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 classified into => Multi class classification problem
```

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import 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
```

```
/home/chitts/anaconda3/envs/datascience_devenv/lib/python3.7/site-pack
ages/sklearn/utils/deprecation.py:143: FutureWarning: The sklearn.metr
ics.classification module is  deprecated in version 0.22 and will be r
emoved in version 0.24. The corresponding classes / functions should i
nstead be imported from sklearn.metrics. Anything that cannot be impor
ted from sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="pytho
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time
```

```
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 : 22.584403 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

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

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result[
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of
X_train, test_df, y_train, y_test = train_test_split(result, y_true, str
# split the train data into train and cross validation by maintaining sa
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stra
```

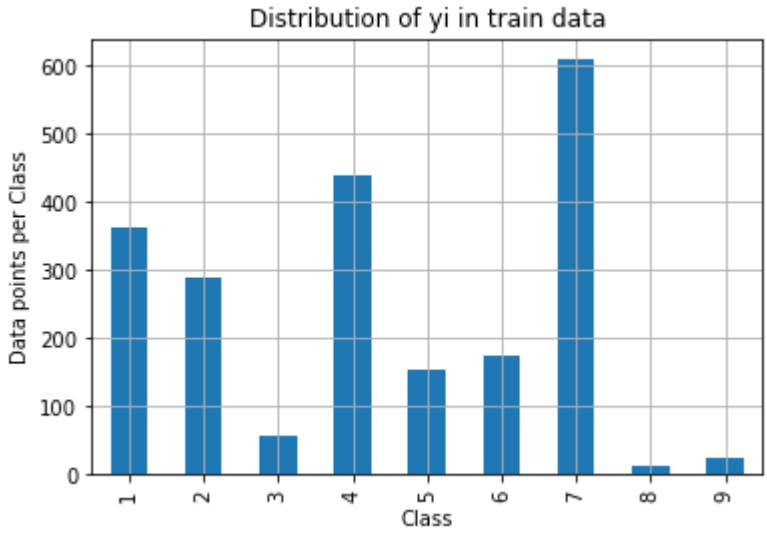
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

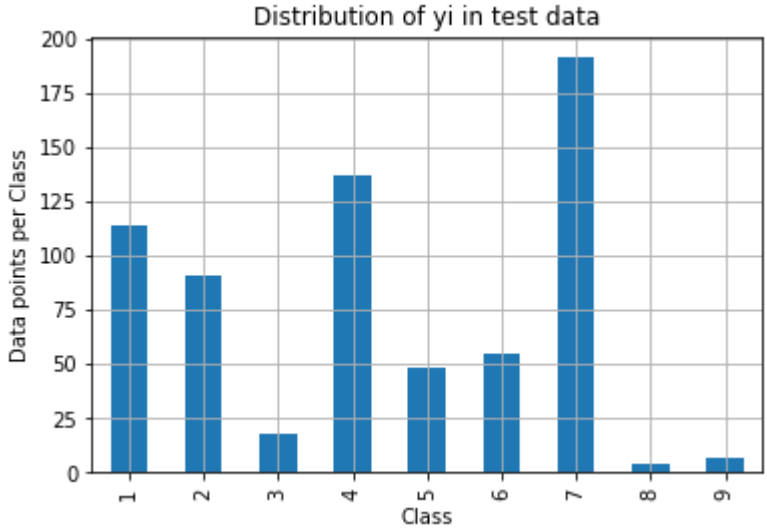
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

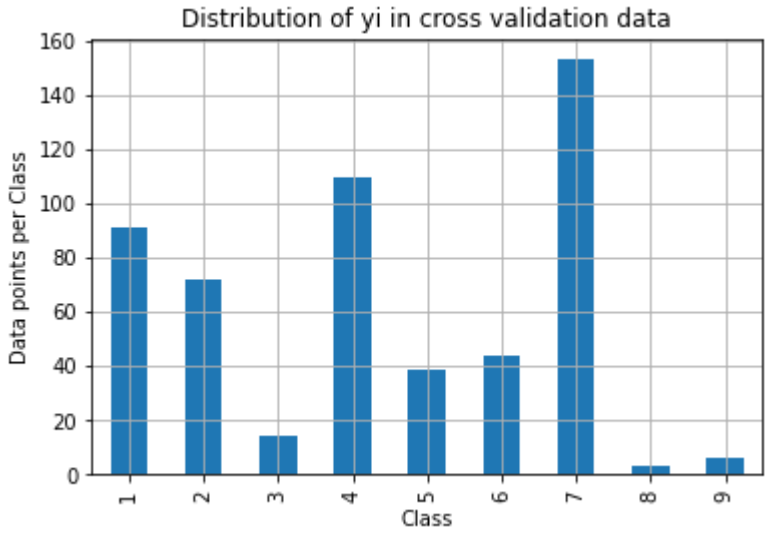
In [12]:
```
# it returns a dict, keys as class labels and values as the number of da
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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/nump
# -(train_class_distribution.values): the minus sign will give us in dec
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distrib


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/nump
# -(train_class_distribution.values): the minus sign will give us in dec
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribu

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/nump
# -(train_class_distribution.values): the minus sign will give us in dec
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distributi
```




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




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




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

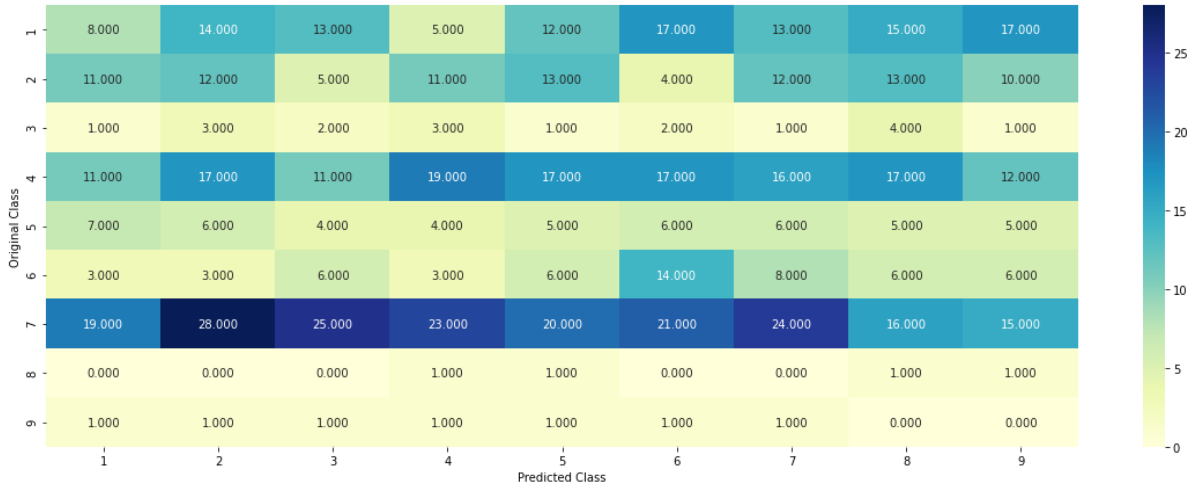
3.2 Prediction using a 'Random' Model

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

In [13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cla

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 correspor
    # 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
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspor
    # 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=lab
    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=lab
    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=lab
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

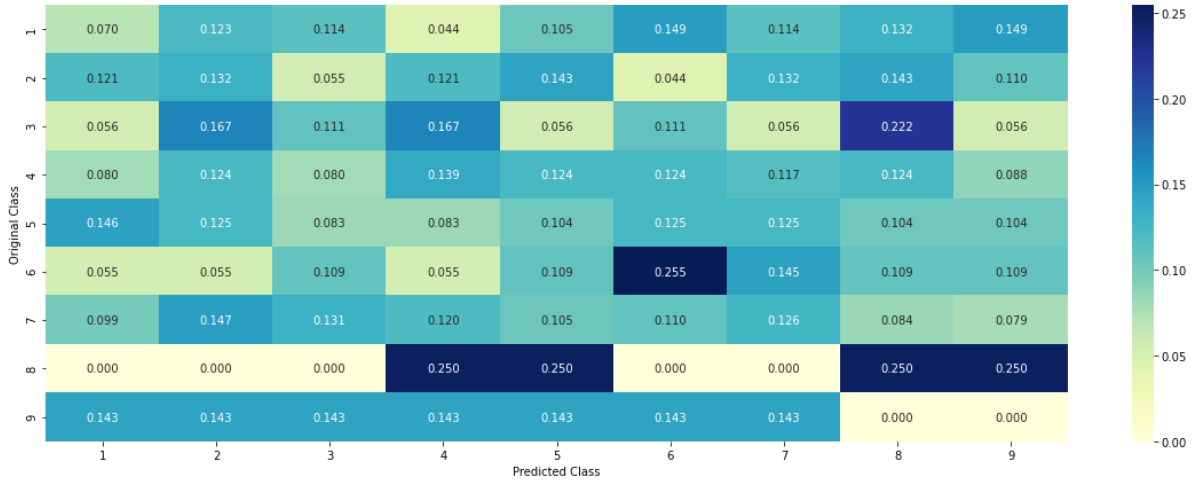
In [14]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers b
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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_


# 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_pr

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 2.433166682767864
4
Log loss on Test Data using Random Model 2.4576106329015204
-------------------- Confusion matrix --------------------
```




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

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



3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given featu
# build a vector (1*9) , the first element = (number of times it occured
# gv_dict is like a look up table, for every gene it store a (1*9) repre
# 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_fe
# 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 arra
    gv_dict = dict()

    # denominator will contain the number of time that particular featur
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation be
        # vec is 9 diamensional vector
        vec = []
```

```
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Ger
            #         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[fe

            # cls_cnt.shape[0](numerator) will contain the number of tim
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*

        # we are adding the gene/variation to the dict as key and vec as
        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, 0.068181
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
    #      ...
    #     }
    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 ea
    gv_fea = []
    # for every feature values in the given data frame we will check if
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- (numerator + 10*alpha) / (denominator + 90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

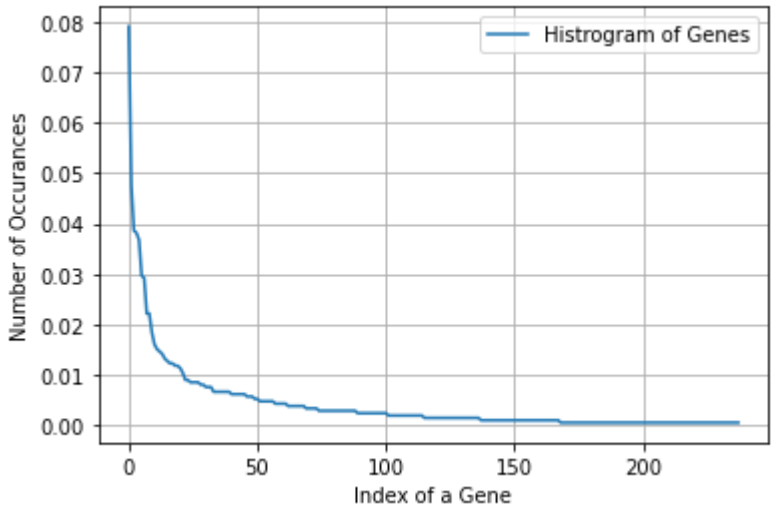
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 238
BRCA1     168
TP53      101
PTEN       82
BRCA2      81
EGFR       78
BRAF       63
KIT        62
ALK        47
ERBB2      47
PIK3CA     39
Name: Gene, dtype: int64
```

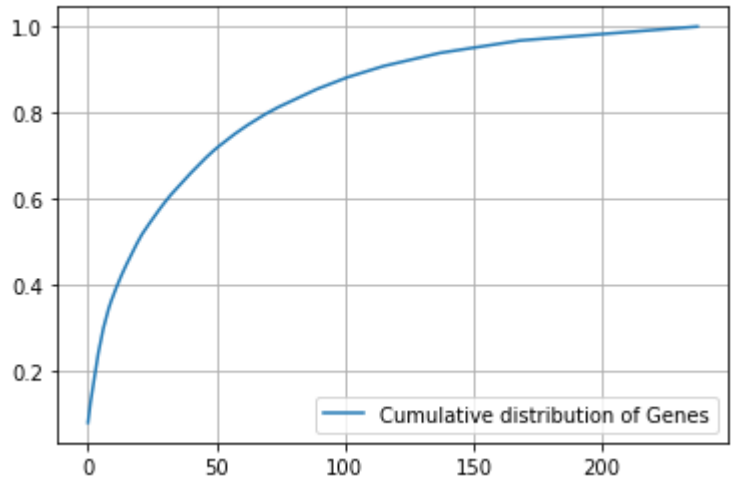
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
```

```
Ans: There are 238 different categories of genes in the train data, an
d they are distibuted as follows
```

In [18]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans. As assignment task below mentioned two approach are applied.

1. CountVectorizer with Unigram and Bigram
2. TFIDF Vectorization

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, both above mentioned encoding are better for Logistic regression(because of high dimensionality) while small dimension vecotization is better suited for Random Forests.

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene'
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
```

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using resp
```

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

In [22]:
```
# one-hot encoding of Gene feature eith unigram and bigram.
gene_vectorizer = CountVectorizer(CountVectorizer(ngram_range=(1, 2)))
train_gene_feature_onehotCoding_unibigram = gene_vectorizer.fit_transfor
test_gene_feature_onehotCoding_unibigram  = gene_vectorizer.transform(te
cv_gene_feature_onehotCoding_unibigram  = gene_vectorizer.transform(cv_d
```

In [23]:
```
#tfidf vectorization , using only top 3000 words and uni, bi , tri and d
gene_tfidf_ngram = TfidfVectorizer(ngram_range=(1, 5),max_features=3000,
train_gene_feature_tfidf_ngram = gene_tfidf_ngram.fit_transform(train_df
test_gene_feature_tfidf_ngram = gene_tfidf_ngram.transform(test_df['Gene
cv_gene_feature_tfidf_ngram = gene_tfidf_ngram.transform(cv_df['Gene'])
```

In [24]:
```
#tfidf vectorization , using only top 1500 words
gene_tfidf = TfidfVectorizer(max_features=1500)
train_gene_feature_tfidf = gene_tfidf.fit_transform(train_df['Gene'])
test_gene_feature_tfidf = gene_tfidf.transform(test_df['Gene'])
cv_gene_feature_tfidf = gene_tfidf.transform(cv_df['Gene'])
```

In [25]:
```
train_df['Gene'].head()
```

Out[25]:
```
2053     MYC
1285    HRAS
112     MSH6
3145    KRAS
274     EGFR
Name: Gene, dtype: object
```

In [26]:
```
gene_vectorizer.get_feature_names()
```

Out[26]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
```

In [27]:
```
print("train_gene_feature_onehotCoding is converted feature using one-ho
```

```
train_gene_feature_onehotCoding is converted feature using one-hot enc
oding(unigram and bigram) method. The shape of gene feature: (2124, 23
7)
```

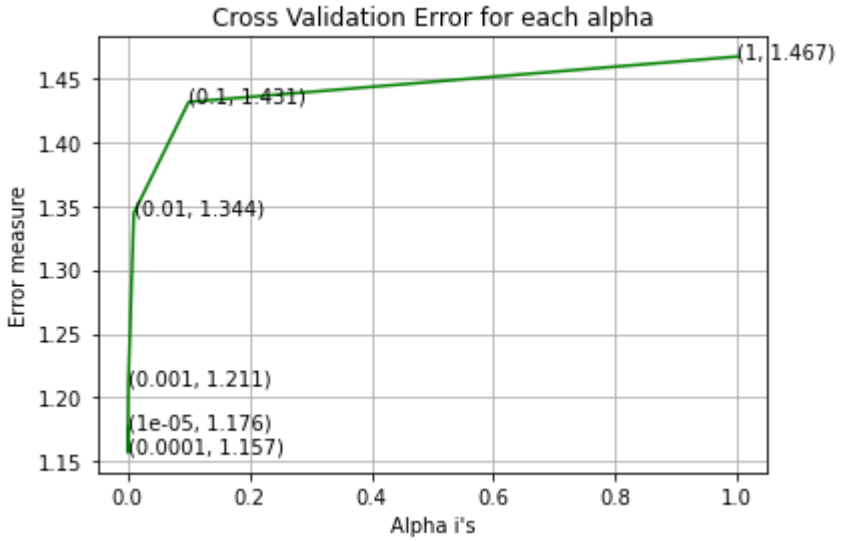
Q4. How good is this gene feature in predicting y_i?

