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 c
lassified into => Multi class classification problem
```

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [2]:

```
import warnings
warnings.filterwarnings('ignore')

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [5]:

```
data = pd.read_csv('training_variants.csv')
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[5]:

	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 [6]:

```
# note the seprator in this file
data_text =pd.read_csv("training_text.csv",sep="\|\|",engine="python",names=|
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[6]:

	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 [7]:

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


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

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

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

In [8]:

```
#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, "se
```

```
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 : 31.828341999999992 seco
nds
```

In [9]:

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

Out[9]:

	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 [10]:

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

Out[10]:

	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 [11]:

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

In [12]:

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

Out[12]:

	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 [13]:

```
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 outp
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify
# split the train data into train and cross validation by maintaining same di
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=
```

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

In [14]:

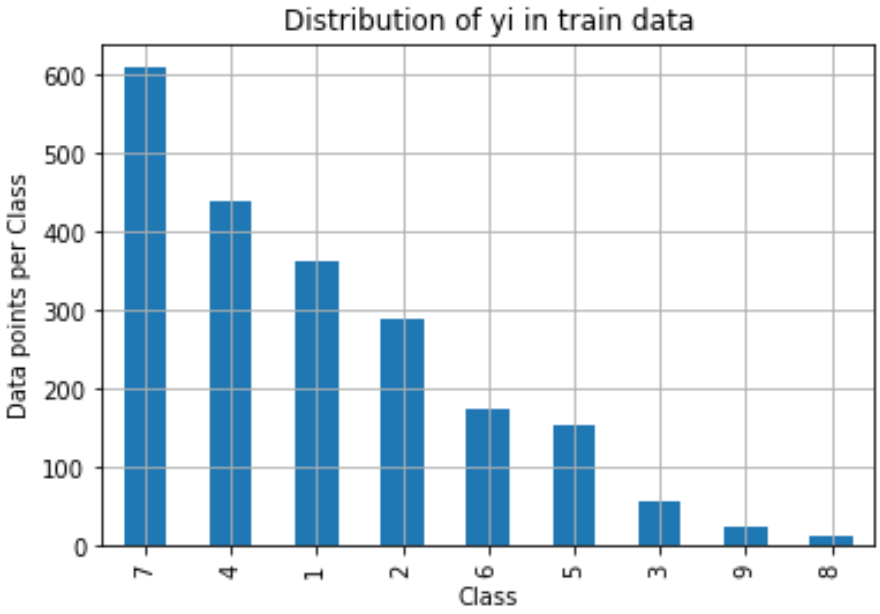
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

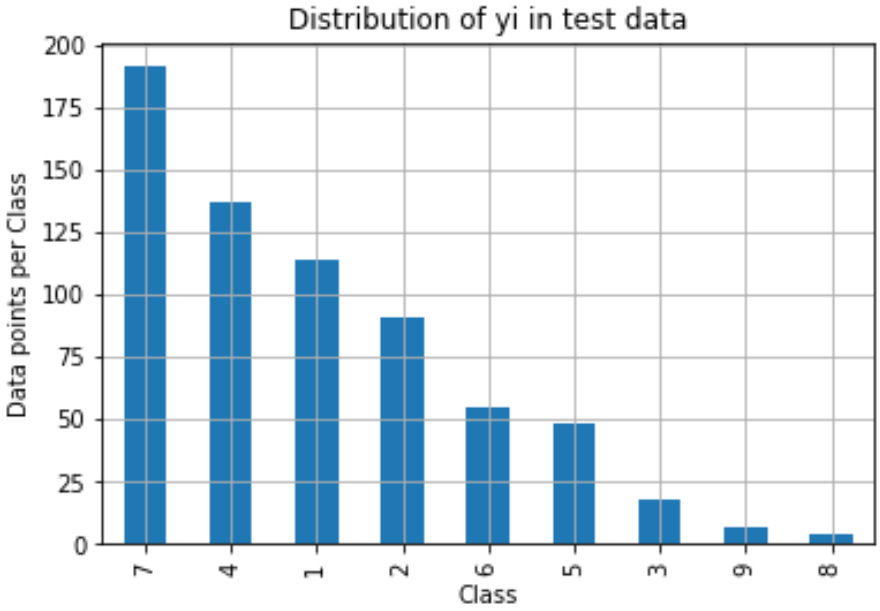
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [16]:

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

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

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


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

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

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

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

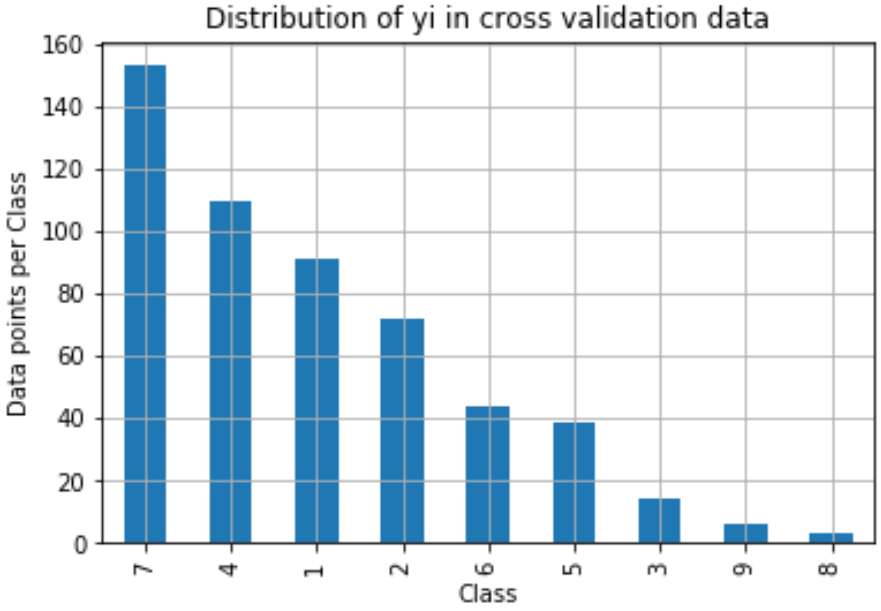



```
Number of data points in class 1 : 609 ( 28.672 %)
Number of data points in class 2 : 439 ( 20.669 %)
Number of data points in class 3 : 363 ( 17.09 %)
Number of data points in class 4 : 289 ( 13.606 %)
Number of data points in class 5 : 176 ( 8.286 %)
Number of data points in class 6 : 155 ( 7.298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




```
Number of data points in class 1 : 191 ( 28.722 %)
Number of data points in class 2 : 137 ( 20.602 %)
Number of data points in class 3 : 114 ( 17.143 %)
Number of data points in class 4 : 91 ( 13.684 %)
Number of data points in class 5 : 55 ( 8.271 %)
Number of data points in class 6 : 48 ( 7.218 %)
```

```
Number of data points in class 7 : 18 ( 2.707 %)
Number of data points in class 8 : 7 ( 1.053 %)
Number of data points in class 9 : 4 ( 0.602 %)
------------------------------------------------------------
--------------------
```




```
Number of data points in class 1 : 153 ( 28.759 %)
Number of data points in class 2 : 110 ( 20.677 %)
Number of data points in class 3 : 91 ( 17.105 %)
Number of data points in class 4 : 72 ( 13.534 %)
Number of data points in class 5 : 44 ( 8.271 %)
Number of data points in class 6 : 39 ( 7.331 %)
Number of data points in class 7 : 14 ( 2.632 %)
Number of data points in class 8 : 6 ( 1.128 %)
Number of data points in class 9 : 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 [17]:

