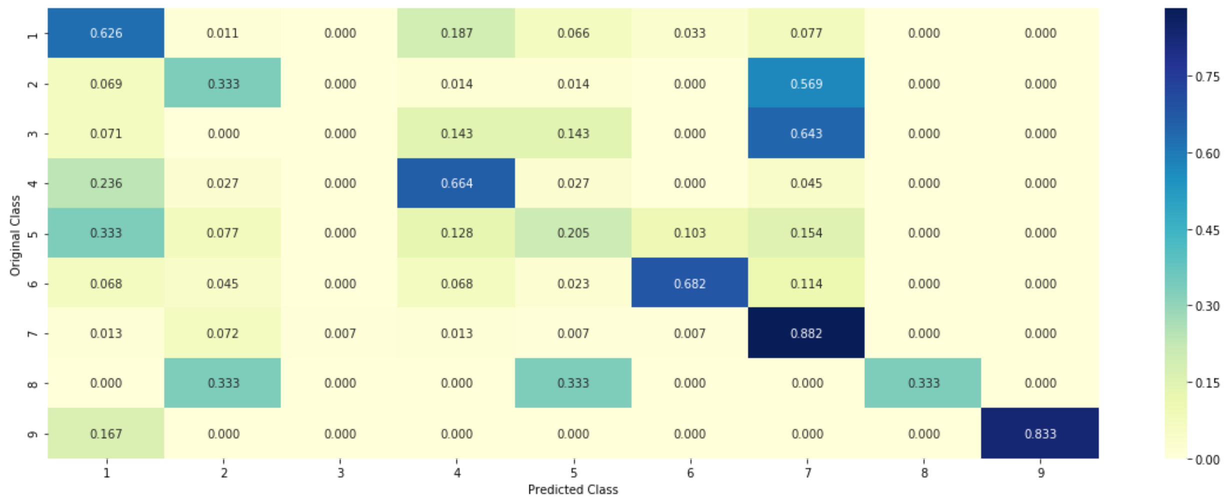
```
Log loss : 1.0824708168768304
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```



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



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



Now we will do feature engineering on all the 3 features so that we get a loss less than 1

We will select top 5000 features using idf values from text data and vectorize them using TFIDF vectorizer

In [15]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_text= TfidfVectorizer( min_df=10,max_features=5000)
vectorizer_tfidf_text.fit(train_df["TEXT"])
text_tfidf_train = vectorizer_tfidf_text.transform(train_df["TEXT"])
text_tfidf_test = vectorizer_tfidf_text.transform(test_df["TEXT"])
text_tfidf_cv = vectorizer_tfidf_text.transform(cv_df["TEXT"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (2124, 5000)
Shape of matrix after one hot encoding  (665, 5000)
Shape of matrix after one hot encoding  (532, 5000)
```

In [38]:
```
print(var_train.shape)
print(gene_train.shape)
print(text_train.shape)
print(var_test.shape)
print(gene_test.shape)
print(text_test.shape)
print(var_cv.shape)
print(gene_cv.shape)
print(text_cv.shape)
```

```
(2124, 5)
(2124, 55)
(2124, 222813)
(665, 5)
(665, 55)
(665, 222813)
(532, 5)
(532, 55)
(532, 222813)
```

In [0]:
```
vargen_train=pd.DataFrame()
vargen_test=pd.DataFrame()
vargen_cv=pd.DataFrame()
```

We will combine 2 features in single column

In [0]:
```
vargen_train['var_gene']=train_df['Variation']+' '+train_df['Gene']
vargen_test['var_gene']=test_df['Variation']+' '+test_df['Gene']
vargen_cv['var_gene']=cv_df['Variation']+' '+cv_df['Gene']
```

In [0]:
```

vargene_train=pd.DataFrame()
vargene_test=pd.DataFrame()
vargene_cv=pd.DataFrame()
```

In [0]:
```
var_n_train=pd.DataFrame()
var_n_test=pd.DataFrame()
var_n_cv=pd.DataFrame()
```

Feature engineering steps:

1.We will count length for the combined var_gene feature

2..We will select top 10 words from variation and gene andperform mean encoding i.e check if word is present in the row,if the word is is present 1 is appended else 0 is appended.