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

In [28]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# 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=
    clf.fit(train_gene_feature_onehotCoding_unibigram, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding_unibigram, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding_unibi
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.class
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,

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
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', r
clf.fit(train_gene_feature_onehotCoding_unibigram, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding_unibigram, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding_unibig
print('For values of best alpha = ', alpha[best_alpha], "The train log l
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding_unibigram
print('For values of best alpha = ', alpha[best_alpha], "The cross valid
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding_unibigr
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
For values of alpha =  1e-05 The log loss is: 1.1760583582564577
For values of alpha =  0.0001 The log loss is: 1.156749204304756
For values of alpha =  0.001 The log loss is: 1.2106594619449444
```

```
For values of alpha =  0.01 The log loss is: 1.3444981663040354
For values of alpha =  0.1 The log loss is: 1.4314724981392364
For values of alpha =  1 The log loss is: 1.4670122945974997
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9952568919
065814
For values of best alpha =  0.0001 The cross validation log loss is:
1.156749204304756
For values of best alpha =  0.0001 The test log loss is: 1.16941729820
2618
```

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

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

In [29]:
```
print("Q6. How many data points in Test and CV datasets are covered by t

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], '
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  2
38  genes in train dataset?
Ans
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 516 out of  532 : 96.99248120300751
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

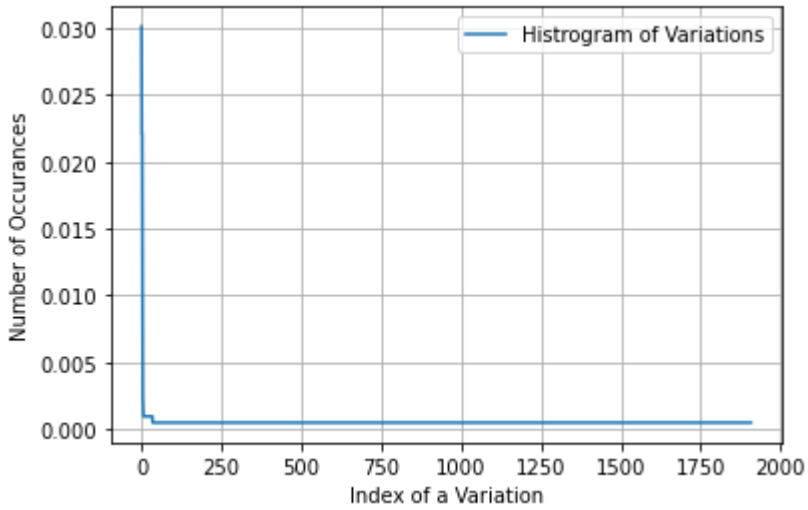
In [30]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1908
Truncating_Mutations    64
Amplification           47
Deletion                47
Fusions                 27
Overexpression           5
Q61L                     3
G67R                     2
G13D                     2
ETV6-NTRK3_Fusion        2
V321M                    2
Name: Variation, dtype: int64
```

In [31]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categorie
```

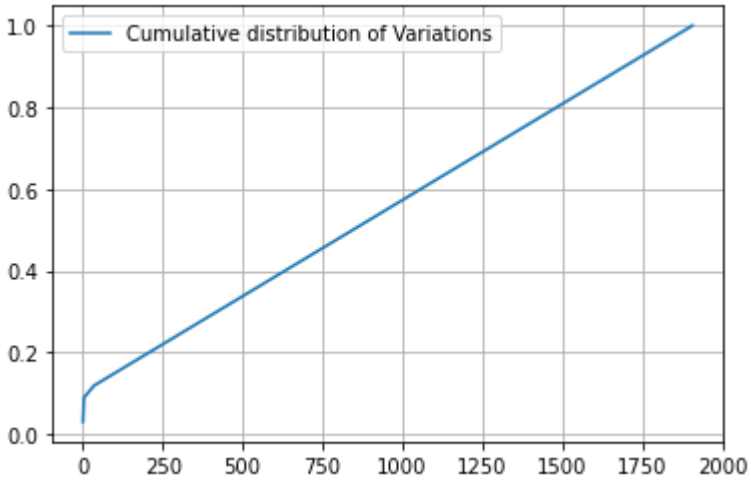
```
Ans: There are 1908 different categories of variations in the train da
ta, and they are distibuted as follows
```

In [32]:
```
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 [33]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.03013183 0.05225989 0.07438795 ... 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 [34]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Va
```

In [35]:
```
print("train_variation_feature_responseCoding is a converted feature usi
```

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

In [36]:
```
# one-hot encoding of variation feature with unigram and bigram.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding_unibigram  = variation_vectorizer.f
test_variation_feature_onehotCoding_unibigram  = variation_vectorizer.tr
cv_variation_feature_onehotCoding_unibigram  = variation_vectorizer.trar
```

In [37]:
```
#tfidf vectorization
variation_tfidf = TfidfVectorizer(max_features=1500)
train_variation_feature_tfidf = variation_tfidf.fit_transform(train_df['
test_variation_feature_tfidf = variation_tfidf.transform(test_df['Variat
cv_variation_feature_tfidf = variation_tfidf.transform(cv_df['Variation'
```

In [38]:
```
#tfidf vectorization ,
variation_tfidf_ngram = TfidfVectorizer(ngram_range=(1,5),max_features=3
train_variation_feature_tfidf_ngram = variation_tfidf_ngram.fit_transfor
test_variation_feature_tfidf_ngram = variation_tfidf_ngram.transform(tes
cv_variation_feature_tfidf_ngram = variation_tfidf_ngram.transform(cv_df
```

In [39]:
```
print("train_variation_feature_onehotEncoded is converted feature using
```

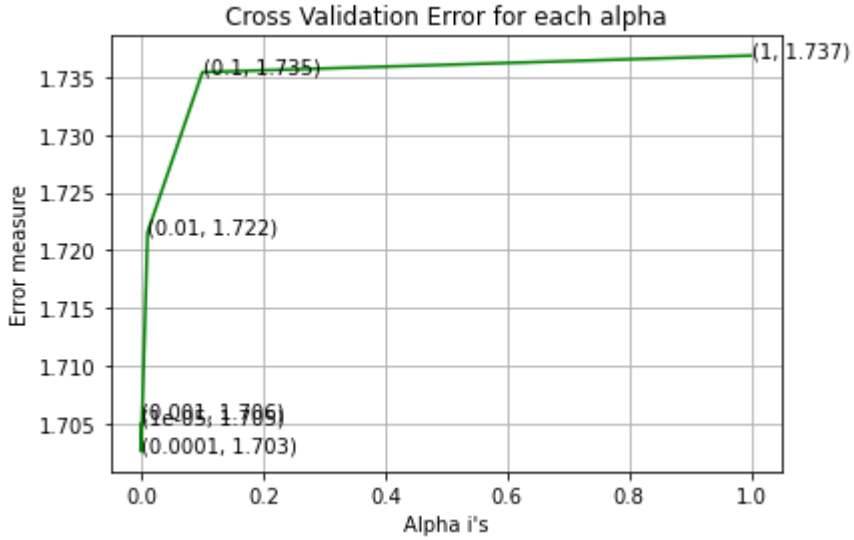
```
train_variation_feature_onehotEncoded is converted feature using the o
nne-hot encoding method. The shape of Variation feature: (2124, 2042)
```

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

Let's build a model just like the earlier!

In [40]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# 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=
    clf.fit(train_variation_feature_tfidf, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.class
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,

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
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', r
clf.fit(train_variation_feature_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_tfidf, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_tfidf)

print('For values of best alpha = ', alpha[best_alpha], "The train log l
predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross valid
predict_y = sig_clf.predict_proba(test_variation_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
For values of alpha =  1e-05 The log loss is: 1.7049757490055917
For values of alpha =  0.0001 The log loss is: 1.7025514959812336
For values of alpha =  0.001 The log loss is: 1.7055345559747044
For values of alpha =  0.01 The log loss is: 1.721564486037125
For values of alpha =  0.1 The log loss is: 1.7354257833139526
For values of alpha =  1 The log loss is: 1.7368589500044465
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9353295372
665512
For values of best alpha =  0.0001 The cross validation log loss is:
1.7025514959812336
For values of best alpha =  0.0001 The test log loss is: 1.72848537950
72517
```

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 [41]:
```
print("Q12. How many data points are covered by total ", unique_variatio
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Varia
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation']
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], '
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0
```

```
Q12. How many data points are covered by total  1908  genes in test an
d cross validation data sets?
Ans
1. In test data 53 out of 665 : 7.969924812030076
2. In cross validation data 50 out of  532 : 9.398496240601503
```

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 [42]:
```
# 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 [43]:
```
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 )/(to
            text_feature_responseCoding[row_index][i] = math.exp(sum_pro
            row_index += 1
    return text_feature_responseCoding
```

In [44]:
```
# building a CountVectorizer with all the words that occured minimum 3 t
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df
# 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
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 numb
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_fea
```

```
Total number of unique words in train data : 54577
```

In [45]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

In [46]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [47]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/t
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_te
```

In [48]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoc

# 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_onehotCodir

# 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, a
```

In [49]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [50]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 5416, 4: 4076, 5: 3071, 6: 2846, 7: 2383, 8: 2025, 9: 176
3, 10: 1400, 12: 1145, 15: 1062, 11: 1007, 13: 988, 16: 939, 14: 911,
18: 735, 19: 649, 20: 616, 17: 592, 24: 505, 21: 502, 22: 433, 23: 38
6, 25: 379, 26: 376, 30: 375, 45: 352, 32: 342, 27: 338, 28: 331, 29:
311, 36: 304, 50: 290, 31: 272, 33: 259, 35: 258, 40: 245, 34: 221, 3
8: 219, 39: 204, 42: 199, 48: 198, 37: 197, 44: 193, 46: 175, 43: 17
2, 49: 168, 41: 168, 54: 164, 56: 160, 51: 149, 47: 145, 63: 140, 53:
140, 57: 136, 58: 127, 55: 127, 52: 126, 59: 121, 64: 117, 61: 116, 6
0: 109, 65: 107, 62: 104, 72: 97, 69: 96, 88: 95, 75: 95, 66: 95, 76:
93, 73: 93, 68: 90, 67: 90, 81: 89, 80: 88, 70: 88, 74: 83, 90: 82, 7
8: 82, 100: 76, 71: 76, 91: 75, 87: 74, 77: 73, 98: 71, 89: 71, 83: 7
0, 99: 69, 105: 65, 92: 65, 85: 65, 82: 65, 79: 65, 114: 63, 103: 63,
95: 62, 84: 61, 94: 60, 101: 58, 97: 57, 86: 57, 104: 56, 120: 55, 10
2: 55, 96: 55, 93: 55, 132: 52, 144: 50, 134: 49, 106: 49, 126: 47, 1
17: 47, 116: 46, 127: 45, 112: 45, 115: 44, 113: 44, 141: 43, 110: 4
3, 121: 42, 108: 41, 109: 40, 129: 38, 128: 38, 118: 38, 107: 38, 15
0: 37, 123: 37, 148: 36, 136: 36, 167: 35, 157: 35, 138: 35, 130: 35,
125: 35, 169: 34, 152: 34, 135: 34, 122: 34, 119: 34, 160: 33, 137: 3
3, 162: 32, 161: 32, 111: 32, 175: 31, 154: 31, 140: 31, 224: 30, 17
```

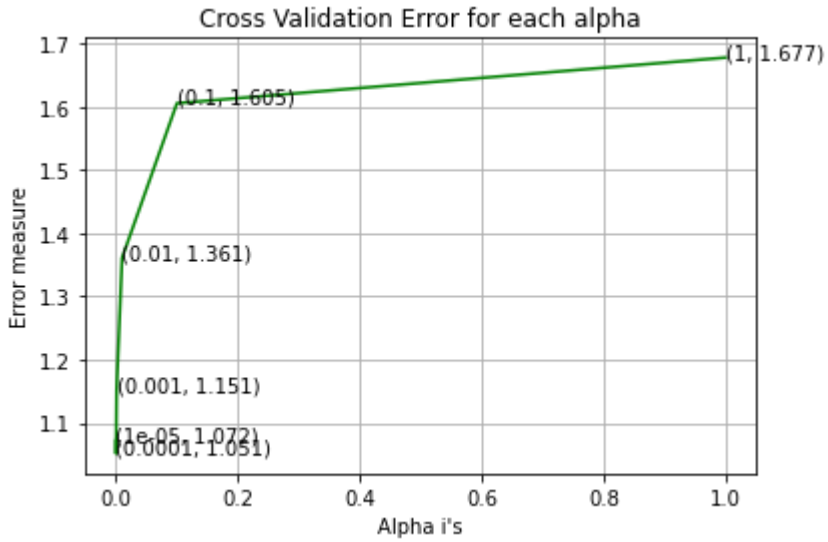
In [51]:
```
#tfidf vectorization with considering top 1500 features only
text_tfidf = TfidfVectorizer(max_features=1500)
train_text_feature_tfidf = text_tfidf.fit_transform(train_df['TEXT'])
test_text_feature_tfidf = text_tfidf.transform(test_df['TEXT'])
cv_text_feature_tfidf = text_tfidf.transform(cv_df['TEXT'])
```

In [52]:
```
#tfidf vectorization with considering top 3000 features only
text_tfidf_ngram = TfidfVectorizer(ngram_range=(1,5),max_features=3000,m
train_text_feature_tfidf_ngram = text_tfidf_ngram.fit_transform(train_df
test_text_feature_tfidf_ngram = text_tfidf_ngram.transform(test_df['TEXT
cv_text_feature_tfidf_ngram = text_tfidf_ngram.transform(cv_df['TEXT'])
```

In [53]:
```
# one-hot encoding of variation feature with unigram and bigram.
text_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_text_feature_onehotCoding_unibigram  = text_vectorizer.fit_transfo
test_text_feature_onehotCoding_unibigram  = text_vectorizer.transform(te
cv_text_feature_onehotCoding_unibigram  = text_vectorizer.transform(cv_c
```

In [54]:
```
# Train a Logistic regression+Calibration model using text features whic
alpha = [10 ** x for x in range(-5, 1)]

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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.class
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,