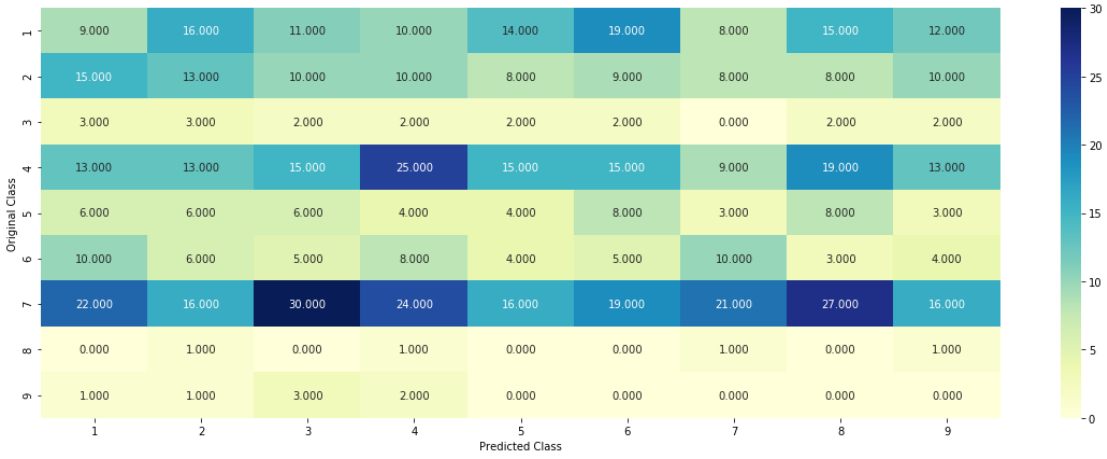
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i

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

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

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

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

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

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

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

```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [18]:

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

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


# 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_predict

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 2.5418828
38280539
Log loss on Test Data using Random Model 2.507511774835214
-------------------- Confusion matrix --------------------
```




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

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

3.3 Univariate Analysis

In [19]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature ir
# build a vector (1*9) , the first element = (number of times it occured in c
# gv_dict is like a look up table, for every gene it store a (1*9) representa
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for
    gv_dict = dict()
```

```
    # denominator will contain the number of time that particular feature occ
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']==
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1               S1715C      1
            # 2486  2486  BRCA1               S1841R      1
            # 2614  2614  BRCA1                  M1R      1
            # 2432  2432  BRCA1               L1657P      1
            # 2567  2567  BRCA1               T1685A      1
            # 2583  2583  BRCA1               E1660G      1
            # 2634  2634  BRCA1               W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

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

        # we are adding the gene/variation to the dict as key and vec as valu
        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, 0.068181818181
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

In [20]:

```
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 : 232
BRCA1     167
TP53      108
EGFR       86
PTEN       84
BRCA2      72
BRAF       62
KIT        61
ERBB2      43
ALK        41
PDGFRA     37
Name: Gene, dtype: int64
```

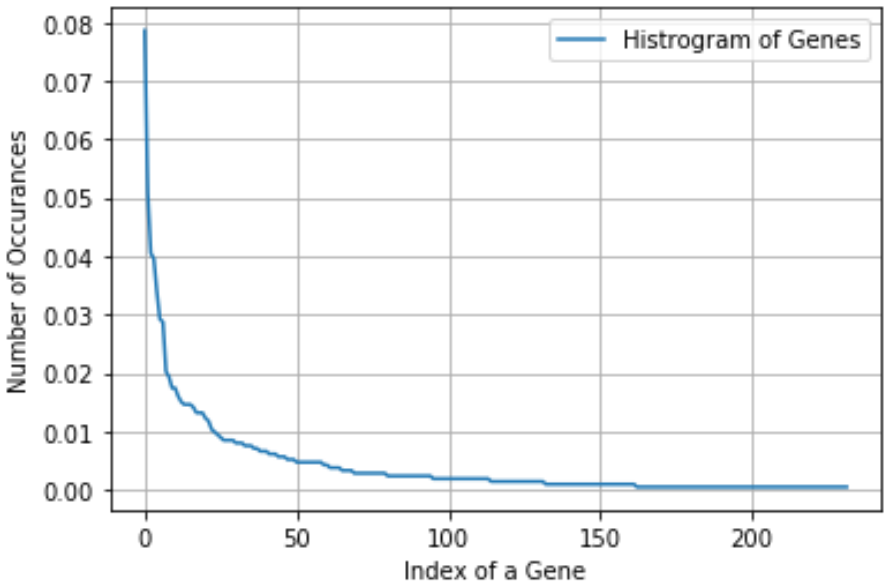
In [21]:

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

```
Ans: There are 232 different categories of genes in the train
data, and they are distibuted as follows
```

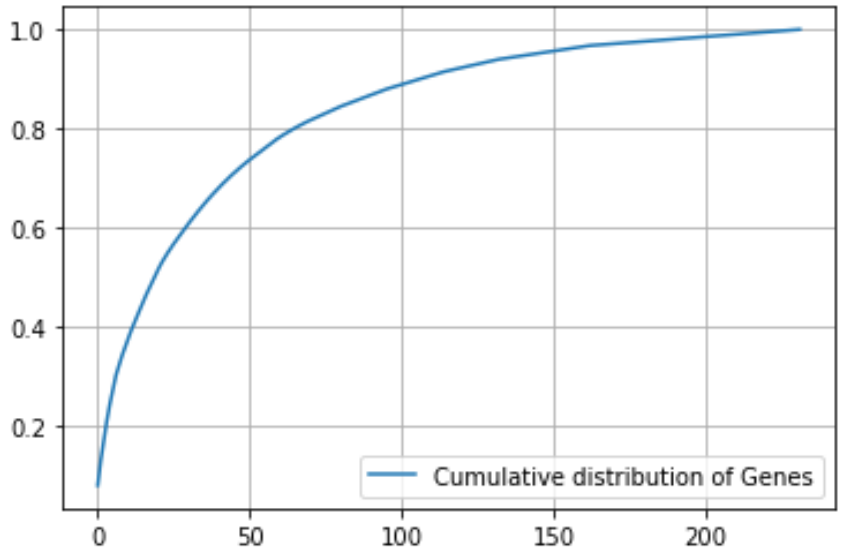
In [22]:

```
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 [23]:

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



In [39]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Ger
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [90]:

```
gene_vectorizer_ohe = CountVectorizer(min_df=10,ngram_range=(1,4))
train_gene_feature_ohe = gene_vectorizer_ohe.fit_transform(train_df['TEXT'])
test_gene_feature_ohe = gene_vectorizer_ohe.transform(test_df['TEXT'])
cv_gene_feature_ohe = gene_vectorizer_ohe.transform(cv_df['TEXT'])
```

In [40]:

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

Out[40]:

```
2068      TET2
2040    MAP2K2
1871      MTOR
2405       NF1
198       EGFR
Name: Gene, dtype: object
```

In [41]:

```
gene_vectorizer.get_feature_names()
```

```
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
```

In [42]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot enc
```

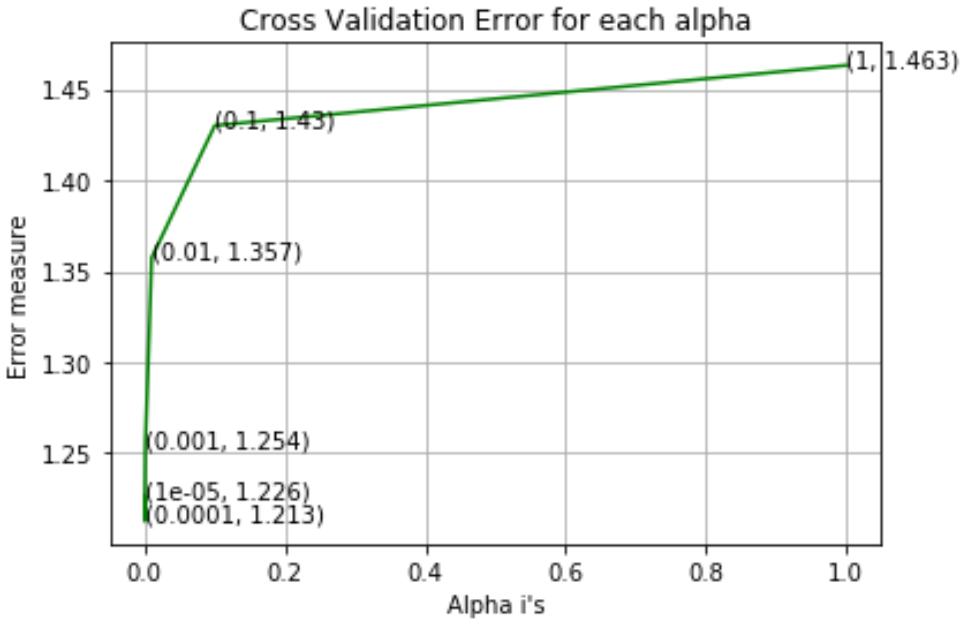
```
train_gene_feature_onehotCoding is converted feature using one
-hot encoding method. The shape of gene feature: (2124, 231)
```