3.TDFIDF Vectorization for text and select 5000 best words

mean encoding Variation feature

In [0]:
```
top_10_var=[ x for x in train_df['Variation'].value_counts().sort_values(ascendi
for i in top_10_var:
  var_n_train[i]=np.where(train_df['Variation']==i,1,0)
  var_n_test[i]=np.where(test_df['Variation']==i,1,0)
  var_n_cv[i]=np.where(cv_df['Variation']==i,1,0)
```

In [0]:
```
gene_n_train=pd.DataFrame()
gene_n_test=pd.DataFrame()
gene_n_cv=pd.DataFrame()
```

Mean encoding Gene feature

In [0]:
```
top_10_gene=[ x for x in train_df['Gene'].value_counts().sort_values(ascending=F
for i in top_10_gene:
  gene_n_train[i]=np.where(train_df['Gene']==i,1,0)
  gene_n_test[i]=np.where(test_df['Gene']==i,1,0)
  gene_n_cv[i]=np.where(cv_df['Gene']==i,1,0)
```

In [0]:
```
text_n_train=pd.DataFrame()
text_n_test=pd.DataFrame()
text_n_cv=pd.DataFrame()
```

In [0]:
```
count=pd.DataFrame()
line=[]
cnt_train=[]
cnt_test=[]
cnt_cv=[]
cnt_text_train=[]
cnt_text_test=[]
cnt_text_cv=[]
```

counting number of words in Var_gene feature

In [53]:
```
for i in vargen_train['var_gene']:
  for j in i:
    line.append(j)
  cnt_train.append(len(line))
for i in vargen_test['var_gene']:
  for j in i:
    line.append(j)
  cnt_test.append(len(line))
for i in vargen_cv['var_gene']:
  for j in i:
    line.append(j)
  cnt_cv.append(len(line))
cnt_train=pd.DataFrame(cnt_train)
cnt_test=pd.DataFrame(cnt_test)
cnt_cv=pd.DataFrame(cnt_cv)
print(cnt_train.shape)
print(cnt_test.shape)
print(cnt_cv.shape)
```

```
(2124, 1)
(665, 1)
(532, 1)
```

Normalizing length of text

In [54]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(cnt_train.values.reshape(-1,1))
cnt_train = normalizer.transform(cnt_train.values.reshape(-1,1))
cnt_cv = normalizer.transform(cnt_cv.values.reshape(-1,1))
cnt_test = normalizer.transform(cnt_test.values.reshape(-1,1))
print("After vectorizations")
print(cnt_train.shape, y_train.shape)
print(cnt_cv.shape, y_cv.shape)
print(cnt_test.shape, y_test.shape)
```

```
After vectorizations
(2124, 1) (2124,)
(532, 1) (532,)
(665, 1) (665,)
```

Stacking the feature engineered features

In [0]:
```
data_train_fe=hstack([var_train,gene_train,text_tfidf_train,var_n_train,gene_n_tr
data_test_fe=hstack([var_test,gene_test,text_tfidf_test,var_n_test,gene_n_test,cn
data_cv_fe=hstack([var_cv,gene_cv,text_tfidf_cv,var_n_cv,gene_n_cv,cnt_cv]).tocsr
```

In [57]:
```
print(data_train_fe.shape)
print(data_test_fe.shape)
print(data_cv_fe.shape)
```

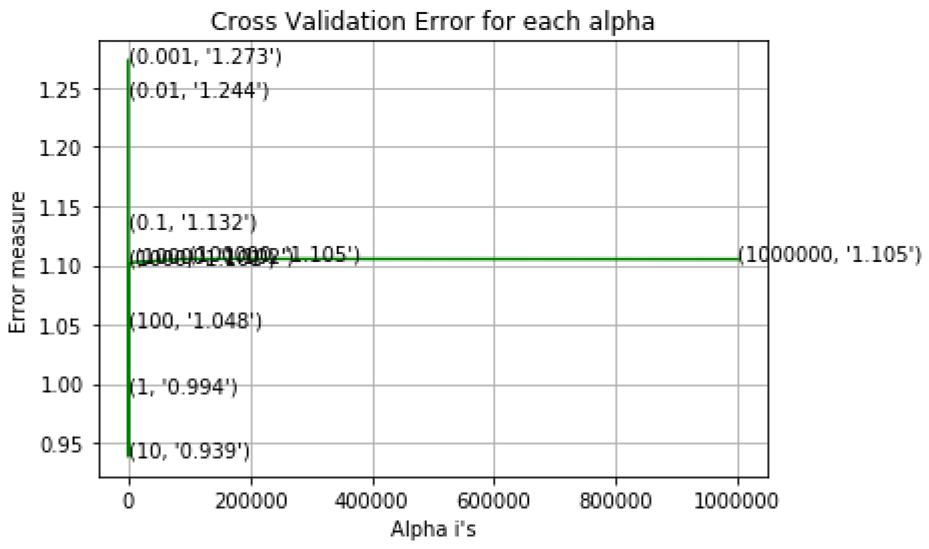
```
(2124, 5081)
(665, 5081)
(532, 5081)
```

Hyperparameter tuning using the CV_data

In [58]:
```python
from tqdm import tqdm
cv_values=[]
alpha=[10 ** x for x in range(-3, 7)]
for c in tqdm(alpha):
  print("for alpha =", c)
  lr = LogisticRegression(random_state=0, C=c,n_jobs=-1)
  clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
  clf.fit(data_train_fe,y_train)
  pre_cv=clf.predict_proba(data_cv_fe)
  print(c,log_loss(y_cv, pre_cv))
  cv_values.append(log_loss(y_cv, pre_cv))
print(alpha,cv_values)
print(len(alpha),len(cv_values))
print(type(cv_values))
fig, ax = plt.subplots()
ax.plot(alpha, cv_values,c='g')
for i, t in enumerate(np.round(cv_values,3)):
  ax.annotate((alpha[i],str(t)), (alpha[i],cv_values[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