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
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', r
clf.fit(train_text_feature_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_tfidf, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_tfidf)
log_reg_train_log_loss = log_loss(y_train, predict_y, labels=clf.classes
print('For values of best alpha = ', alpha[best_alpha], "The train log l
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
log_reg_cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps
print('For values of best alpha = ', alpha[best_alpha], "The cross valid
predict_y = sig_clf.predict_proba(test_text_feature_tfidf)
log_reg_test_log_loss=log_loss(y_test, predict_y, labels=clf.classes_, e
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
For values of alpha =  1e-05 The log loss is: 1.0717170323763943
For values of alpha =  0.0001 The log loss is: 1.0512792395932156
For values of alpha =  0.001 The log loss is: 1.1511748073634984
For values of alpha =  0.01 The log loss is: 1.3607462483454702
For values of alpha =  0.1 The log loss is: 1.6051637624714628
For values of alpha =  1 The log loss is: 1.6773374694291572
```




```
For values of best alpha =  0.0001 The train log loss is: 0.8005054947
736797
For values of best alpha =  0.0001 The cross validation log loss is:
1.0512792395932156
For values of best alpha =  0.0001 The test log loss is: 1.08067253850
82228
```

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

Ans. Yes, it seems like!

In [55]:
```
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_count
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [56]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appea
```

```
97.08 % of word of test data appeared in train data
97.616 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [57]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, c
    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)

    loss = log_loss(test_y, sig_clf.predict_proba(test_x))
    # for calculating log_loss we willl provide the array of probabiliti
    print("Log loss :",loss)
    # calculating the number of data points that are misclassified
    misclassified_points = np.count_nonzero((pred_y- test_y))/test_y.sha
    print("Number of mis-classified points :", misclassified_points)
    plot_confusion_matrix(test_y, pred_y)
    return (loss,misclassified_points)
```

In [58]:
```
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 [59]:
```
# 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 o
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 [
        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 po
        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 [

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

Stacking the three types of features

In [60]:
```
# 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_unibigram = hstack((train_gene_feature_oneho
test_gene_var_onehotCoding_unibigram = hstack((test_gene_feature_onehotC
cv_gene_var_onehotCoding_unibigram = hstack((cv_gene_feature_onehotCodin

train_x_onehotCoding_unibigram = hstack((train_gene_var_onehotCoding_uni
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding_unibigram = hstack((test_gene_var_onehotCoding_unibi
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding_unibigram = hstack((cv_gene_var_onehotCoding_unibigram
cv_y = np.array(list(cv_df['Class']))
```

In [61]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]
import scipy
train_gene_var_tfidf = hstack((train_gene_feature_tfidf,train_variation_
test_gene_var_tfidf =hstack((test_gene_feature_tfidf,test_variation_feat
cv_gene_var_tfidf= hstack((cv_gene_feature_tfidf,cv_variation_feature_tf

train_x_tfidf = hstack((train_gene_var_tfidf, train_text_feature_tfidf))
train_y = np.array(list(train_df['Class']))

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

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


#tfidf vectorization with ngram range of 1-4 and max_features=3000
train_gene_var_tfidf_ngram = hstack((train_gene_feature_tfidf_ngram,trai
test_gene_var_tfidf_ngram =hstack((test_gene_feature_tfidf_ngram ,test_v
cv_gene_var_tfidf_ngram = hstack((cv_gene_feature_tfidf_ngram ,cv_variat

train_x_tfidf_ngram  = hstack((train_gene_var_tfidf_ngram , train_text_t
train_y_ngram  = np.array(list(train_df['Class']))

test_x_tfidf_ngram  = hstack((test_gene_var_tfidf_ngram , test_text_feat
test_y_ngram  = np.array(list(test_df['Class']))

cv_x_tfidf_ngram  = hstack((cv_gene_var_tfidf_ngram , cv_text_feature_tf
cv_y_ngram  = np.array(list(cv_df['Class']))
```

In [62]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", t
print("(number of data points * number of features) in test data = ", te
print("(number of data points * number of features) in cross validation
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2
420608)
(number of data points * number of features) in test data =  (665, 242
0608)
(number of data points * number of features) in cross validation data
= (532, 2420608)
```

In [63]:
```
print(" TFIDF encoding features :")
print("(number of data points * number of features) in train data = ", t
print("(number of data points * number of features) in test data = ", te
print("(number of data points * number of features) in cross validation
```

```
 TFIDF encoding features :
(number of data points * number of features) in train data =  (2124, 3
237)
(number of data points * number of features) in test data =  (665, 323
7)
(number of data points * number of features) in cross validation data
= (532, 3237)
```

In [64]:
```
print(" TFIDF encoding features with ngram range of 1-4 and max_features
print("(number of data points * number of features) in train data = ", t
print("(number of data points * number of features) in test data = ", te
print("(number of data points * number of features) in cross validation
```

```
 TFIDF encoding features with ngram range of 1-4 and max_features 3000
:
(number of data points * number of features) in train data =  (2124, 3
113)
(number of data points * number of features) in test data =  (665, 311
3)
(number of data points * number of features) in cross validation data
= (532, 3113)
```

4.1. Base Line Model

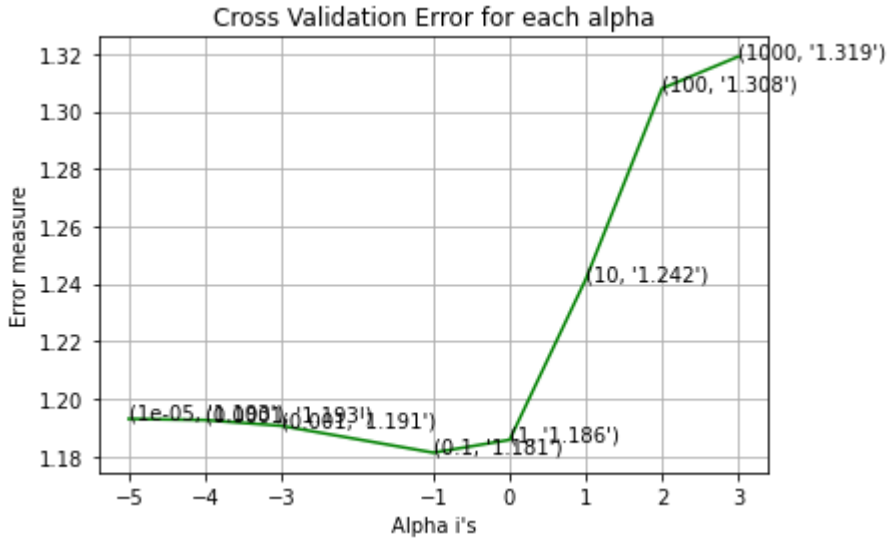
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [65]:
```
# find more about Multinomial Naive base function here http://scikit-lea
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pri

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


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


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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    # to avoid rounding error while multiplying probabilites we use log-
    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_ar
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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)


predict_y = sig_clf.predict_proba(train_x_tfidf)
nb_train_logloss = log_loss(y_train, predict_y, labels=clf.classes_, eps
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf)
nb_cv_logloss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf)
nb_test_logloss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for alpha = 1e-05
Log Loss : 1.1930367019979988
for alpha = 0.0001
Log Loss : 1.1926213465089757
for alpha = 0.001
Log Loss : 1.190626439298192
for alpha = 0.1
Log Loss : 1.1812790058864484
for alpha = 1
Log Loss : 1.185738307501908
for alpha = 10
Log Loss : 1.2416029604415557
for alpha = 100
Log Loss : 1.3079219828033826
for alpha = 1000
Log Loss : 1.3190166550614015
```



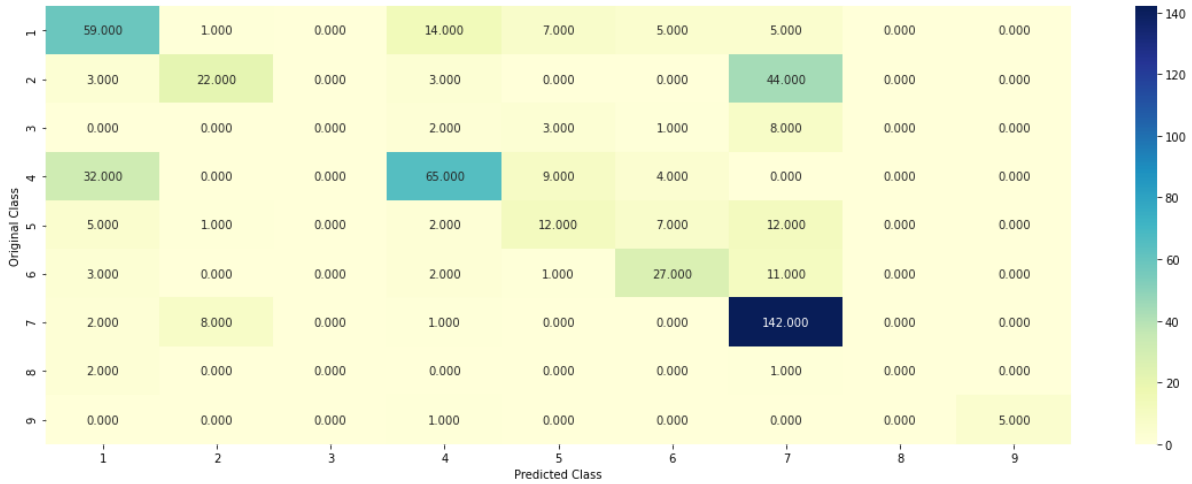

```
For values of best alpha =  0.1 The train log loss is: 0.758098408163
5147
For values of best alpha =  0.1 The cross validation log loss is: 1.1
812790058864484
```

```
For values of best alpha =  0.1 The test log loss is: 1.2118257462881
639
```

4.1.1.2. Testing the model with best hyper paramters

In [66]:
```
# find more about Multinomial Naive base function here http://scikit-lea
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pri

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
# to avoid rounding error while multiplying probabilites we use log-prob
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pred
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf))
```

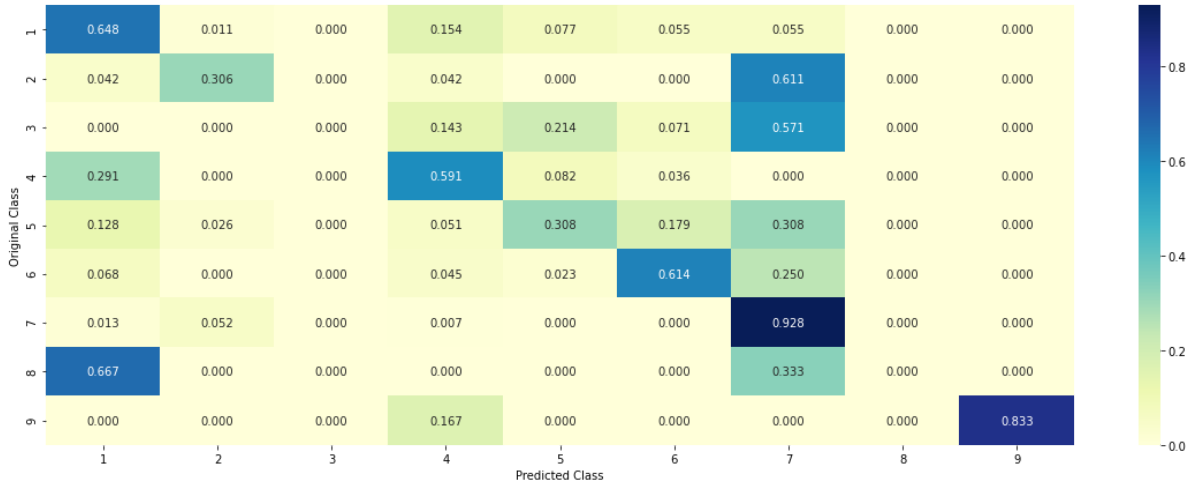
```
Log Loss : 1.1812790058864484
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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

```
----
```


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



4.1.1.3. Feature Importance, Correctly classified point

In [67]:
```
test_point_index = 34
no_feature = 300

predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0566 0.062  0.0169 0.7159 0.0391 0.
0394 0.0613 0.0045 0.0043]]
Actual Class : 4
--------------------------------------------------
Out of the top  300  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [68]:
```
test_point_index = 19
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.089  0.0688 0.0256 0.1097 0.5286 0.
0755 0.0895 0.0068 0.0065]]
Actual Class : 3
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

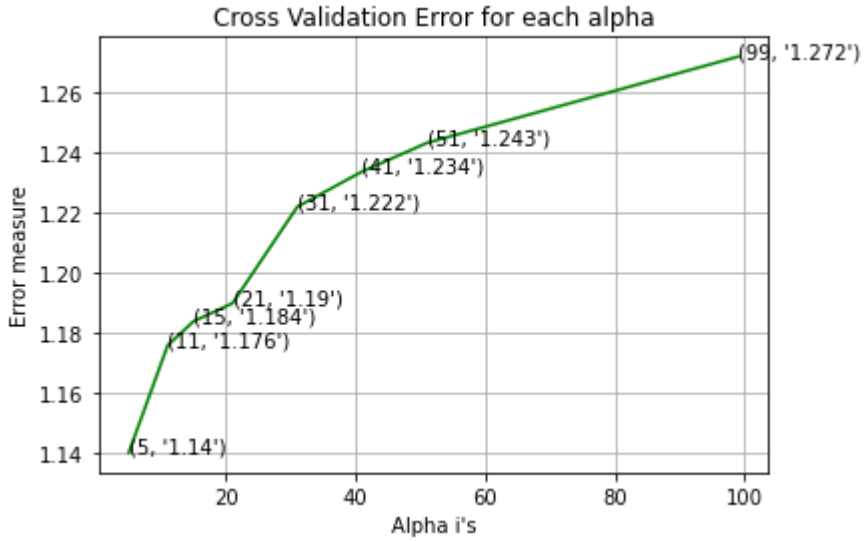
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [69]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/st
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto
# 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 val
# 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-d
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    # to avoid rounding error while multiplying probabilites we use log-
    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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
knn_train_logloss = log_loss(y_train, predict_y, labels=clf.classes_, ep
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf)
knn_cv_logloss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-1
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf)
knn_test_logloss = log_loss(y_test, predict_y, labels=clf.classes_, eps=
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for alpha = 5
Log Loss : 1.1401523619651517
for alpha = 11
Log Loss : 1.1757562506988009
for alpha = 15
Log Loss : 1.1837848000867857
for alpha = 21
Log Loss : 1.1897429314859729
for alpha = 31
Log Loss : 1.2218232809029348
for alpha = 41
Log Loss : 1.2335220919210976
for alpha = 51
Log Loss : 1.2430320561111556
for alpha = 99
Log Loss : 1.2717056536877207
```

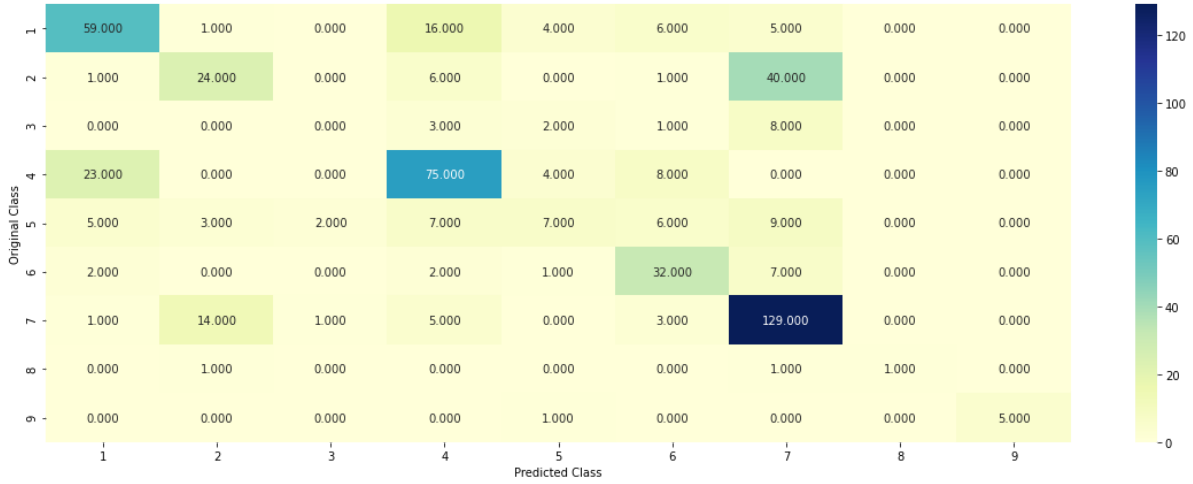