In [29]:

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

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

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

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


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

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


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

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

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
```

```
For values of alpha =  1e-05 The log loss is: 1.22579700815685
77
For values of alpha =  0.0001 The log loss is: 1.2127005622548
805
For values of alpha =  0.001 The log loss is: 1.25414901381896
3
For values of alpha =  0.01 The log loss is: 1.357415884565039
3
For values of alpha =  0.1 The log loss is: 1.4301042681326697
For values of alpha =  1 The log loss is: 1.4631671863044364
```




```
For values of best alpha =  0.0001 The train log loss is: 0.98
98487354730678
For values of best alpha =  0.0001 The cross validation log lo
ss is: 1.2127005622548805
For values of best alpha =  0.0001 The test log loss is: 1.175
9599164408787
```

In [30]:

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

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(t
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 b
y the  232  genes in train dataset?
Ans
1. In test data 635 out of 665 : 95.48872180451127
2. In cross validation data 520 out of  532 : 97.7443609022556
4
```

3.2.2 Univariate Analysis on Variation Feature

In [31]:

```
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 : 1922
Truncating_Mutations    61
Deletion                46
Amplification           45
Fusions                 22
E17K                     3
Q61L                     3
T58I                     3
Overexpression           3
Q61H                     3
G67R                     2
Name: Variation, dtype: int64
```

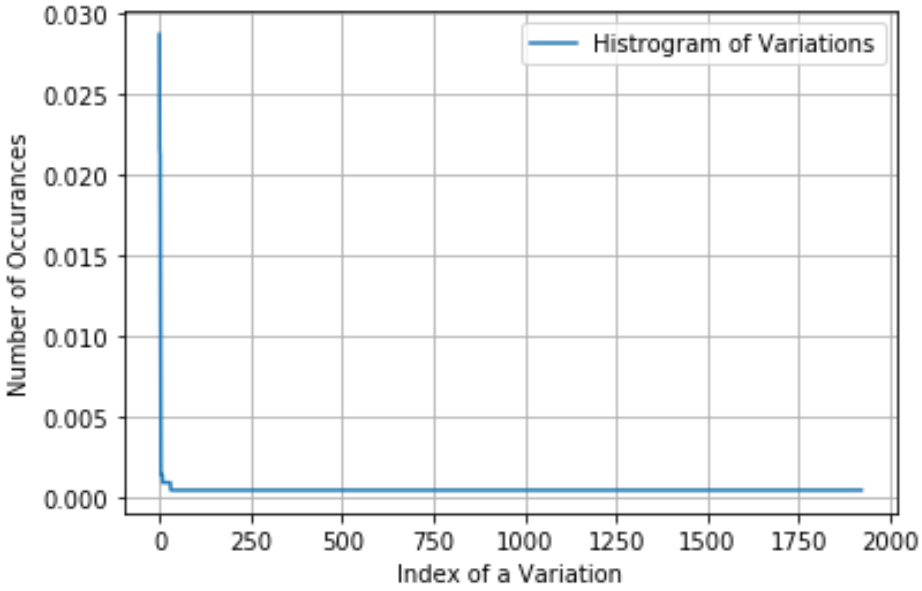
In [32]:

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

```
Ans: There are 1922 different categories of variations in the
train data, and they are distibuted as follows
```

In [33]:

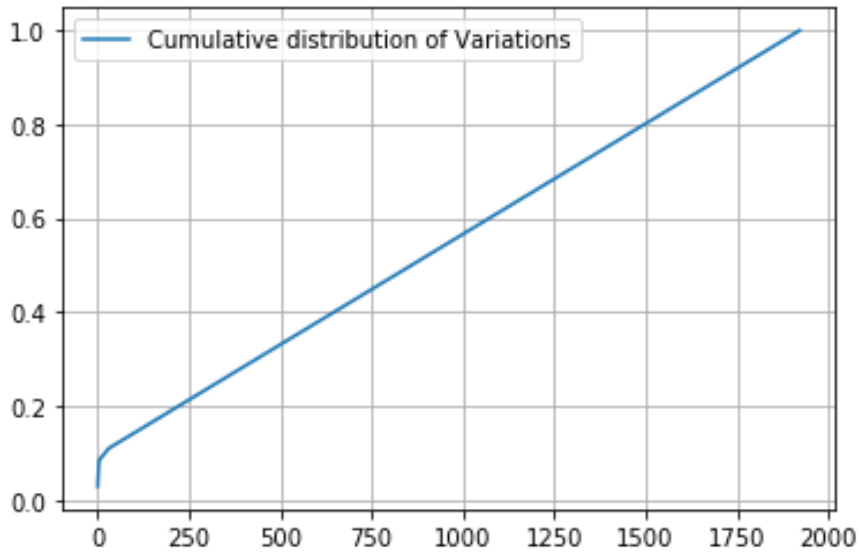
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [34]:

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

```
[0.0287194  0.05037665 0.07156309 ... 0.99905838 0.99952919 1.
]
```



In [35]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(tra
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df|
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Var
```

In [36]:

```
print("train_variation_feature_onehotEncoded is converted feature using the c
```

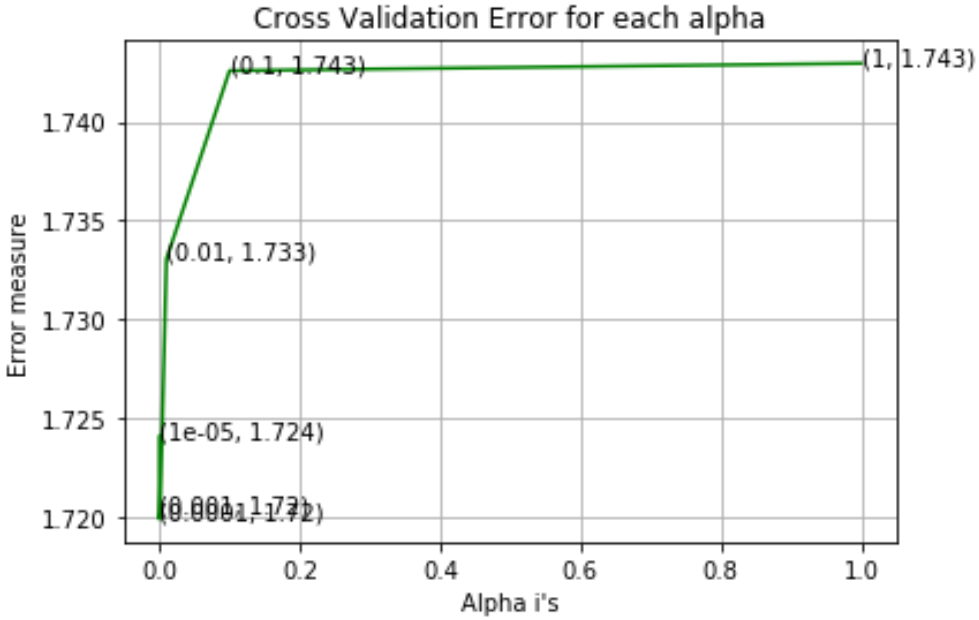
```
train_variation_feature_onehotEncoded is converted feature usi
ng the onne-hot encoding method. The shape of Variation featur
e: (2124, 1951)
```

In [37]:

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

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

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

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


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The cross validatior
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
```

```
For values of alpha =  1e-05 The log loss is: 1.72403508613912
74
For values of alpha =  0.0001 The log loss is: 1.7198599095871
534
For values of alpha =  0.001 The log loss is: 1.72021337564112
73
For values of alpha =  0.01 The log loss is: 1.733038945040417
For values of alpha =  0.1 The log loss is: 1.742585507916301
For values of alpha =  1 The log loss is: 1.7429600341471403
```




```
For values of best alpha =  0.0001 The train log loss is: 0.69
30602917010352
For values of best alpha =  0.0001 The cross validation log lo
ss is: 1.7198599095871534
For values of best alpha =  0.0001 The test log loss is: 1.695
5586591329577
```