for alpha = 0.001

 10%|█         | 1/10 [00:08<01:18,  8.69s/it]

0.001 1.27309174912667
for alpha = 0.01

 20%|██        | 2/10 [00:15<01:05,  8.22s/it]

0.01 1.244138503182963
for alpha = 0.1

 30%|███       | 3/10 [00:28<01:05,  9.42s/it]

0.1 1.1315744489504467
for alpha = 1

 40%|████      | 4/10 [00:49<01:17, 12.99s/it]

1 0.9937219276589544
for alpha = 10

 50%|█████     | 5/10 [01:10<01:17, 15.41s/it]

10 0.9385640085921434
for alpha = 100

 60%|██████    | 6/10 [01:31<01:08, 17.02s/it]

100 1.0482154337714413
for alpha = 1000

 70%|███████   | 7/10 [01:52<00:55, 18.33s/it]
```

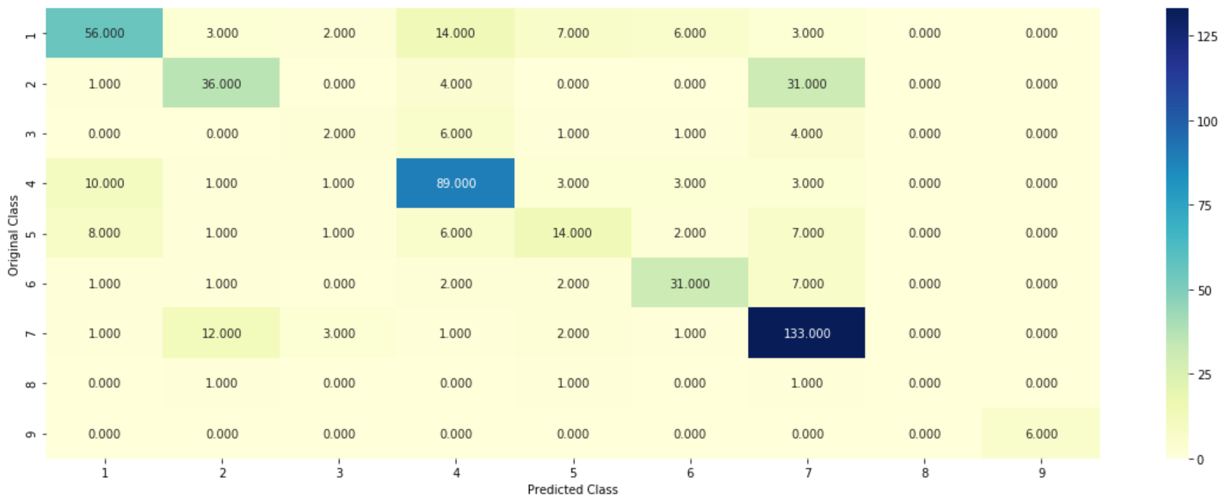
```
1000 1.1010144397944943
for alpha = 10000

 80%|████████  | 8/10 [02:13<00:38, 19.15s/it]