```
For values of best alpha =  5 The train log loss is: 0.900275701001416
7
For values of best alpha =  5 The cross validation log loss is: 1.1401
523619651517
For values of best alpha =  5 The test log loss is: 1.2312924708430761
```

4.2.2. Testing the model with best hyper paramters

In [70]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/st
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto
# 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 val
# 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-d
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv
```

```
Log loss : 1.1401523619651517
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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


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

Out[70]: (1.1401523619651517, 0.37593984962406013)

4.2.3.Sample Query point -1

In [71]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
Predicted Class : 7
Actual Class : 2
The  5  nearest neighbours of the test points belongs to classes [7 6
2 7 7]
Fequency of nearest points : Counter({7: 3, 6: 1, 2: 1})
```

4.2.4. Sample Query Point-2

In [72]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

test_point_index = 100

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

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

4.3. Logistic Regression

4.3.1. With Class balancing

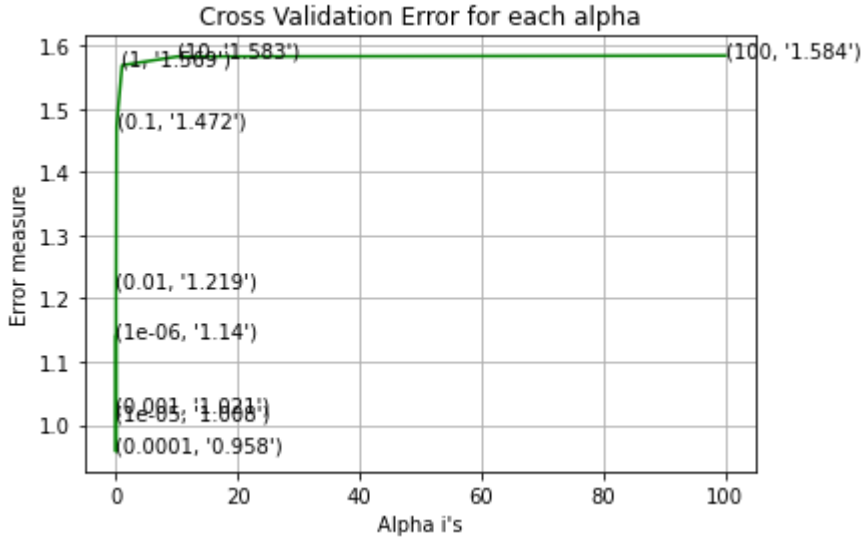
4.3.1.1. Hyper paramter tuning

In [73]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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',
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    # to avoid rounding error while multiplying probabilites we use log-
    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], pe
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
log_reg_classbalancing_train_logloss = log_loss(y_train, predict_y, labe
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf)
log_reg_classbalancing_cv_logloss = log_loss(y_cv, predict_y, labels=clf
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf)
log_reg_classbalancing_test_logloss = log_loss(y_test, predict_y, labels
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for alpha = 1e-06
Log Loss : 1.1402013244436862
for alpha = 1e-05
Log Loss : 1.008170583997704
for alpha = 0.0001
Log Loss : 0.95764467822355
for alpha = 0.001
Log Loss : 1.0209911297975676
for alpha = 0.01
Log Loss : 1.2191976197348715
for alpha = 0.1
Log Loss : 1.4722289416486185
for alpha = 1
Log Loss : 1.5688136221492157
for alpha = 10
Log Loss : 1.5825113673382452
for alpha = 100
Log Loss : 1.5840952440527494
```



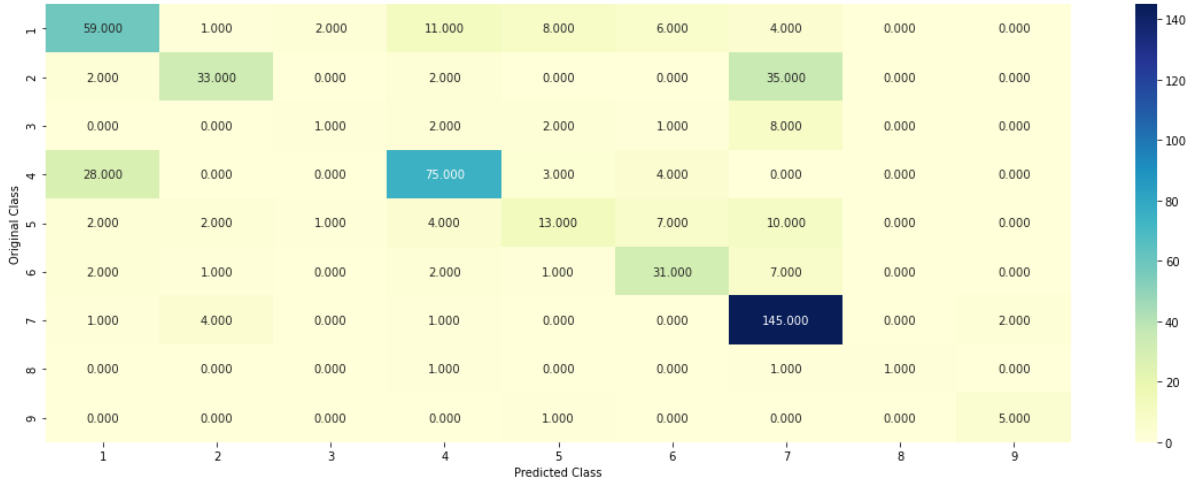

```
For values of best alpha =  0.0001 The train log loss is: 0.467678045
77957706
For values of best alpha =  0.0001 The cross validation log loss is:
0.95764467822355
```

```
For values of best alpha =  0.0001 The test log loss is: 1.0088153202
193393
```

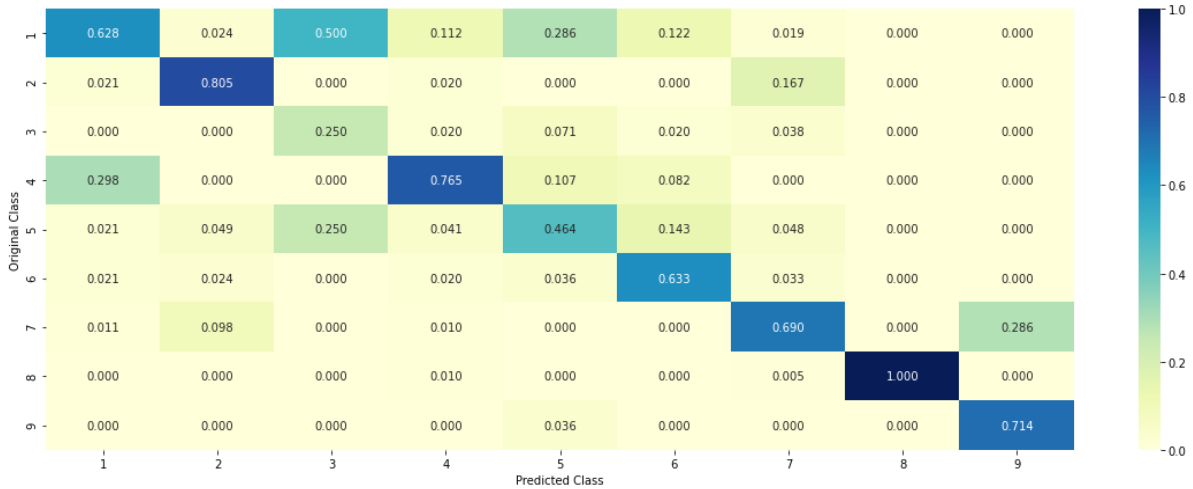
4.3.1.2. Testing the model with best hyper paramters

In [74]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-d
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], pe
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv
```

```
Log loss : 0.95764467822355
Number of mis-classified points : 0.3176691729323308
-------------------- Confusion matrix --------------------
```




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




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

Out[74]: (0.95764467822355, 0.3176691729323308)

4.3.1.3. Feature Importance

In [75]:
```
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[
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our quer
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pres
```

4.3.1.3.1. Correctly Classified point

In [76]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], pe
clf.fit(train_x_tfidf,train_y)
test_point_index = 46
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0242 0.1317 0.0088 0.0185 0.0199 0.
0384 0.7499 0.0033 0.0053]]
Actual Class : 7
--------------------------------------------------
134 Text feature [003] present in test data point [True]
274 Text feature [0003] present in test data point [True]
340 Text feature [035] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [77]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0264 0.3611 0.0078 0.0133 0.0273 0.
0801 0.4681 0.006  0.0098]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

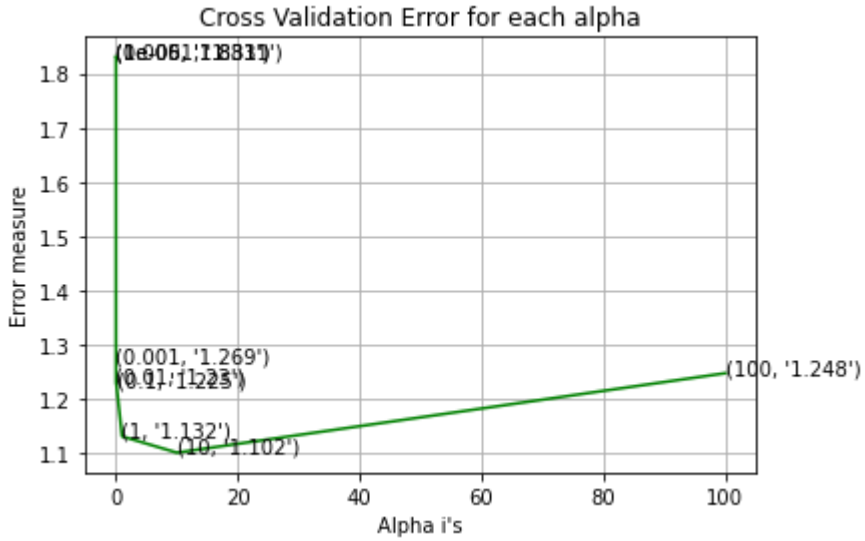
Logistic Regression CountVectorizer with Unigram and Bigrams

In [78]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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',
    clf.fit(train_x_onehotCoding_unibigram, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_unibigram, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_unibigram)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    # to avoid rounding error while multiplying probabilites we use log-
    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], pe
clf.fit(train_x_onehotCoding_unibigram, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_unibigram, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_unibigram)
log_reg_classbalancing_unibigram_train_logloss = log_loss(y_train, predi
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_onehotCoding_unibigram)
log_reg_classbalancing_unibigram_cv_logloss = log_loss(y_cv, predict_y,
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_onehotCoding_unibigram)
log_reg_classbalancing_unibigram_test_logloss = log_loss(y_test, predict
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

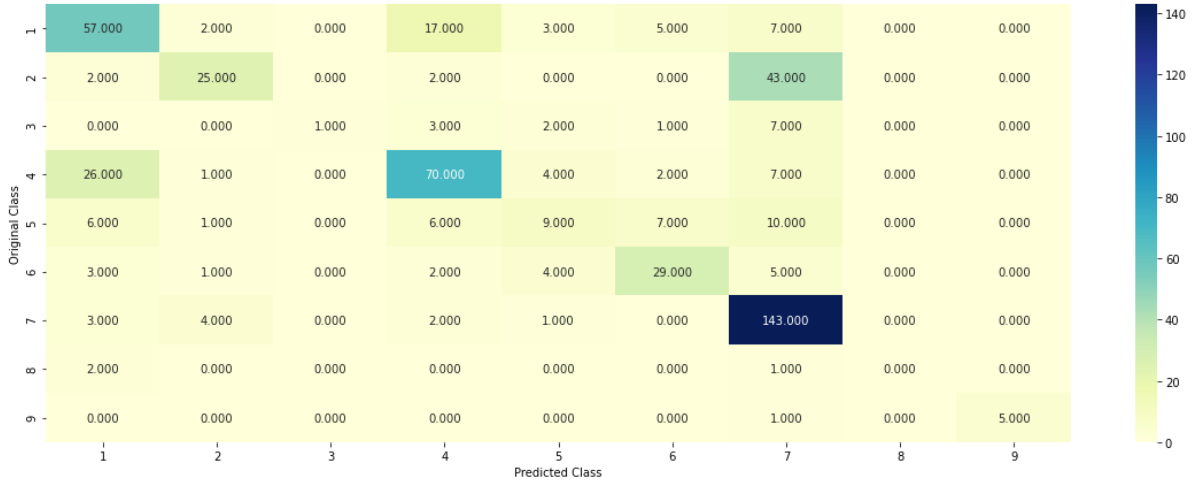
```
for alpha = 1e-06
Log Loss : 1.830889358572364
for alpha = 1e-05
Log Loss : 1.830889358572364
for alpha = 0.0001
Log Loss : 1.830889358572364
for alpha = 0.001
Log Loss : 1.2690033179464888
for alpha = 0.01
Log Loss : 1.2303178140100133
for alpha = 0.1
Log Loss : 1.2246563820883858
for alpha = 1
Log Loss : 1.132010356154486
for alpha = 10
Log Loss : 1.1020308462124042
for alpha = 100
Log Loss : 1.248254328760224
```



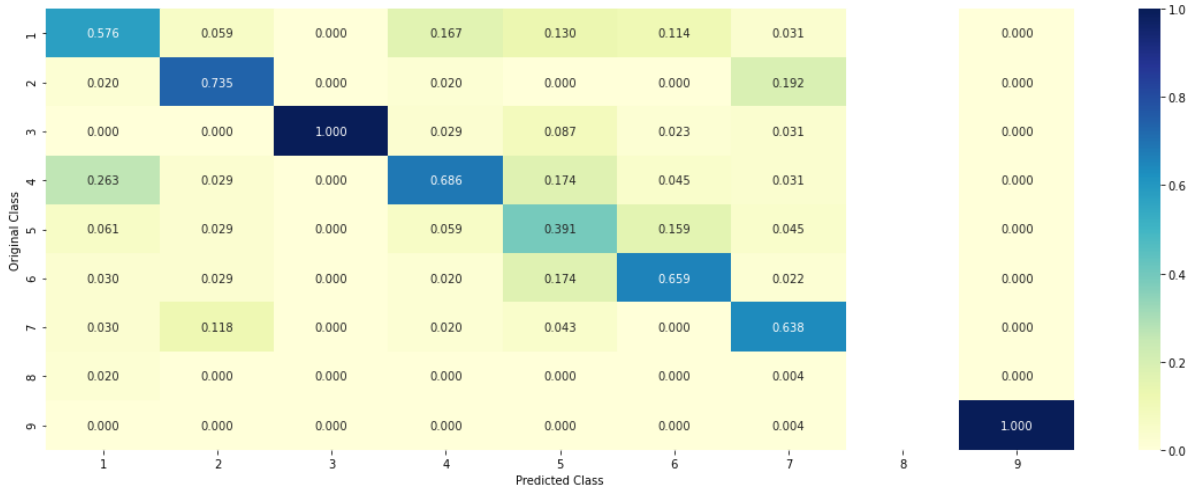
```
For values of best alpha =  10 The train log loss is: 0.46767804577957
706
For values of best alpha =  10 The cross validation log loss is: 0.957
64467822355
For values of best alpha =  10 The test log loss is: 1.008815320219339
3
```