In [38]:

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

```
Q12. How many data points are covered by total  1922  genes in
test and cross validation data sets?
Ans
1. In test data 68 out of 665 : 10.225563909774436
2. In cross validation data 51 out of  532 : 9.586466165413533
```

3.2.3 Univariate Analysis on Text Feature

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

In [0]:

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

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

In [0]:

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

In [43]:

```
# building a CountVectorizer with all the words that occured minimum 3 times
text_vectorizer = TfidfVectorizer(max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
# 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 retur
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 2000
```

In [0]:

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

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


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

In [0]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [0]:

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

In [47]:

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

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

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

In [91]:

```
train_gene_feature_ohe =  normalize(train_gene_feature_ohe, axis=0)
test_gene_feature_ohe =  normalize(test_gene_feature_ohe, axis=0)
cv_gene_feature_ohe =  normalize(cv_gene_feature_ohe, axis=0)
```

In [95]:

```
train_text_ohe =  train_gene_feature_ohe
test_text_ohe =  test_gene_feature_ohe
cv_text_ohe =  cv_gene_feature_ohe
```

In [96]:

```
test_text_ohe.shape
```

Out[96]:

```
(665, 600046)
```

In [48]:

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

In [49]:

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

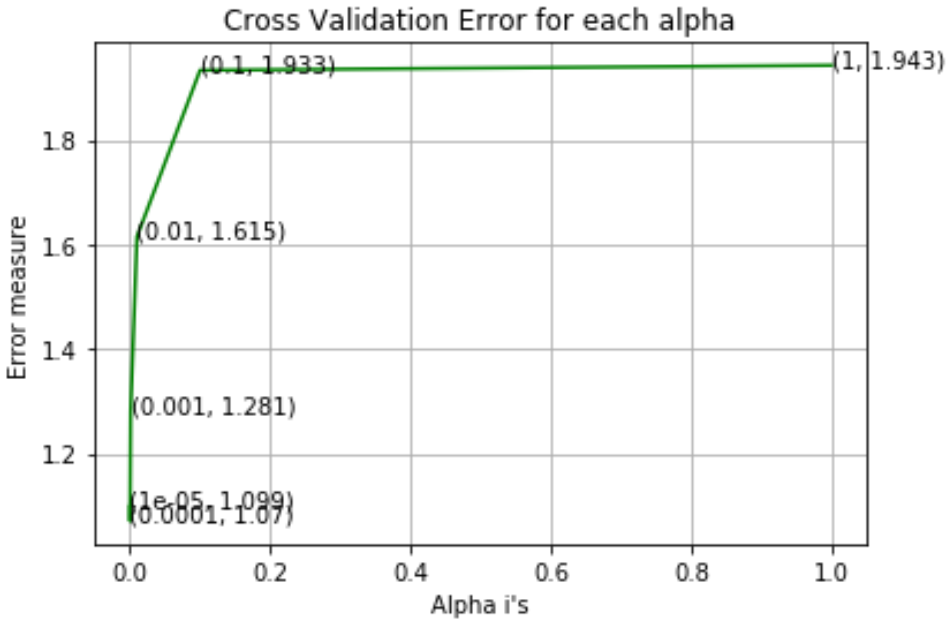
```
Counter({209.50706920759723: 1, 141.56312749469848: 1, 122.
53245912745577: 1, 108.55278293745872: 1, 101.9680631881371
7: 1, 97.70606690098126: 1, 97.15826089414246: 1, 96.883302
75631209: 1, 93.84560312425474: 1, 89.11383309698658: 1, 8
8.88053223203686: 1, 80.6639206008064: 1, 76.6845954634642
3: 1, 76.18907741003821: 1, 73.13018162557101: 1, 71.244044
28837198: 1, 68.04298175979599: 1, 66.82700950669272: 1, 6
5.12336496665485: 1, 64.37010120105118: 1, 62.6608594010185
56: 1, 61.20576840033799: 1, 57.80507834806128: 1, 55.66032
000842091: 1, 54.62208682925938: 1, 54.54785670829584: 1, 5
4.54577535210473: 1, 54.490892781888924: 1, 53.961330590057
3: 1, 53.64281973311552: 1, 52.46344295432083: 1, 52.289689
53575397: 1, 52.253179583353656: 1, 51.78791097740928: 1, 4
8.88977656975987: 1, 48.27681367110115: 1, 47.9593042472309
46: 1, 47.10033450780168: 1, 45.91122958572895: 1, 43.25033
052699735: 1, 42.88712313676894: 1, 42.233631957745246: 1,
41.73943580566862: 1, 41.28430196197292: 1, 40.475274209911
355: 1, 40.30943701164794: 1, 38.937701623062196: 1, 38.787
90279498007: 1, 38.5651910212831: 1, 38.17716697287364: 1,
```

In [50]:

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

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

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
```

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

```
For values of alpha =  1e-05 The log loss is: 1.09857668946424
48
For values of alpha =  0.0001 The log loss is: 1.0703580824181
05
For values of alpha =  0.001 The log loss is: 1.28107176639265
59
For values of alpha =  0.01 The log loss is: 1.614764807997464
4
For values of alpha =  0.1 The log loss is: 1.9334432298777866
For values of alpha =  1 The log loss is: 1.9428270023150276
```




```
For values of best alpha =  0.0001 The train log loss is: 0.70
06356986827159
For values of best alpha =  0.0001 The cross validation log lo
ss is: 1.070358082418105
For values of best alpha =  0.0001 The test log loss is: 1.112
5137100569935
```

In [51]:

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

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

In [52]:

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

```
6.625 % of word of test data appeared in train data
7.75 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [53]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [54]:

```
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 [55]:

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

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

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

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

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

Stacking the three types of features

In [56]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_v
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_vari
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_

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

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

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

In [97]:

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

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

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

In [57]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_
print("(number of data points * number of features) in test data = ", test_x_
print("(number of data points * number of features) in cross validation data
```

```
One hot encoding features :
(number of data points * number of features) in train data =
(2124, 4182)
(number of data points * number of features) in test data =
(665, 4182)
(number of data points * number of features) in cross validati
on data = (532, 4182)
```

In [98]:

```
print("One hot encoding features :")
print("train data with text one hot encoding = ", train_x_ohe.shape)
print("test data with text one hot encoding = ", test_x_ohe.shape)
print("cv data with text one hot encoding = ", cv_x_ohe.shape)
```

```
One hot encoding features :
train data with text one hot encoding =  (2124, 602228)
test data with text one hot encoding =  (665, 602228)
cv data with text one hot encoding =  (532, 602228)
```