10000 1.102444900198064
for alpha = 100000

 90%|█████████ | 9/10 [02:34<00:19, 19.65s/it]

100000 1.1052350566275917
for alpha = 1000000

100%|██████████| 10/10 [02:55<00:00, 20.09s/it]

1000000 1.104914598584836
[0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000] [1.27309174912667,
1.244138503182963, 1.1315744489504467, 0.9937219276589544, 0.9385640085921434,
1.0482154337714413, 1.1010144397944943, 1.102444900198064, 1.1052350566275917,
1.104914598584836]
10 10
<class 'list'>
```



Using the best hyperparameter c=10

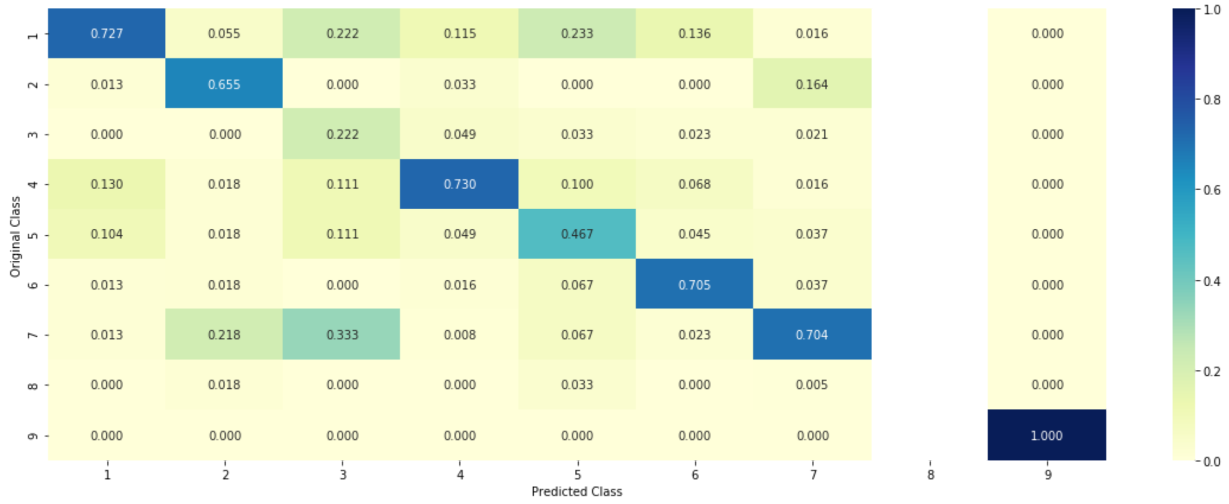
In [64]:
```
l = LogisticRegression(random_state=0, C=10,n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=l,method='sigmoid')
clf.fit(data_train_fe,y_train)
pre_test=clf.predict_proba(data_test_fe)
print("Log Loss value for the test data is ",log_loss(y_test, pre_test))
```