In [79]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-d
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], pe
predict_and_plot_confusion_matrix(train_x_onehotCoding_unibigram, train_
```

```
Log loss : 1.1020308462124042
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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




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

Out[79]: (1.1020308462124042, 0.36278195488721804)

In [80]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], pe
clf.fit(train_x_onehotCoding_unibigram,train_y)
test_point_index = 41
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding_unibigram[test_point
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
print(indices)
#get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
```

```
Predicted Class : 4
Predicted Class Probabilities: [[7.240e-02 1.560e-02 4.000e-04 7.923e-
01 6.360e-02 7.700e-03 3.880e-02
  9.000e-03 1.000e-04]]
Actual Class : 1
--------------------------------------------------
[[1630545 1009041 1798570  446801 2377761 2220787 2172543 2106552 1548
642
  1629994  296259 1279649 1283164 1494764 1464713 1363235 1639709 1813
745
  1566001 1004322 2018624  193189 1954450  443844 1406532 1161771 1368
589
  1430134 1461019 1146631 1509432  288879  252578  497965  946354 1458
168
   939652 1926586  897061 1287601  455533 1678343  491787 1664622 1634
042
   504821  252046  389674 1458331 1492326  703876 1566354 1295626 2082
324
   357352 2355903 1658870  863834 1933522 1001802 1815531  321352 2218
404
   553941 1816340 1709074  707860 1158019 1431630 1802956 1721860  446
670
  1443808 1972216  257787 1725432 1865459 1104610 1279205  990904  505
250
  1744951 1024412 1390035 2136618  947330  785550 2221517 1177663 2084
726
  2293203  492913 2053307  403555 1569523 1699980 1662538  779695 1440
421
  2267026]]
```

In [81]:
```
test_point_index = 1
no_feature = 300
predicted_cls = sig_clf.predict(test_x_onehotCoding_unibigram[test_point
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

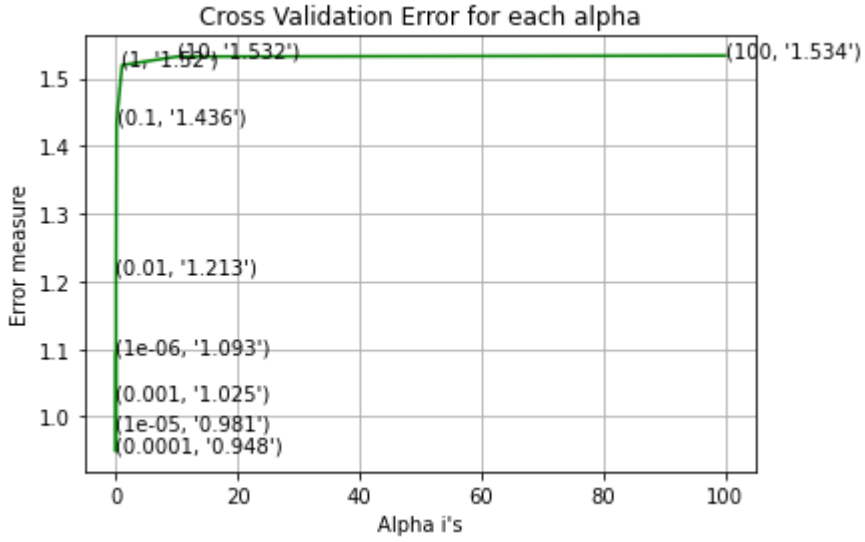
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0962 0.2757 0.0198 0.0734 0.0532 0.
0911 0.3704 0.0071 0.013 ]]
Actual Class : 2
--------------------------------------------------
```

Logistic regression with class balancing ,tfidf featurization using ngram range [1 , 4] and top 3000 features

In [82]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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',
    clf.fit(train_x_tfidf_ngram, train_y_ngram)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf_ngram, train_y_ngram)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf_ngram)
    cv_log_error_array.append(log_loss(cv_y_ngram, sig_clf_probs, labels
    # to avoid rounding error while multiplying probabilites we use log-
    print("Log Loss :",log_loss(cv_y_ngram, 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], pe
```

```
clf.fit(train_x_tfidf_ngram, train_y_ngram)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf_ngram, train_y_ngram)

predict_y = sig_clf.predict_proba(train_x_tfidf_ngram)
log_reg_classbalancing_train_logloss_ngram = log_loss(train_y_ngram, pre
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf_ngram)
log_reg_classbalancing_cv_logloss_ngram = log_loss(cv_y_ngram, predict_y
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf_ngram)
log_reg_classbalancing_test_logloss_ngram = log_loss(test_y_ngram, predi
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

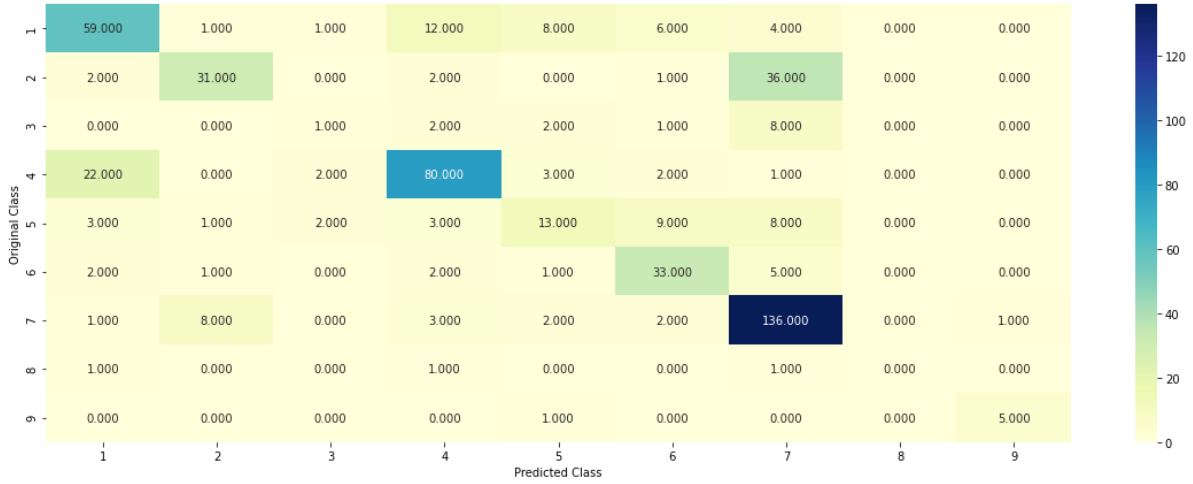
```
for alpha = 1e-06
Log Loss : 1.092869210297739
for alpha = 1e-05
Log Loss : 0.9808712919617774
for alpha = 0.0001
Log Loss : 0.9481342469907434
for alpha = 0.001
Log Loss : 1.0254397855659443
for alpha = 0.01
Log Loss : 1.2126540513242619
for alpha = 0.1
Log Loss : 1.4356792255055342
for alpha = 1
Log Loss : 1.5197785740889123
for alpha = 10
Log Loss : 1.5323492498701117
for alpha = 100
Log Loss : 1.5338250627245795
```



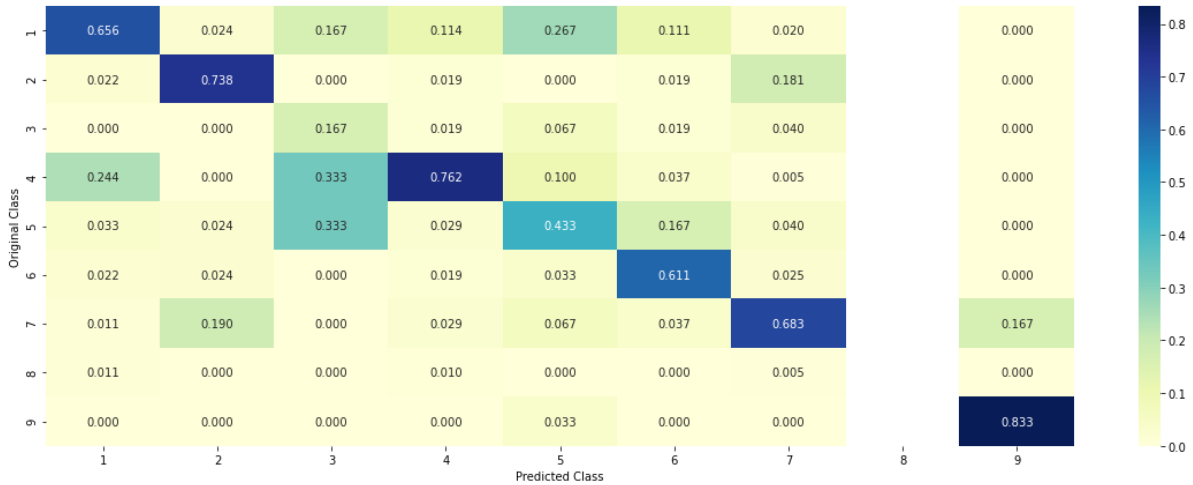

```
For values of best alpha =  0.0001 The train log loss is: 0.6695119203
387395
For values of best alpha =  0.0001 The cross validation log loss is:
0.9481342469907434
For values of best alpha =  0.0001 The test log loss is: 0.98011947504
59213
```

In [83]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-d
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], pe
log_loss_balancing_tfidf_ngram, misclassified_percentage_ngram = predict
```

```
Log loss : 0.9481342469907434
Number of mis-classified points : 0.32706766917293234
-------------------- Confusion matrix --------------------
```




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




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

Correctly classified points

In [84]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], pe
clf.fit(train_x_tfidf_ngram,train_y_ngram)
test_point_index = 46
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf_ngram[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0195 0.1222 0.005  0.0145 0.0141 0.
0636 0.7535 0.0035 0.0041]]
Actual Class : 7
--------------------------------------------------
30 Text feature [12] present in test data point [True]
258 Text feature [01] present in test data point [True]
306 Text feature [09] present in test data point [True]
343 Text feature [046] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

Incorrectly classified points

In [85]:
```
test_point_index = 3
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf_ngram[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

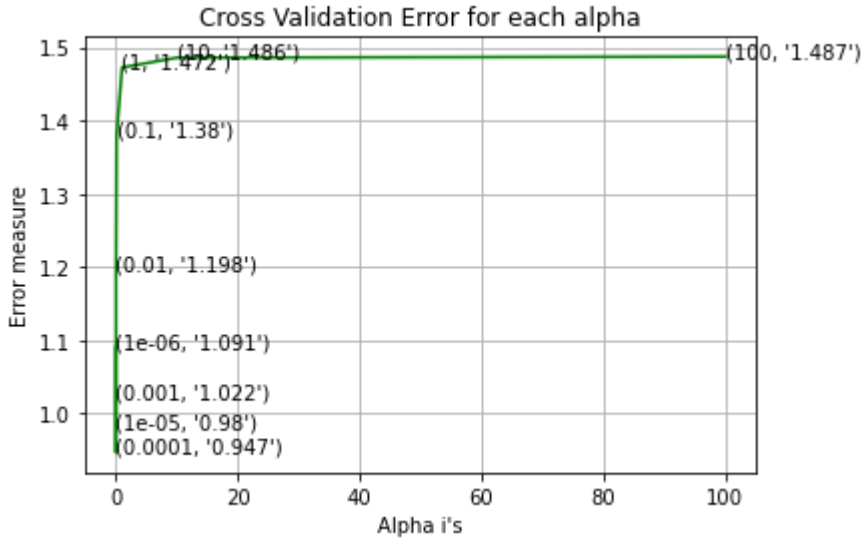
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0052 0.1901 0.0217 0.0162 0.0144 0.
0333 0.71   0.0061 0.0031]]
Actual Class : 7
--------------------------------------------------
30 Text feature [12] present in test data point [True]
138 Text feature [10th] present in test data point [True]
258 Text feature [01] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

Logistic regression without class balancing with ngram range(1,4) and max features 3000

In [86]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

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


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=
    clf.fit(train_x_tfidf_ngram, train_y_ngram)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf_ngram, train_y_ngram)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf_ngram)
    cv_log_error_array.append(log_loss(cv_y_ngram, sig_clf_probs, labels
    # to avoid rounding error while multiplying probabilites we use log-
    print("Log Loss :",log_loss(cv_y_ngram, 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], pe
```

```
clf.fit(train_x_tfidf_ngram, train_y_ngram)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf_ngram, train_y_ngram)

predict_y = sig_clf.predict_proba(train_x_tfidf_ngram)
log_reg_train_logloss_ngram = log_loss(train_y_ngram, predict_y, labels=
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf_ngram)
log_reg_cv_logloss_ngram = log_loss(cv_y_ngram, predict_y, labels=clf.cl
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf_ngram)
log_reg_test_logloss_ngram = log_loss(test_y_ngram, predict_y, labels=cl
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for alpha = 1e-06
Log Loss : 1.090791553205878
for alpha = 1e-05
Log Loss : 0.9801613614470974
for alpha = 0.0001
Log Loss : 0.9467560545012379
for alpha = 0.001
Log Loss : 1.0224927204673386
for alpha = 0.01
Log Loss : 1.197588922548627
for alpha = 0.1
Log Loss : 1.3801325326095382
for alpha = 1
Log Loss : 1.472472976865018
for alpha = 10
Log Loss : 1.485867059686772
for alpha = 100
Log Loss : 1.487369694938374
```

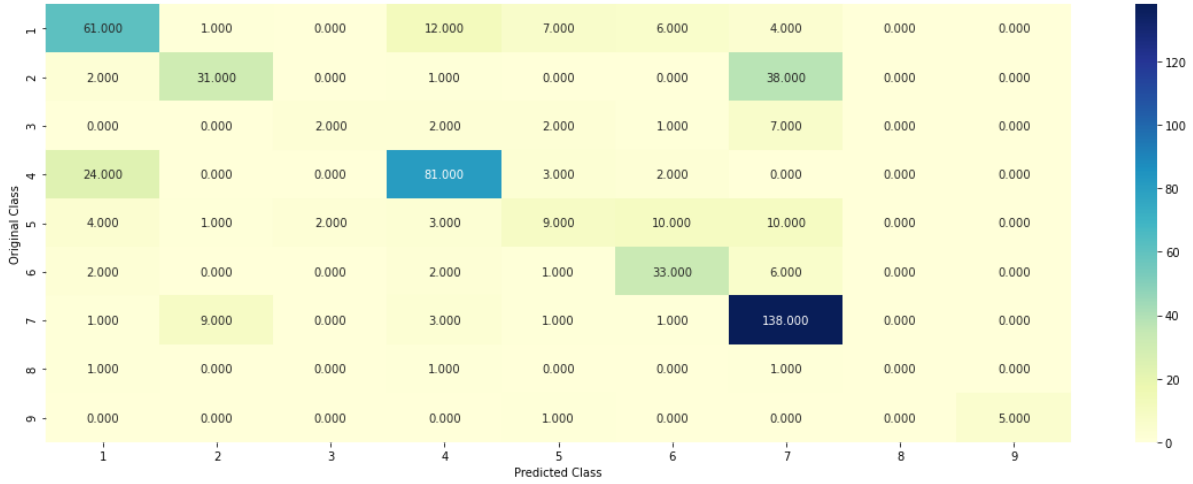



```
For values of best alpha =  0.0001 The train log loss is: 0.669511920
3387395
For values of best alpha =  0.0001 The cross validation log loss is:
0.9481342469907434
```