4.1. Base Line Model

4.1.1. Naive Bayes

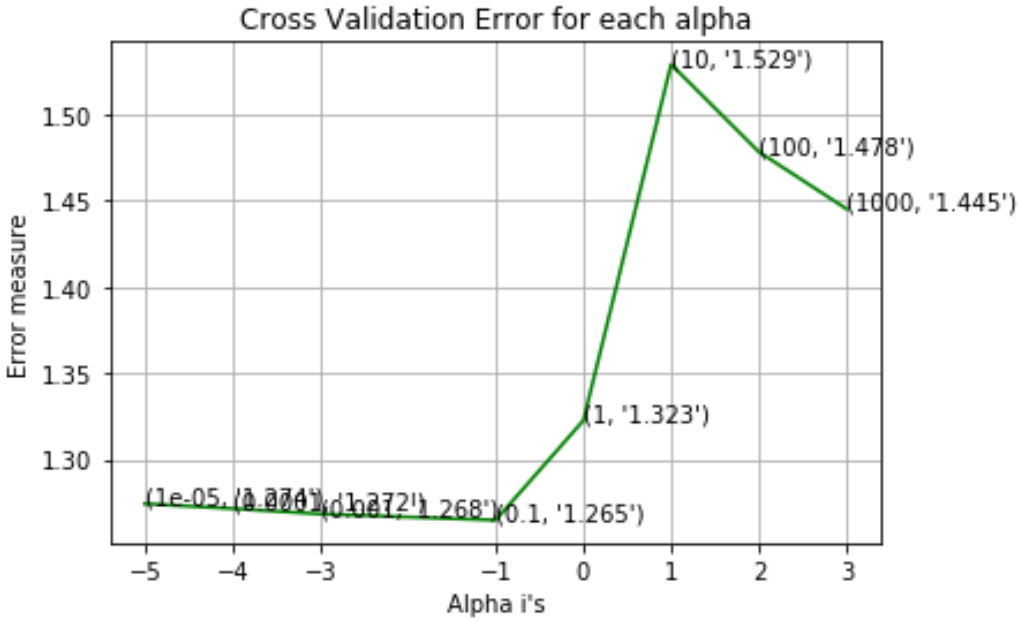
4.1.1.1. Hyper parameter tuning

In [59]:

```
# find more about Multinomial Naive base function here http://scikit-learn.or
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Nc

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test vector
# ----------------------


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


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

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

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


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

```
for alpha = 1e-05
Log Loss : 1.2743597695503905
for alpha = 0.0001
Log Loss : 1.27155570961278
for alpha = 0.001
Log Loss : 1.268454411076731
for alpha = 0.1
Log Loss : 1.2648538650221404
for alpha = 1
Log Loss : 1.3225756913642526
for alpha = 10
Log Loss : 1.5289709780922058
for alpha = 100
Log Loss : 1.4782932961927002
for alpha = 1000
Log Loss : 1.4451799203568823
```



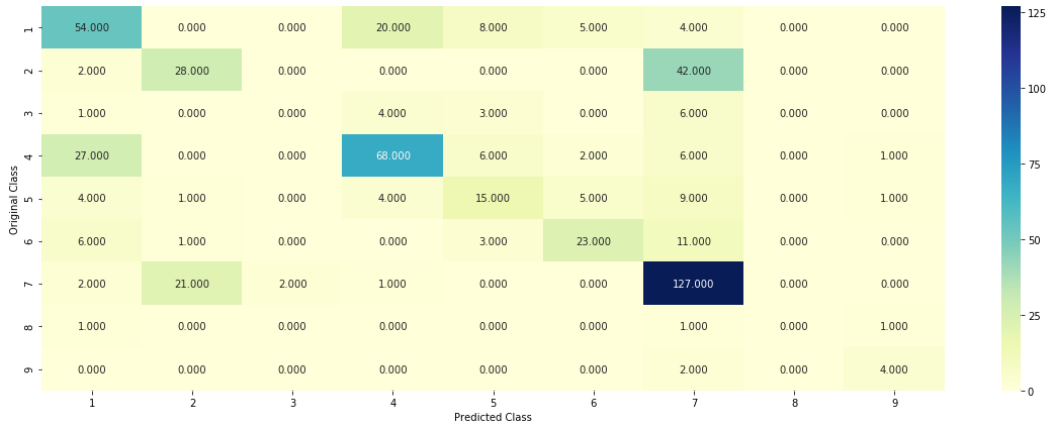
```
For values of best alpha =  0.1 The train log loss is: 0.66356
38085894525
For values of best alpha =  0.1 The cross validation log loss
is: 1.2648538650221404
For values of best alpha =  0.1 The test log loss is: 1.216572
8778072211
```

4.1.1.2. Testing the model with best hyper paramters

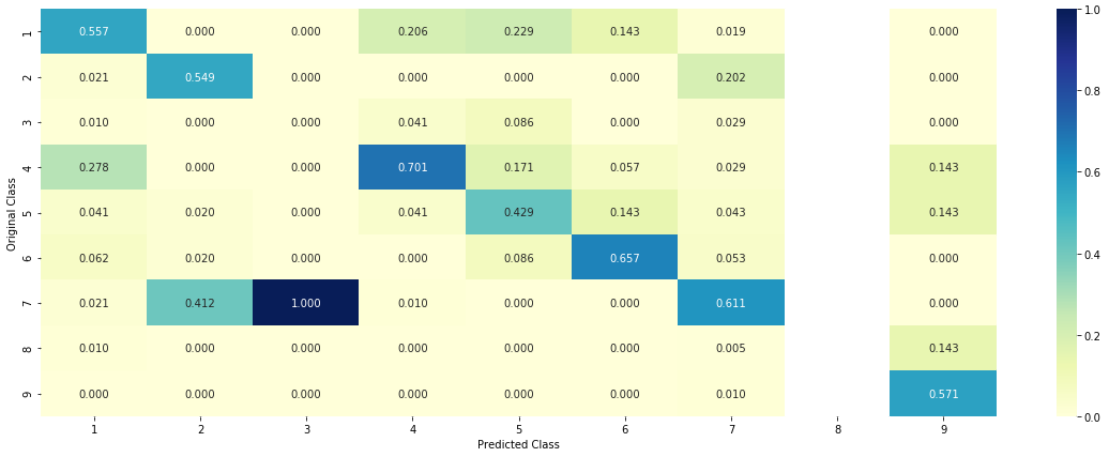
In [60]:

```
# find more about Multinomial Naive base function here http://scikit-learn.or
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Nc

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test vector
# ----------------------


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

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

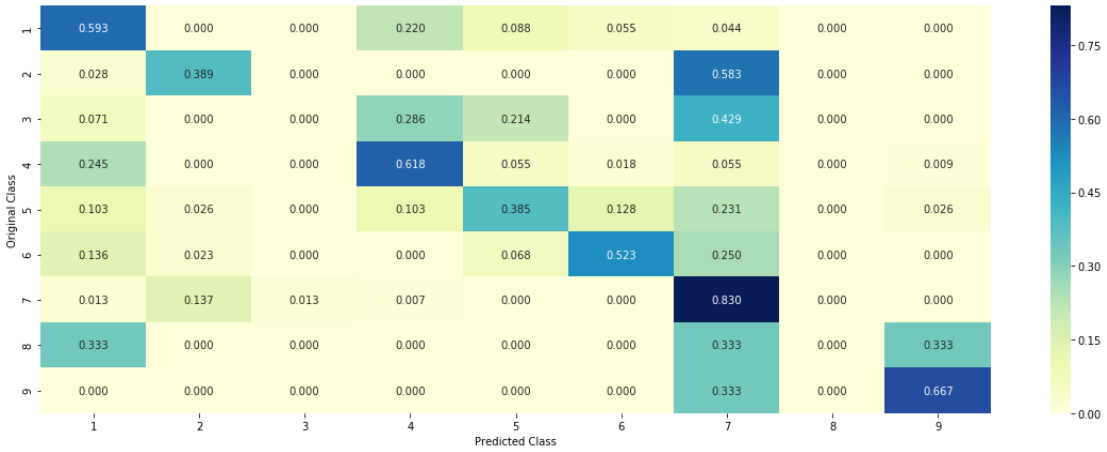
```
Log Loss : 1.2648538650221404
Number of missclassified point : 0.40037593984962405
-------------------- Confusion matrix --------------------
```



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




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



4.1.1.3. Feature Importance, Correctly classified point

In [61]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.062  0.5994 0.0193 0.0769
0.042  0.0475 0.1429 0.006  0.004 ]]
Actual Class : 5
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [62]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1683 0.0442 0.017  0.6205
0.0382 0.042  0.061  0.0052 0.0036]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