```
Log Loss value for the test data is  0.9887221971467367
Log loss : 0.9088807259280711
Number of mis-classified points : 0.3101503759398496
-------------------- Confusion matrix --------------------
```

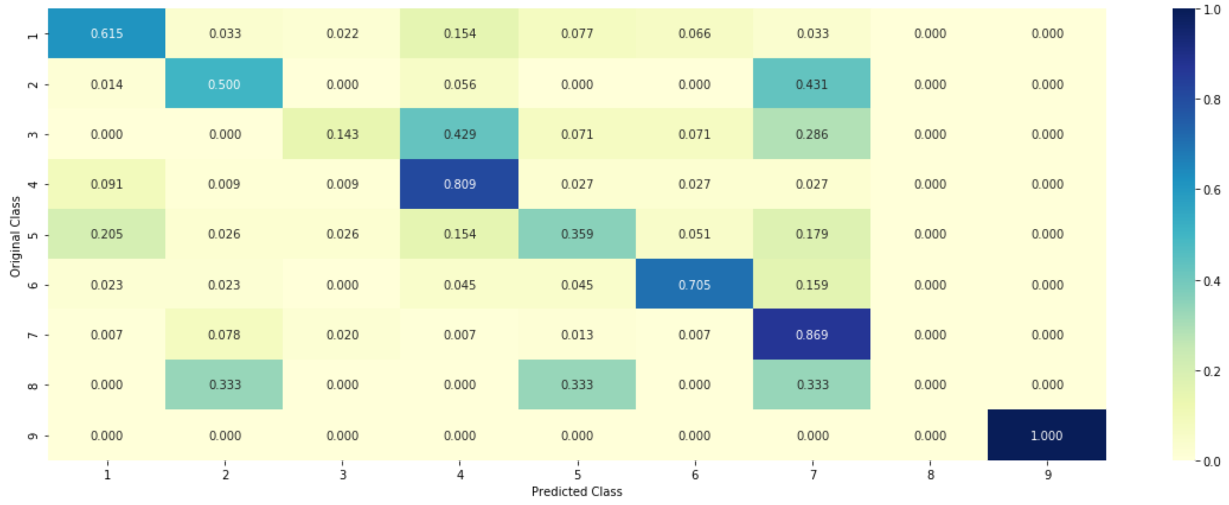


	1	2	3	4	5	6	7	8	9
1	56.000	3.000	2.000	14.000	7.000	6.000	3.000	0.000	0.000
2	1.000	36.000	0.000	4.000	0.000	0.000	31.000	0.000	0.000
3	0.000	0.000	2.000	6.000	1.000	1.000	4.000	0.000	0.000
4	10.000	1.000	1.000	89.000	3.000	3.000	3.000	0.000	0.000
5	8.000	1.000	1.000	6.000	14.000	2.000	7.000	0.000	0.000
6	1.000	1.000	0.000	2.000	2.000	31.000	7.000	0.000	0.000
7	1.000	12.000	3.000	1.000	2.000	1.000	133.000	0.000	0.000
8	0.000	1.000	0.000	0.000	1.000	0.000	1.000	0.000	0.000
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) --------------------
```



In [69]:
```
# Names of models
from prettytable import PrettyTable
model=['Naive Bayes ','KNN','Logistic Regression With Class balancing ','Logisti

train =[0.4446,0.5723,0.3820,0.3766,0.3066, 0.8939,0.3323,0.7116,0.6608,0.3762,0
test = [1.2911,1.1678,1.0949,1.0852,1.1182,1.2623,1.3509,1.2826,1.3516,1.3723,0.
cv=[1.2405,1.0429,0.9949,0.9938,1.019,1.1837,1.2894,1.2176,1.2991,1.0824,0.9385]
mp=[40.9,34.58,34.77,35.34,34,41.17,42.11,41.53,45.67,37.406,31.01]
numbering=[1,2,3,4,5,6,7,8,9,10,11]
p = PrettyTable()
p.add_column("S.NO.",numbering)
p.add_column("model",model)
p.add_column("train",train)
p.add_column("test",test)
p.add_column("cv",cv)
p.add_column("% Misclassified Points",mp)
print(p)
```

```
+-------+------------------------------------------------------------+------
--+--------+--------+-----------------------+
| S.NO. |                           model                            | train
|  test  |   cv   | % Misclassified Points |
+-------+------------------------------------------------------------+------
--+--------+--------+-----------------------+
|   1   |                        Naive Bayes                         | 0.444
6 | 1.2911 | 1.2405 |          40.9          |
|   2   |                            KNN                             | 0.572
3 | 1.1678 | 1.0429 |         34.58          |
|   3   |          Logistic Regression With Class balancing          | 0.382
| 1.0949 | 0.9949 |         34.77          |
|   4   |         Logistic Regression Without Class balancing        | 0.376
6 | 1.0852 | 0.9938 |         35.34          |
|   5   |                         Linear SVM                         | 0.306
6 | 1.1182 | 1.019  |           34           |
|   6   |        Random Forest Classifier With One hot Encoding      | 0.893
9 | 1.2623 | 1.1837 |         41.17          |
|   7   |        Random Forest Classifier With Response Coding       | 0.332
3 | 1.3509 | 1.2894 |         42.11          |
|   8   |                   Stack Models:LR+NB+SVM                   | 0.711
6 | 1.2826 | 1.2176 |         41.53          |
|   9   |                 Maximum Voting classifier                  | 0.660
8 | 1.3516 | 1.2991 |         45.67          |
|   10  | CountVectorizer Features, including both unigrams and bigrams | 0.376
2 | 1.3723 | 1.0824 |         37.406         |
|   11  |                 after feature engineering                  | 0.908
8 | 0.9887 | 0.9385 |         31.01          |
+-------+------------------------------------------------------------+------
--+--------+--------+-----------------------+
```

Conclusion:

After performing feature engineering the loss for train test and cv are below 1.