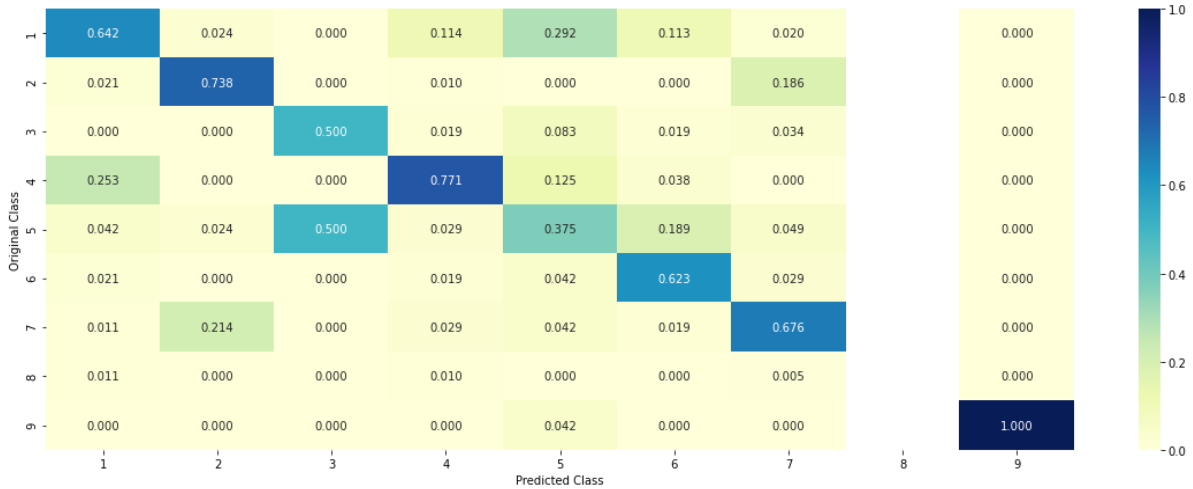
```
For values of best alpha =  0.0001 The test log loss is: 0.9801194750
459213
```

In [87]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-d
#-------------------------------
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
log_loss_without_balancing_tfidf_ngram, misclassified_without_balancing_
```

```
Log loss : 0.9467560545012379
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```




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




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

Correctly classified points

In [88]:
```
# from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
clf.fit(train_x_tfidf_ngram,train_y_ngram)
test_point_index = 46
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf_ngram[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0195 0.1222 0.005  0.0145 0.0141 0.
0636 0.7535 0.0035 0.0041]]
Actual Class : 7
--------------------------------------------------
9 Text feature [12] present in test data point [True]
245 Text feature [01] present in test data point [True]
318 Text feature [09] present in test data point [True]
405 Text feature [0013] present in test data point [True]
460 Text feature [046] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

Incorrectly classified points

In [89]:
```
test_point_index = 3
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf_ngram[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0052 0.1901 0.0217 0.0162 0.0144 0.
0333 0.71   0.0061 0.0031]]
Actual Class : 7
--------------------------------------------------
9 Text feature [12] present in test data point [True]
125 Text feature [10th] present in test data point [True]
245 Text feature [01] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

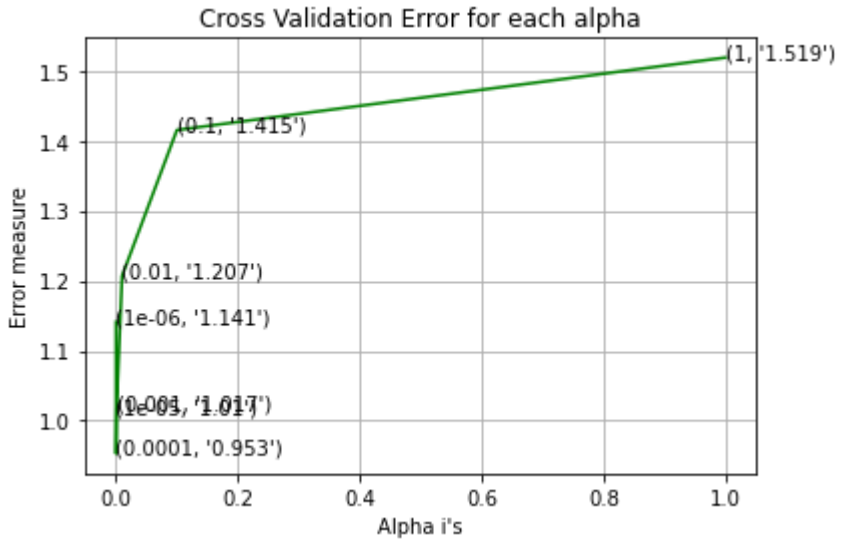
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [90]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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=
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    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', r
```

```
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
log_reg_train_tfidf_logloss = log_loss(y_train, predict_y, labels=clf.cl
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf)
log_reg_cv_tfidf_logloss = log_loss(y_cv, predict_y, labels=clf.classes_
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf)
log_reg_test_tfidf_logloss = log_loss(y_test, predict_y, labels=clf.clas
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for alpha = 1e-06
Log Loss : 1.1410883566713925
for alpha = 1e-05
Log Loss : 1.0101097546619342
for alpha = 0.0001
Log Loss : 0.9529559265760372
for alpha = 0.001
Log Loss : 1.016868848490849
for alpha = 0.01
Log Loss : 1.2067932447616938
for alpha = 0.1
Log Loss : 1.4151849303285828
for alpha = 1
Log Loss : 1.5185842409413115
```

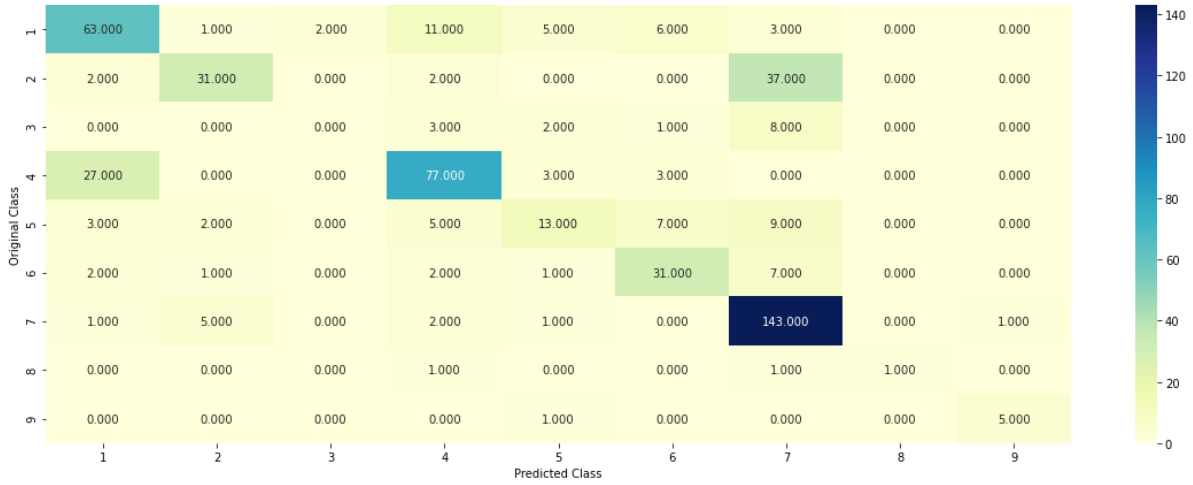



```
For values of best alpha =  0.0001 The train log loss is: 0.4545069475
4303317
For values of best alpha =  0.0001 The cross validation log loss is:
0.9529559265760372
For values of best alpha =  0.0001 The test log loss is: 1.00767073216
25014
```

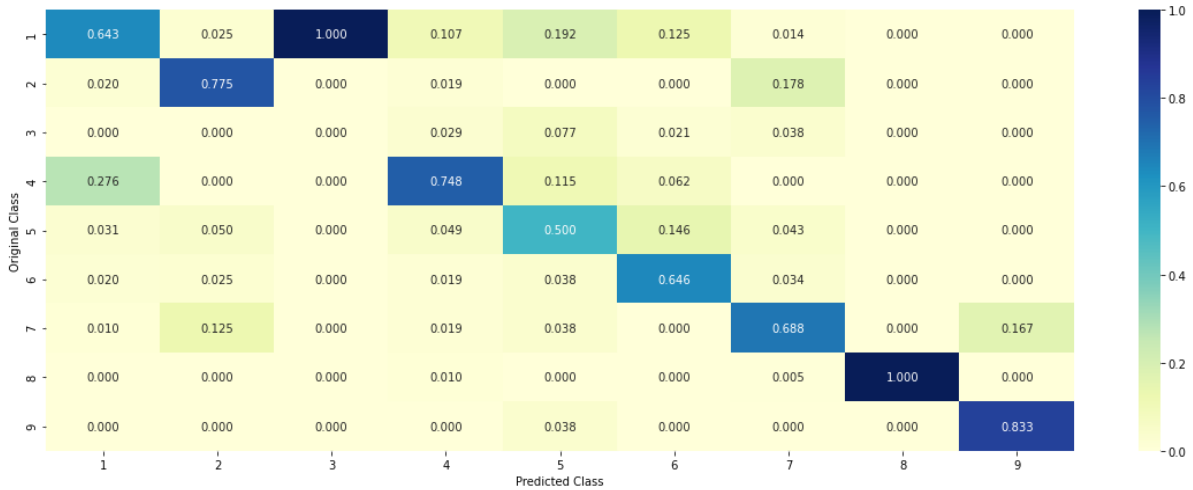
4.3.2.2. Testing model with best hyper parameters

In [91]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv
```

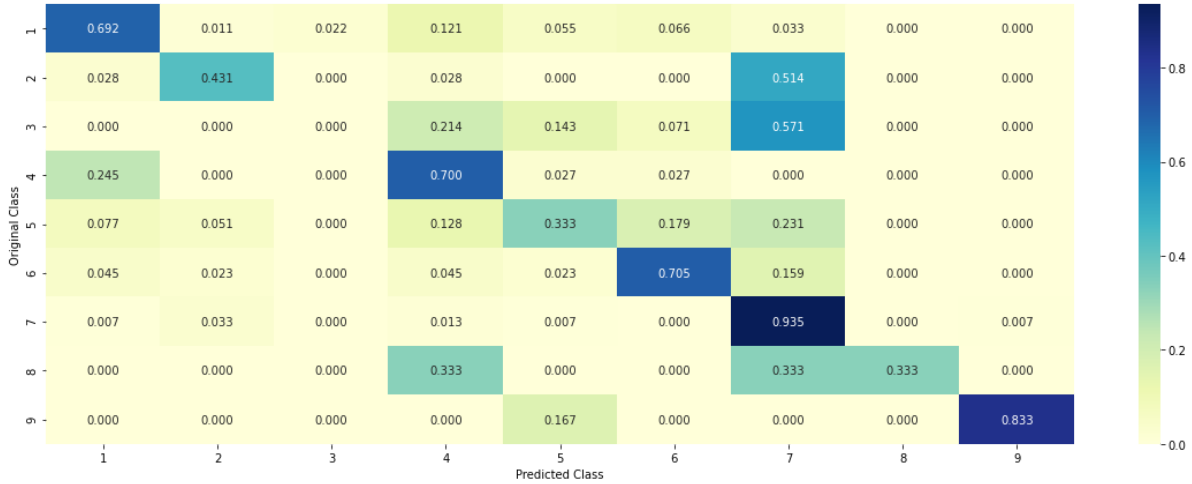
```
Log loss : 0.9529559265760372
Number of mis-classified points : 0.3157894736842105
-------------------- Confusion matrix --------------------
```




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




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



Out[91]: (0.9529559265760372, 0.3157894736842105)

4.3.2.3. Feature Importance, Correctly Classified point

In [92]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
clf.fit(train_x_tfidf,train_y)
test_point_index = 46
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0238 0.1304 0.0085 0.0173 0.0199 0.
0415 0.7497 0.0039 0.005 ]]
Actual Class : 7
--------------------------------------------------
267 Text feature [003] present in test data point [True]
349 Text feature [035] present in test data point [True]
356 Text feature [0003] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [93]:
```
test_point_index = 3
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0059 0.2296 0.0164 0.0204 0.0177 0.
0373 0.6657 0.0035 0.0035]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

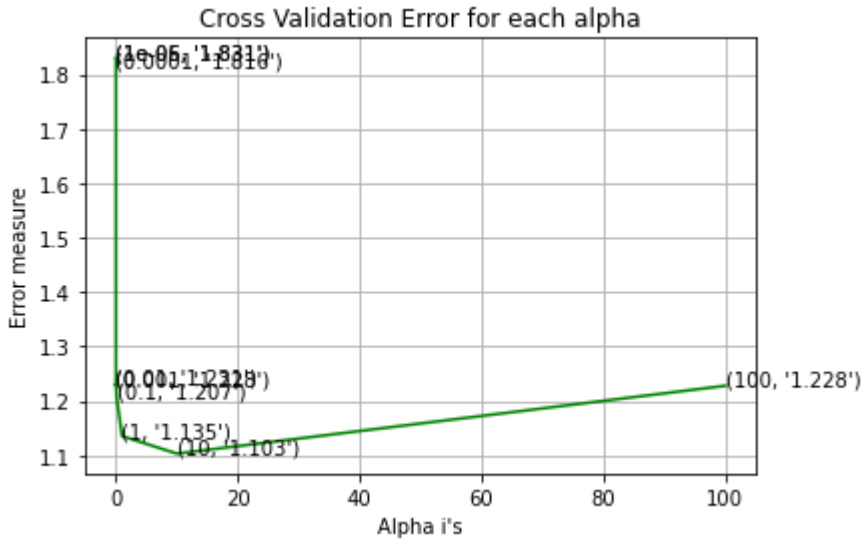
Logistic Regression CountVectorizer Unigram and Bigram without data balancing

In [94]:

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

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


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=
    clf.fit(train_x_onehotCoding_unibigram, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_unibigram, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_unibigram)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    # to avoid rounding error while multiplying probabilites we use log-
    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], pe
clf.fit(train_x_onehotCoding_unibigram, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_unibigram, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_unibigram)
log_reg_unibigram_train_logloss = log_loss(y_train, predict_y, labels=cl
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_onehotCoding_unibigram)
log_reg_unibigram_cv_logloss = log_loss(y_cv, predict_y, labels=clf.clas
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_onehotCoding_unibigram)
log_reg_unibigram_test_logloss = log_loss(y_test, predict_y, labels=clf.
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for alpha = 1e-06
Log Loss : 1.830889358572364
for alpha = 1e-05
Log Loss : 1.830889358572364
for alpha = 0.0001
Log Loss : 1.8161500692945696
for alpha = 0.001
Log Loss : 1.2278139711378306
for alpha = 0.01
Log Loss : 1.2313791725332983
for alpha = 0.1
Log Loss : 1.2073752898091663
for alpha = 1
Log Loss : 1.1352430817826227
for alpha = 10
Log Loss : 1.1030030973998817
for alpha = 100
Log Loss : 1.2277457961584488
```