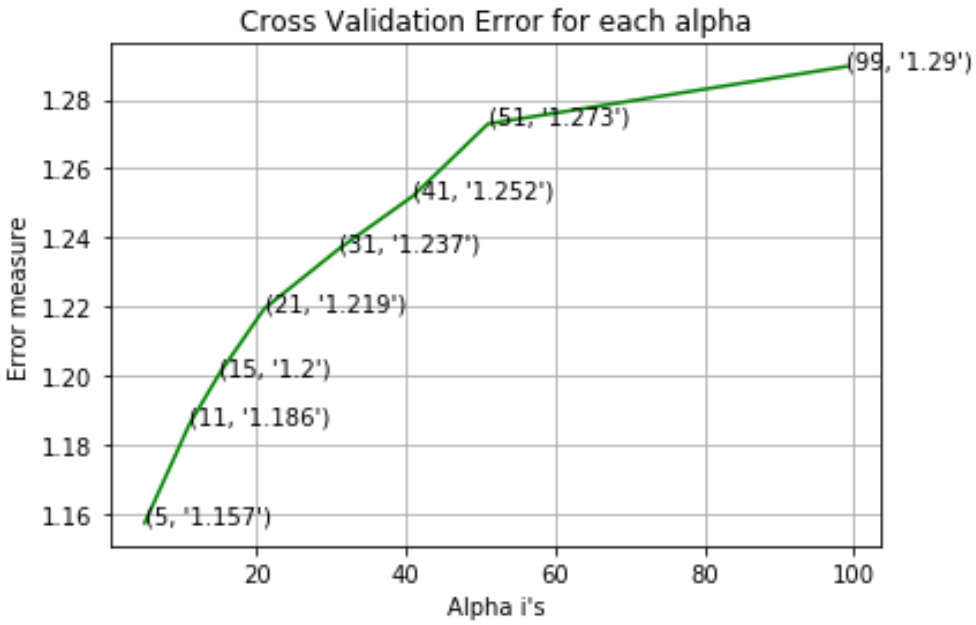
4.2.1. Hyper parameter tuning

In [64]:

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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------


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


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_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classe
    # to avoid rounding error while multiplying probabilites we use log-probs
    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_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 5
Log Loss : 1.157234749889869
for alpha = 11
Log Loss : 1.1858803560066438
for alpha = 15
Log Loss : 1.200491198115779
for alpha = 21
Log Loss : 1.2193174797033421
for alpha = 31
Log Loss : 1.2366645394301825
for alpha = 41
Log Loss : 1.2521257450628247
for alpha = 51
Log Loss : 1.2729512825567701
for alpha = 99
Log Loss : 1.2895798091918085
```




```
For values of best alpha =  5 The train log loss is: 0.8334705
```

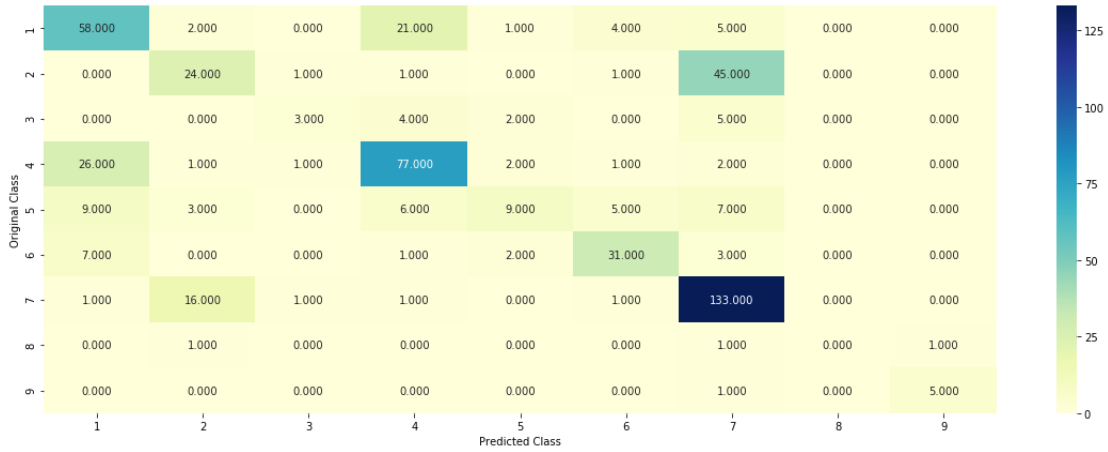
```
208598305
For values of best alpha =  5 The cross validation log loss i
s: 1.157234749889869
For values of best alpha =  5 The test log loss is: 1.11652801
67154183
```

4.2.2. Testing the model with best hyper paramters

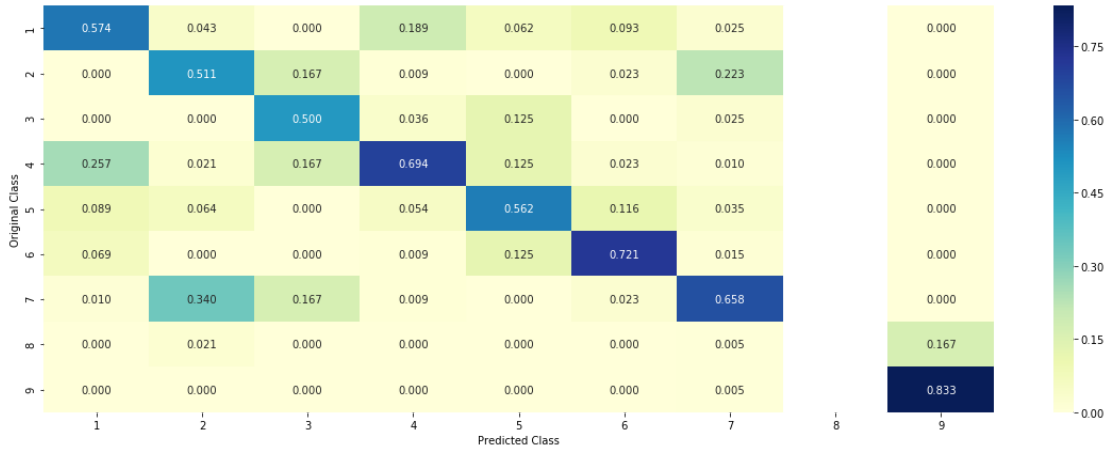
In [65]:

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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotC
```

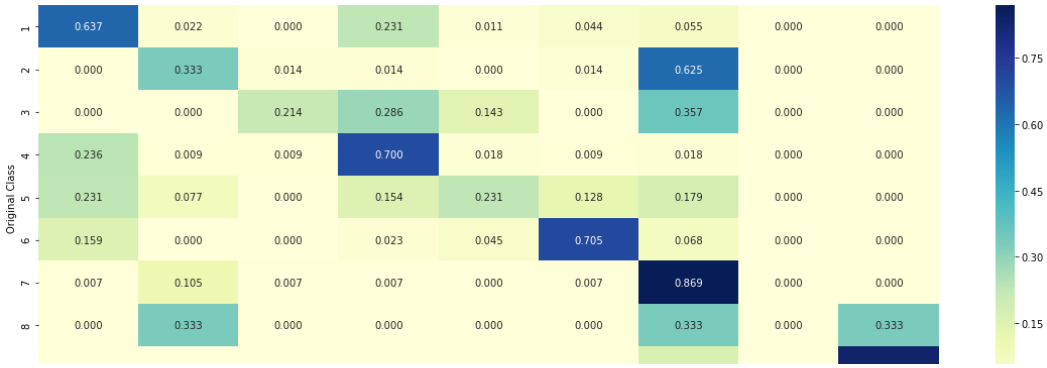
```
Log loss : 1.157234749889869
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [66]:

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

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

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

4.2.4. Sample Query Point-2

In [67]:

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

test_point_index = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_onehotCoding[test_point_index].reshape(1, 
print("The ",alpha[best_alpha]," nearest neighbours of the test points belon
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 4
The  5  nearest neighbours of the test points belongs to class
es [4 4 1 3 1]
Fequency of nearest points : Counter({4: 2, 1: 2, 3: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [68]:

```
print("One hot encoding features :")
print("train data with text one hot encoding = ", train_x_ohe.shape)
print("test data with text one hot encoding = ", test_x_ohe.shape)
print("cv data with text one hot encoding = ", cv_x_ohe.shape)
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/g
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------


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


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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.0597420328809801
for alpha = 1e-05
Log Loss : 1.0026374856620024
for alpha = 0.0001
Log Loss : 0.9721499915509625
for alpha = 0.001
Log Loss : 1.0246254781302195
for alpha = 0.01
Log Loss : 1.2386585786283877
for alpha = 0.1
Log Loss : 1.6826437298736456
for alpha = 1
Log Loss : 1.7984314951338345
for alpha = 10
Log Loss : 1.8116343893480438
for alpha = 100
Log Loss : 1.8130781248093957
```

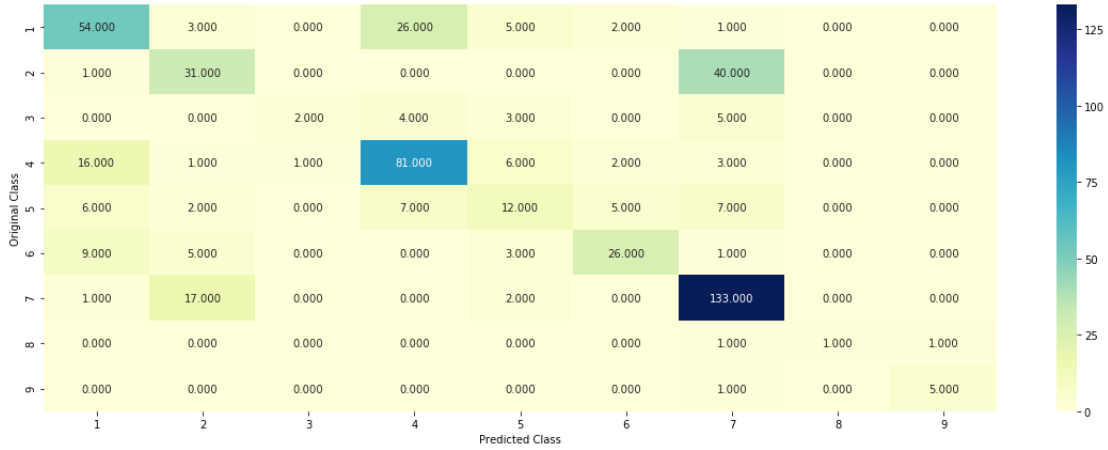

```
For values of best alpha =  0.0001 The train log loss is: 0.38
789674962077864
For values of best alpha =  0.0001 The cross validation log lo
ss is: 0.9721499915509625
For values of best alpha =  0.0001 The test log loss is: 0.978
935695570413
```

4.3.1.2. Testing the model with best hyper paramters

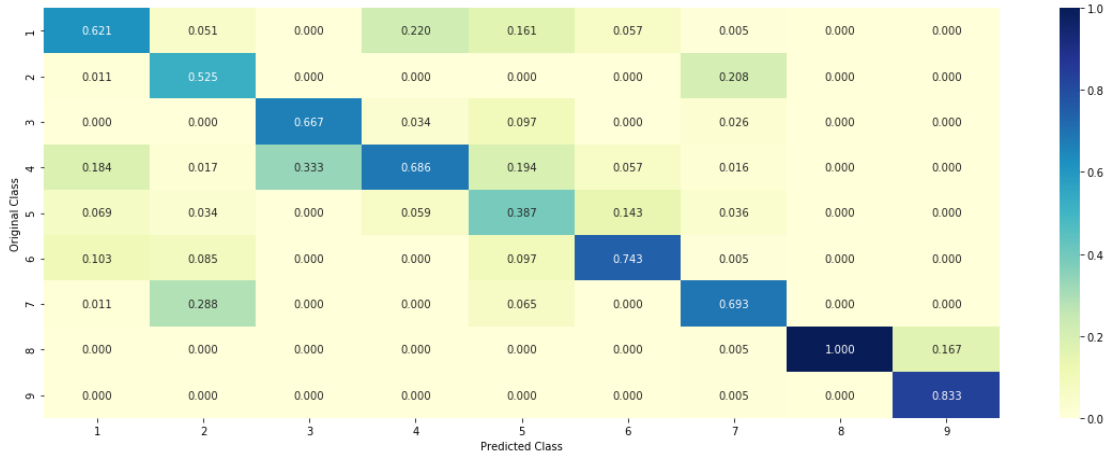
In [69]:

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

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

#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotC
```

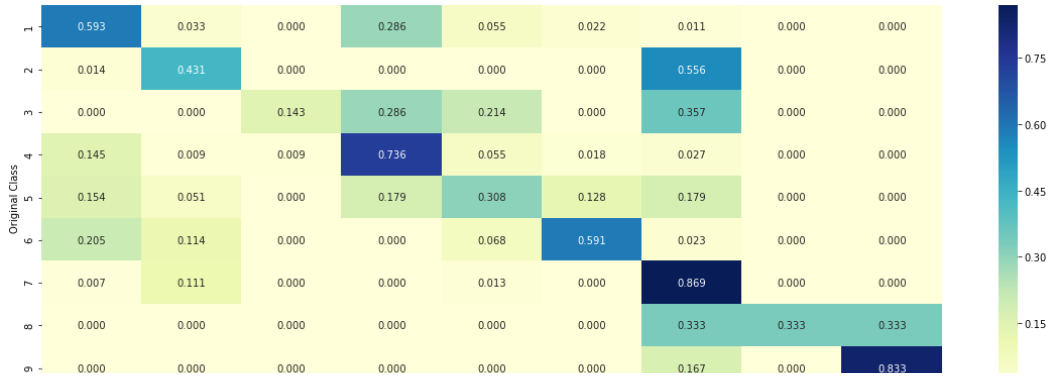
```
Log loss : 0.9721499915509625
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Feature Importance

In [70]:

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

4.3.1.3.1. Correctly Classified point

In [71]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 2
Predicted Class Probabilities: [[1.450e-02 8.877e-01 2.700e-03
2.350e-02 1.620e-02 3.800e-03 4.970e-02
  1.500e-03 4.000e-04]]
Actual Class : 5
--------------------------------------------------
139 Text feature [1a] present in test data point [True]
299 Text feature [100] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [72]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.127  0.0149 0.0154 0.7499
0.033  0.0211 0.0322 0.0043 0.0022]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2. Without Class balancing

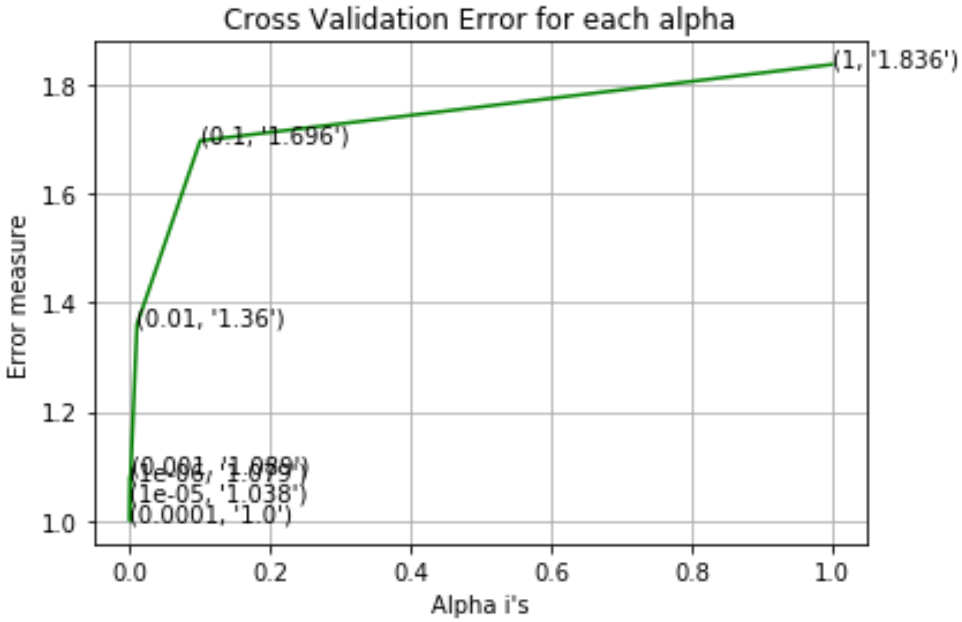
4.3.2.1. Hyper paramter tuning

In [73]:

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

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

#-------------------------------

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stab
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sig
#
# 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
#-------------------------------------
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classe
    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', randor
clf.fit(train_x_onehotCoding, train_y)
```