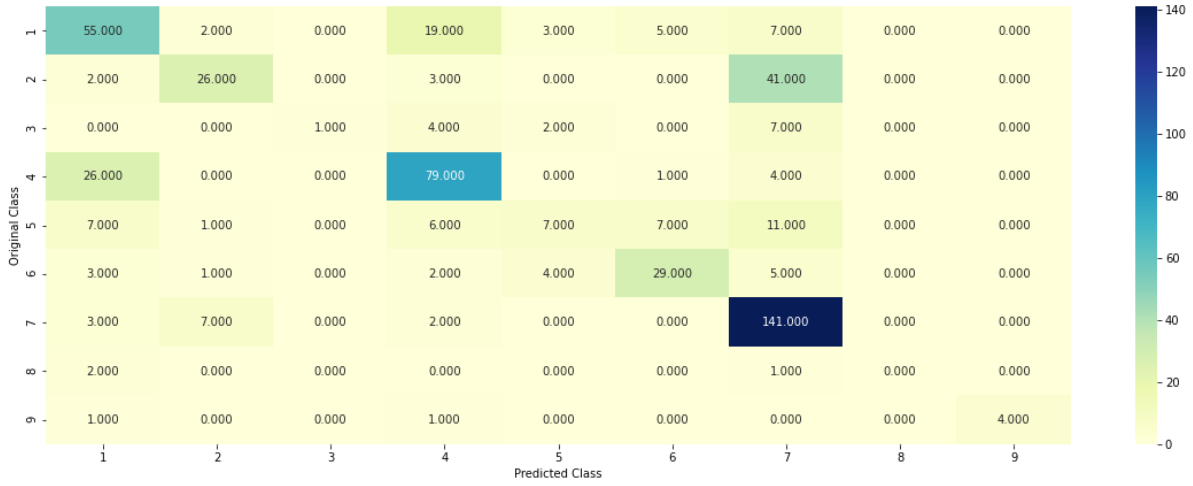



```
For values of best alpha =  10 The train log loss is: 0.8282299532701
297
For values of best alpha =  10 The cross validation log loss is: 1.10
20308462124042
```

```
For values of best alpha =  10 The test log loss is: 1.13452121394202
49
```

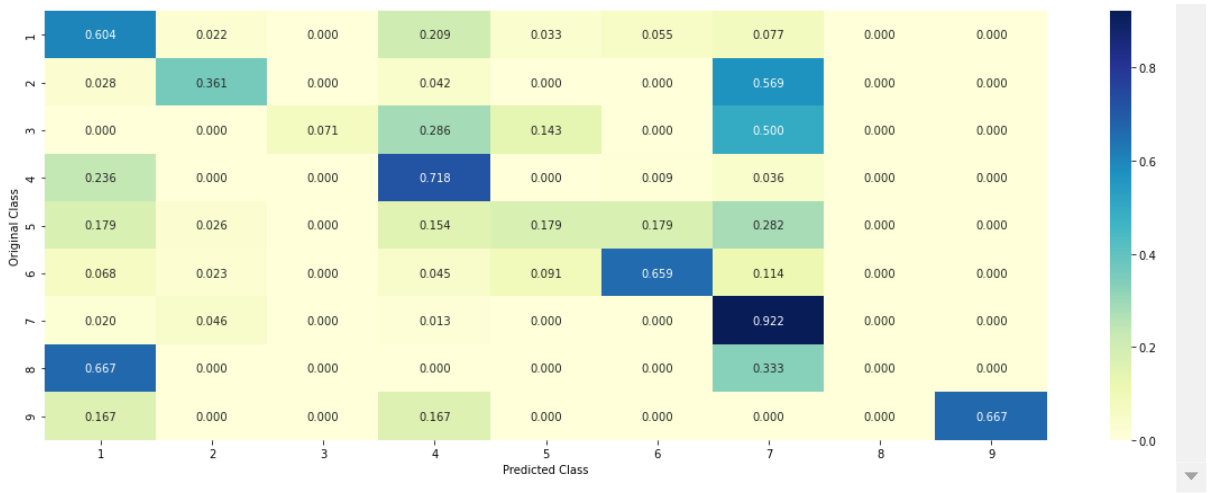
In [95]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-d
#-------------------------------
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
predict_and_plot_confusion_matrix(train_x_onehotCoding_unibigram, train_
```

```
Log loss : 1.1030030973998817
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```




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


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



Out[95]: (1.1030030973998817, 0.35714285714285715)

In [96]:
```
# from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
clf.fit(train_x_onehotCoding_unibigram,train_y)
test_point_index = 46
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_unibigram[test_point
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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]
```

```
Predicted Class : 7
Predicted Class Probabilities: [[5.000e-04 2.841e-01 0.000e+00 2.000e-
04 3.000e-04 9.730e-02 6.116e-01
  6.100e-03 0.000e+00]]
Actual Class : 7
--------------------------------------------------
```

In [97]:
```
test_point_index = 3
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_unibigram[test_point
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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]
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.000e-03 1.929e-01 5.400e-03 3.900e-
03 1.240e-02 9.900e-03 7.689e-01
  4.200e-03 5.000e-04]]
Actual Class : 7
--------------------------------------------------
```

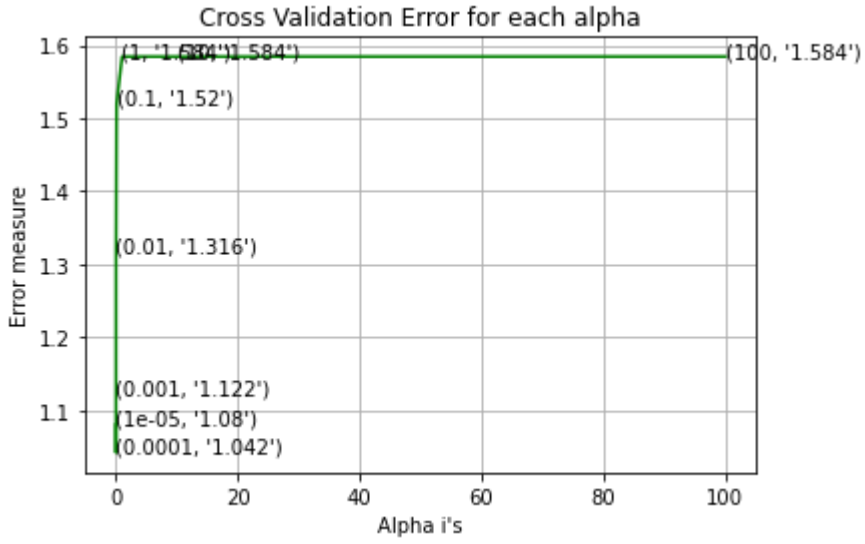
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [98]:
```
# read more about support vector machines with linear kernals here http:

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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='bala
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2',
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.c
    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], pe
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
lr_svm_train_logloss = log_loss(y_train, predict_y, labels=clf.classes_,
print('For values of best alpha = ', alpha[best_alpha], "The train log l

predict_y = sig_clf.predict_proba(cv_x_tfidf)
lr_svm_cv_logloss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1
print('For values of best alpha = ', alpha[best_alpha], "The cross valid

predict_y = sig_clf.predict_proba(test_x_tfidf)
lr_svm_test_logloss = log_loss(y_test, predict_y, labels=clf.classes_, e
print('For values of best alpha = ', alpha[best_alpha], "The test log lo
```

```
for C = 1e-05
Log Loss : 1.0802299529230226
for C = 0.0001
Log Loss : 1.041798939103503
for C = 0.001
Log Loss : 1.1219612472115477
for C = 0.01
Log Loss : 1.3163859811054737
for C = 0.1
Log Loss : 1.5196414100718385
for C = 1
Log Loss : 1.58444318481082
for C = 10
Log Loss : 1.5844431354213748
for C = 100
Log Loss : 1.5844431372663312
```

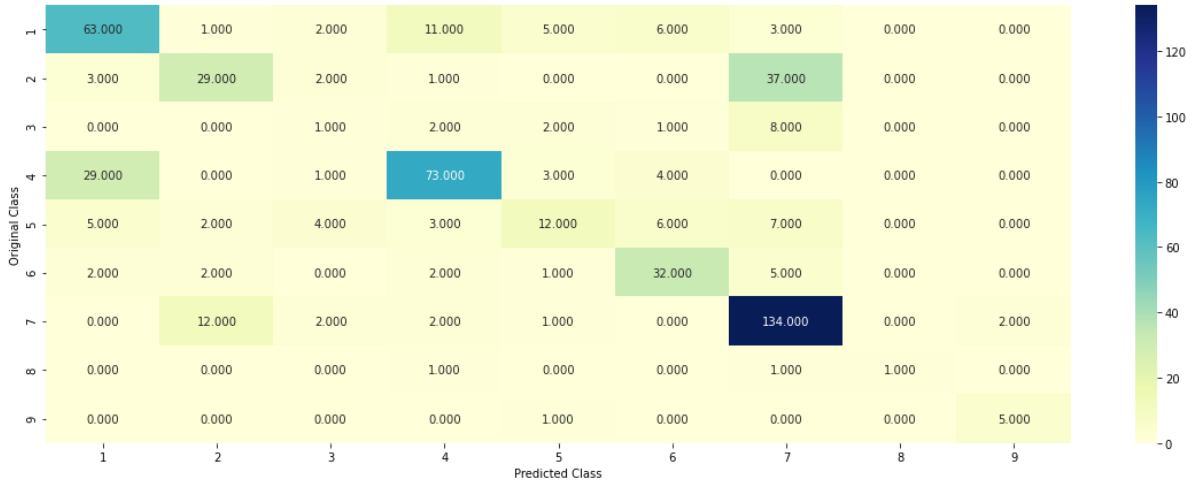



```
For values of best alpha =  0.0001 The train log loss is: 0.4456290457
8742296
For values of best alpha =  0.0001 The cross validation log loss is:
1.041798939103503
For values of best alpha =  0.0001 The test log loss is: 1.11610672910
7064
```

4.4.2. Testing model with best hyper parameters

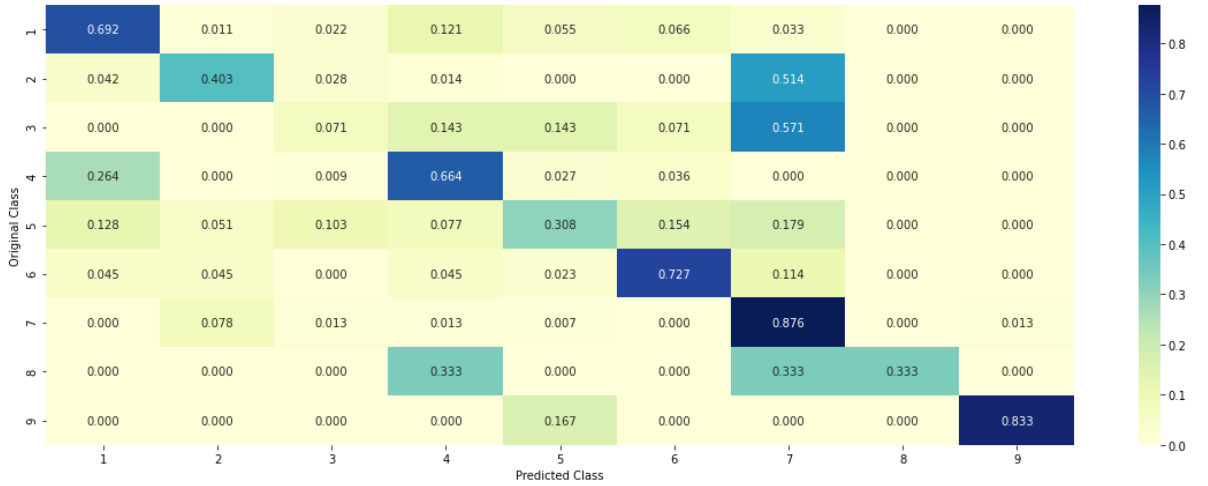
In [99]:
```
# read more about support vector machines with linear kernals here http:

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

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y
```

```
Log loss : 1.041798939103503
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```




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

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



Out[99]: (1.041798939103503, 0.34210526315789475)

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [100]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
clf.fit(train_x_tfidf,train_y)
test_point_index = 100
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5518 0.1379 0.0047 0.0077 0.024  0.
1115 0.1557 0.0047 0.002 ]]
Actual Class : 2
--------------------------------------------------
218 Text feature [01] present in test data point [True]
400 Text feature [021] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [101]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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],
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0519 0.2766 0.0044 0.0091 0.0408 0.
0655 0.5367 0.0057 0.0094]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [102]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gi
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, ra
# class_weight=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method
#
# 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', m
        clf.fit(train_x_tfidf, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidf, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=c
        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]).rav
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)), (feature
plt.grid()
plt.title("Cross Validation Error for each alpha")
```

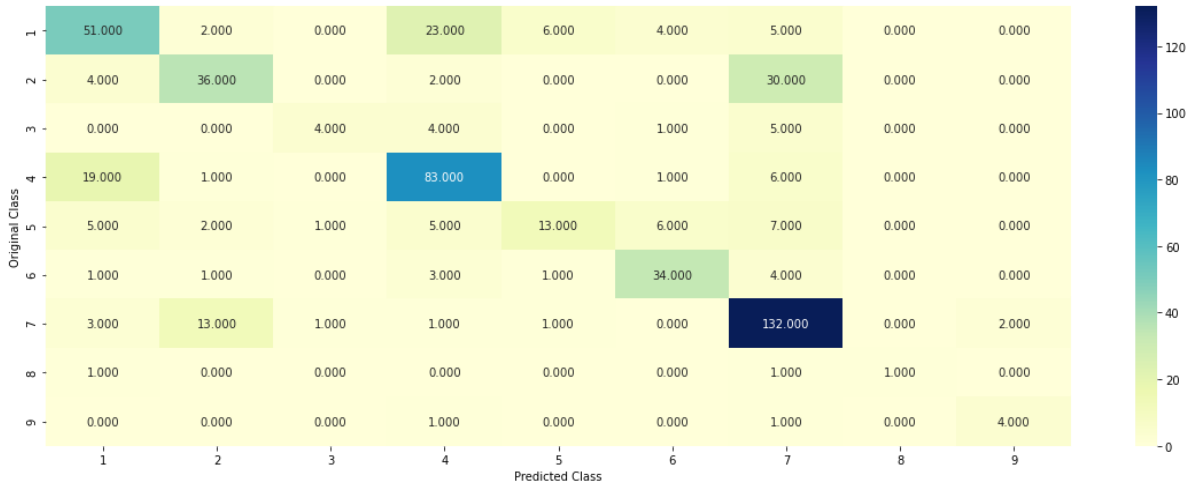
```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], crit
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf)
rf_train_logloss = log_loss(y_train, predict_y, labels=clf.classes_, eps
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The

predict_y = sig_clf.predict_proba(cv_x_tfidf)
rf_cv_logloss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The

predict_y = sig_clf.predict_proba(test_x_tfidf)
tf_test_logloss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.071444538764783
for n_estimators = 100 and max depth =  10
Log Loss : 1.0132452110026344
for n_estimators = 200 and max depth =  5
Log Loss : 1.0624334401958249
for n_estimators = 200 and max depth =  10
Log Loss : 1.0065874784698583
for n_estimators = 500 and max depth =  5
Log Loss : 1.0541866662006156
for n_estimators = 500 and max depth =  10
Log Loss : 1.0057357944380105
for n_estimators = 1000 and max depth =  5
Log Loss : 1.0510307451220782
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0053774405879758
for n_estimators = 2000 and max depth =  5
Log Loss : 1.0514537773432482
for n_estimators = 2000 and max depth =  10
Log Loss : 1.0058833252890418
For values of best estimator =  1000 The train log loss is: 0.51009508
50761513
For values of best estimator =  1000 The cross validation log loss is:
1.0053774405879758
For values of best estimator =  1000 The test log loss is: 1.086161958
2174893
```

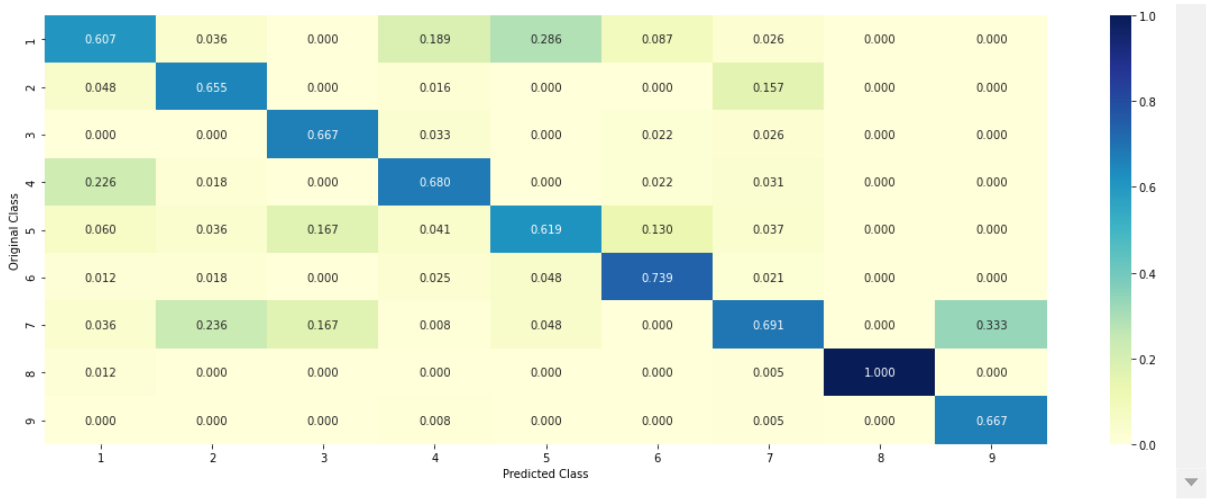
4.5.2. Testing model with best hyper parameters (TFIDF)

In [103]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gi
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, ra
# class_weight=None)

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], crit
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y
```

```
Log loss : 1.0053774405879758
Number of mis-classified points : 0.32706766917293234
-------------------- Confusion matrix --------------------
```




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




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

Out[103]: (1.0053774405879758, 0.32706766917293234)

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [104]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], crit
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0462 0.5563 0.0144 0.0426 0.0353 0.
0331 0.2625 0.0047 0.005 ]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [105]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
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_poi
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8384 0.0249 0.0131 0.0318 0.0329 0.
0253 0.0247 0.0047 0.0043]]
Actuall Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [106]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

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


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

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


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

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


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

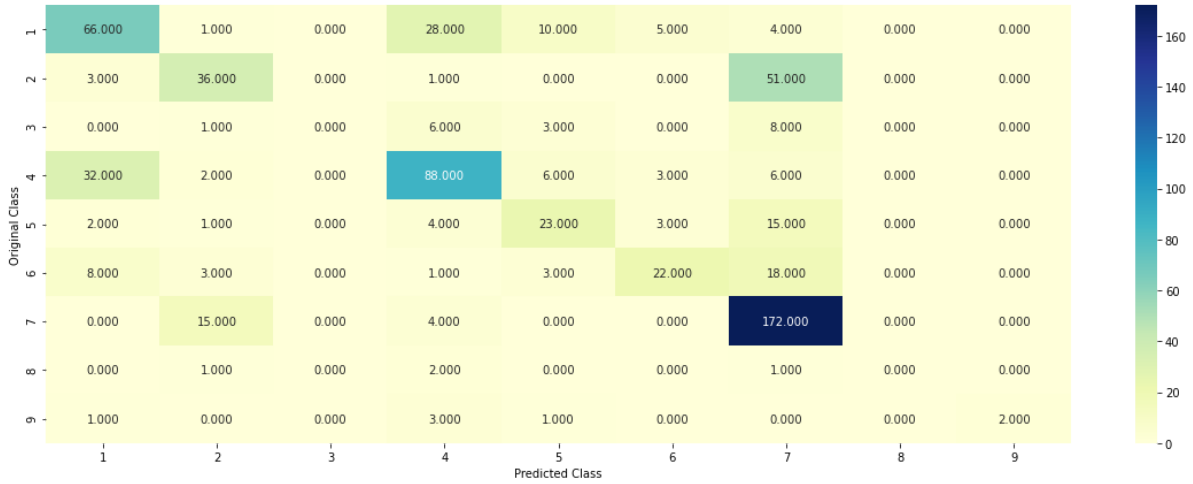
```
clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='
clf2.fit(train_x_tfidf, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

sig_clf1.fit(train_x_tfidf, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf
sig_clf2.fit(train_x_tfidf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_
sig_clf3.fit(train_x_tfidf, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict
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]
    sclf.fit(train_x_tfidf, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.02
Support vector machines : Log Loss: 1.58
Naive Bayes : Log Loss: 1.19
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.819
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.733
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.365
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.135
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.185
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.219
```

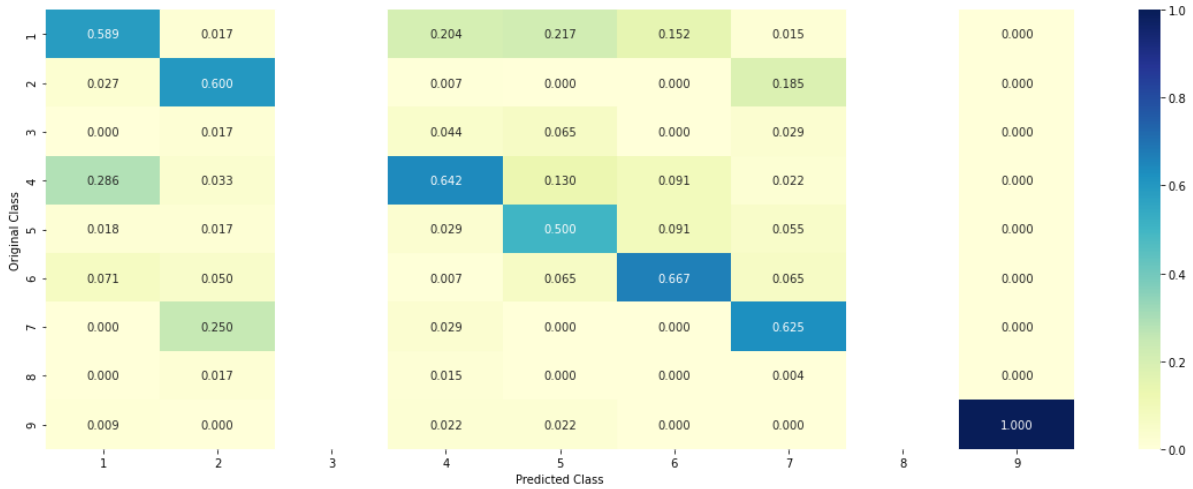
4.7.2 testing the model with the best hyper parameters

In [107]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], me
sclf.fit(train_x_tfidf, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_tfidf))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_tfidf))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of missclassified point :", np.count_nonzero((sclf.predict
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_tfidf
```

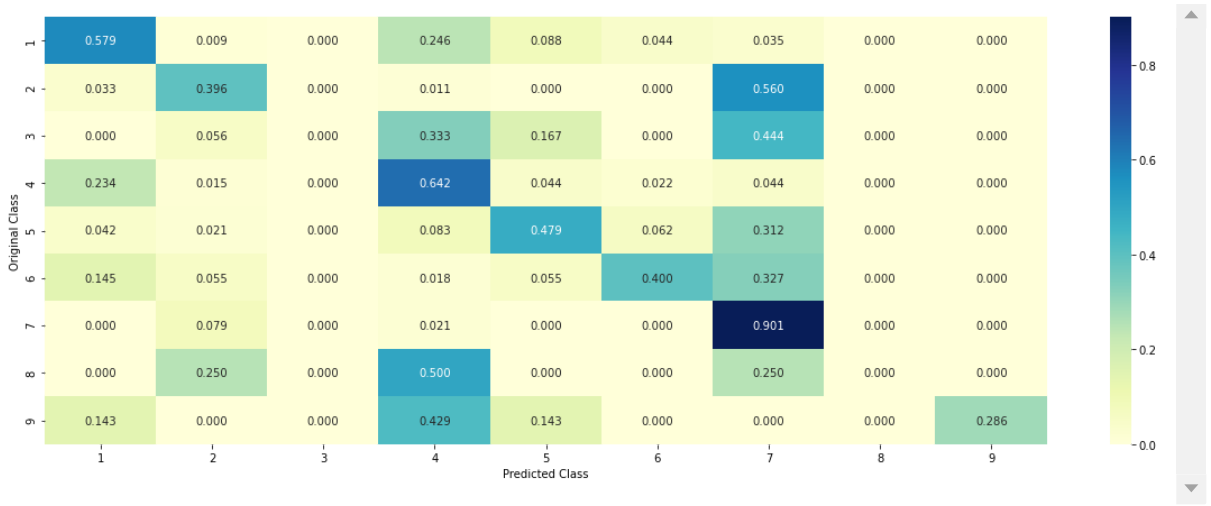
```
Log loss (train) on the stacking classifier : 0.5720018203344271
Log loss (CV) on the stacking classifier : 1.1350183186072773
Log loss (test) on the stacking classifier : 1.172812903879872
Number of missclassified point : 0.3849624060150376
-------------------- Confusion matrix --------------------
```




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



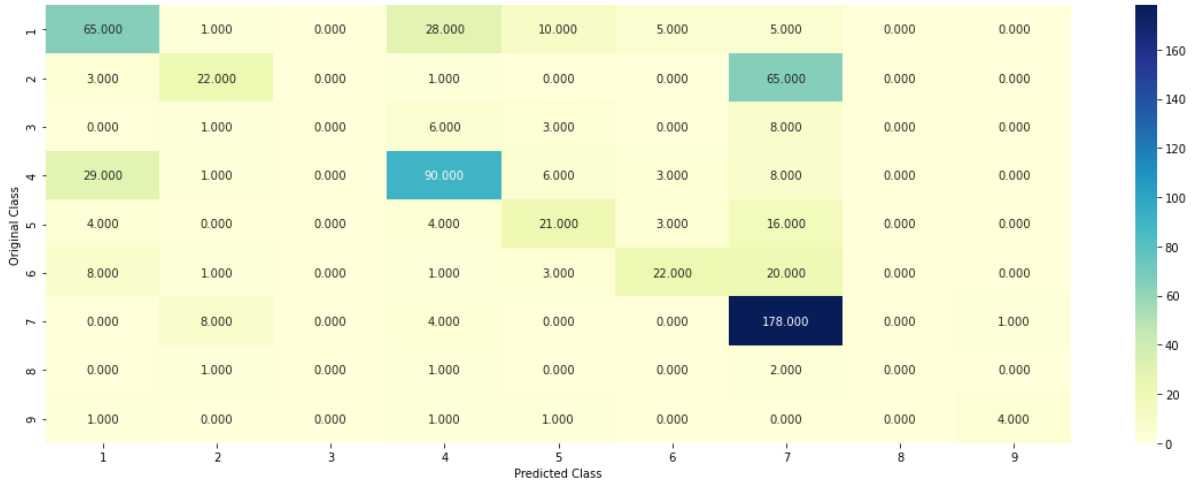

```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.7.3 Maximum Voting classifier

In [108]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2),
vclf.fit(train_x_tfidf, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vc
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.pre
log_error_voting = log_loss(test_y, vclf.predict_proba(test_x_tfidf))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf
print("Number of missclassified point :", np.count_nonzero((vclf.predict
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidf
```

```
Log loss (train) on the VotingClassifier : 0.8745983162727653
Log loss (CV) on the VotingClassifier : 1.186777172725942
Log loss (test) on the VotingClassifier : 1.2045364566874455
Number of missclassified point : 0.3954887218045113
-------------------- Confusion matrix --------------------
```




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


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

Type *Markdown* and LaTeX: α^2

Conclusion

In [109]:
```
from prettytable import PrettyTable

# Specify the Column Names while initializing the Table
myTable = PrettyTable(["Vectorization", "Model", "Log Loss"])

# Add rows
myTable.add_row(["TFIDF", "Naive Bayes", nb_test_logloss])
myTable.add_row(["TFIDF", "KNN", knn_test_logloss])
myTable.add_row(["TFIDF", "Logistic Regression with Class Balancing", lo
myTable.add_row(["CountVectorizer(Uni-Bigram)", "Logistic Regression wit
myTable.add_row(["CountVectorizer(Uni-Bigram)", "Logistic Regression wit
myTable.add_row(["TFIDF (Uni,Bigram to Quadgram)", "Logistic Regression
myTable.add_row(["TFIDF", "Linear SVM", lr_svm_test_logloss])
myTable.add_row(["TFIDF", "Random Forest", tf_test_logloss])
myTable.add_row(["TFIDF", "Stacked Model", log_error])
myTable.add_row(["TFIDF", "Majority Voting Classifier", log_error_voting


print(myTable)
```

```
+---------------------------------+-----------------------------------
---------+--------------------+
|         Vectorization           |                    Model
|      Log Loss      |
+---------------------------------+-----------------------------------
---------+--------------------+
|             TFIDF               |                 Naive Bayes
| 1.2118257462881639 |
|             TFIDF               |                    KNN
| 1.2312924708430761 |
|             TFIDF               |   Logistic Regression with Class Ba
lancing  | 1.0088153202193393 |
|  CountVectorizer(Uni-Bigram)    |   Logistic Regression with Class Ba
lancing  | 1.1345212139420249 |
|  CountVectorizer(Uni-Bigram)    | Logistic Regression without class b
alancing | 1.1345212139420249 |
| TFIDF (Uni,Bigram to Quadgram) |   Logistic Regression with class ba
lancing  | 0.9467560545012379 |
|             TFIDF               |                  Linear SVM
| 1.116106729107064  |
|             TFIDF               |                Random Forest
| 1.0861619582174893 |
|             TFIDF               |                Stacked Model
| 1.172812903879872  |
|             TFIDF               |          Majority Voting Classifier
| 1.2045364566874455 |
+---------------------------------+-----------------------------------
---------+--------------------+
```