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

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

```
for alpha = 1e-06
Log Loss : 1.078902773367035
for alpha = 1e-05
Log Loss : 1.0380307269427715
for alpha = 0.0001
Log Loss : 0.9998360476218316
for alpha = 0.001
Log Loss : 1.0886564048866212
for alpha = 0.01
Log Loss : 1.3597441979117773
for alpha = 0.1
Log Loss : 1.696032371152017
for alpha = 1
Log Loss : 1.8357336116632716
```



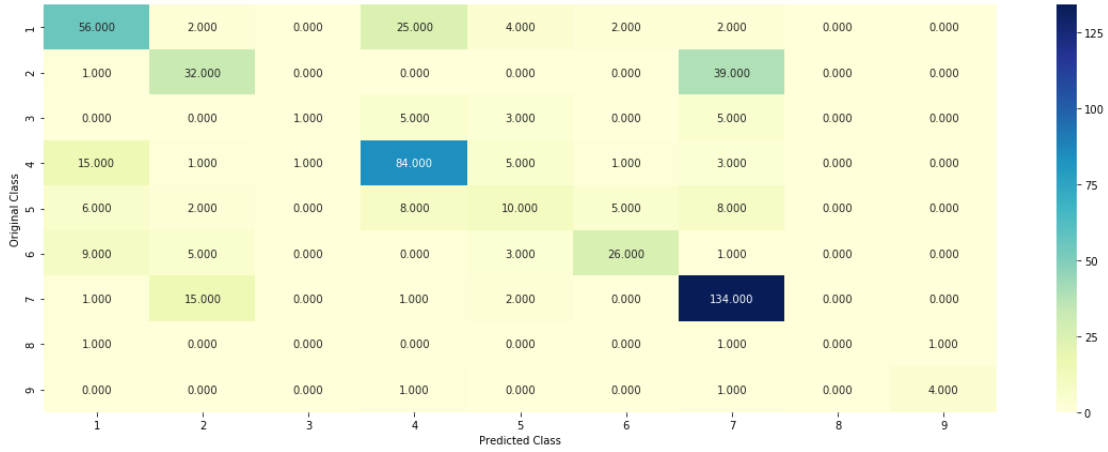

```
For values of best alpha =  0.0001 The train log loss is: 0.38
071161241021784
For values of best alpha =  0.0001 The cross validation log lo
ss is: 0.9998360476218316
For values of best alpha =  0.0001 The test log loss is: 1.004
4679505969074
```

4.3.2.2. Testing model with best hyper parameters

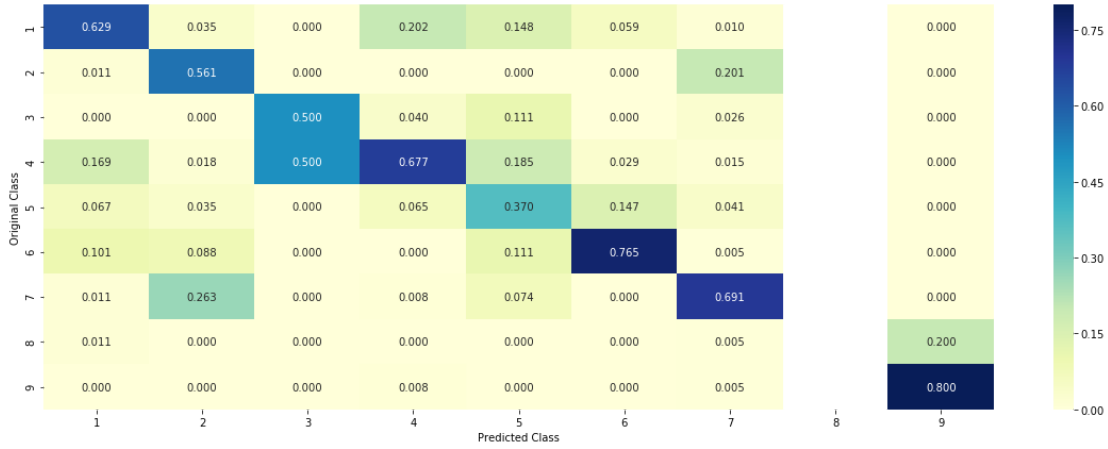
In [74]:

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

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

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

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

```
Log loss : 0.9998360476218316
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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



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

4.3.2.3. Feature Importance, Correctly Classified point

In [75]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', randor
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 2
Predicted Class Probabilities: [[1.550e-02 8.809e-01 1.600e-03
2.920e-02 1.460e-02 3.700e-03 4.780e-02
  6.600e-03 1.000e-04]]
Actual Class : 5
--------------------------------------------------
143 Text feature [1a] present in test data point [True]
314 Text feature [100] present in test data point [True]
487 Text feature [105] 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 [76]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 4
Predicted Class Probabilities: [[1.308e-01 1.580e-02 8.300e-03
7.368e-01 2.960e-02 1.930e-02 4.530e-02
  1.360e-02 4.000e-04]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.4. Linear Support Vector Machines

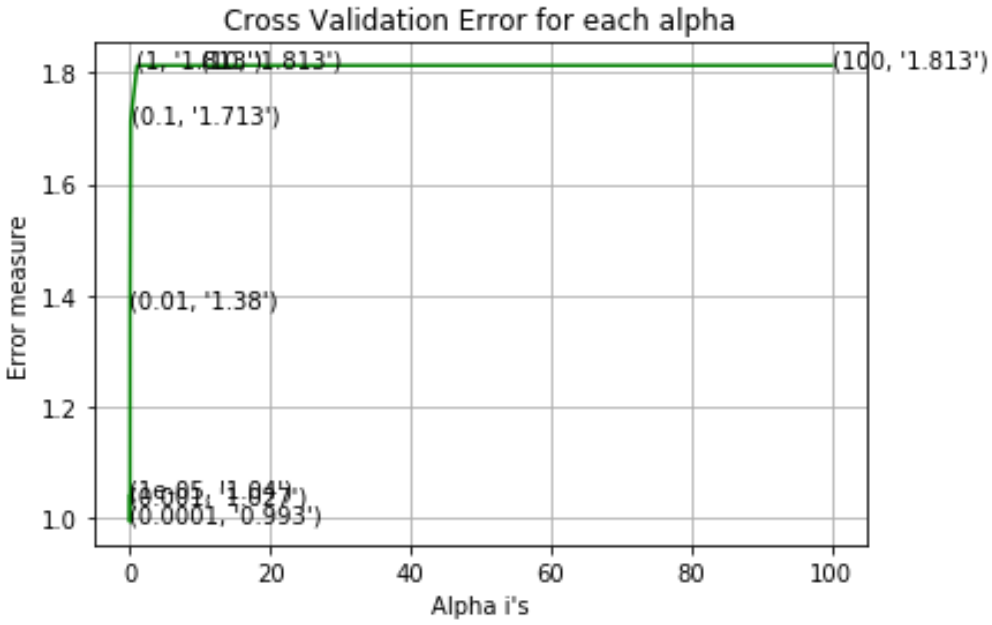
4.4.1. Hyper paramter tuning

In [77]:

```
# read more about support vector machines with linear kernals here http://sci

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given trai
# predict(X)    Perform classification on samples in X.
# --------------------------------



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

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

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

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

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

```
for C = 1e-05
Log Loss : 1.039947495522423
for C = 0.0001
Log Loss : 0.9930573316600035
for C = 0.001
Log Loss : 1.027201813492623
for C = 0.01
Log Loss : 1.3797085034853125
for C = 0.1
Log Loss : 1.713421943812689
for C = 1
Log Loss : 1.8133478224079318
for C = 10
Log Loss : 1.8133484225593681
for C = 100
Log Loss : 1.8133484767361585
```




```
For values of best alpha =  0.0001 The train log loss is:
0.3266280043761162
For values of best alpha =  0.0001 The cross validation log
loss is: 0.9930573316600035
```

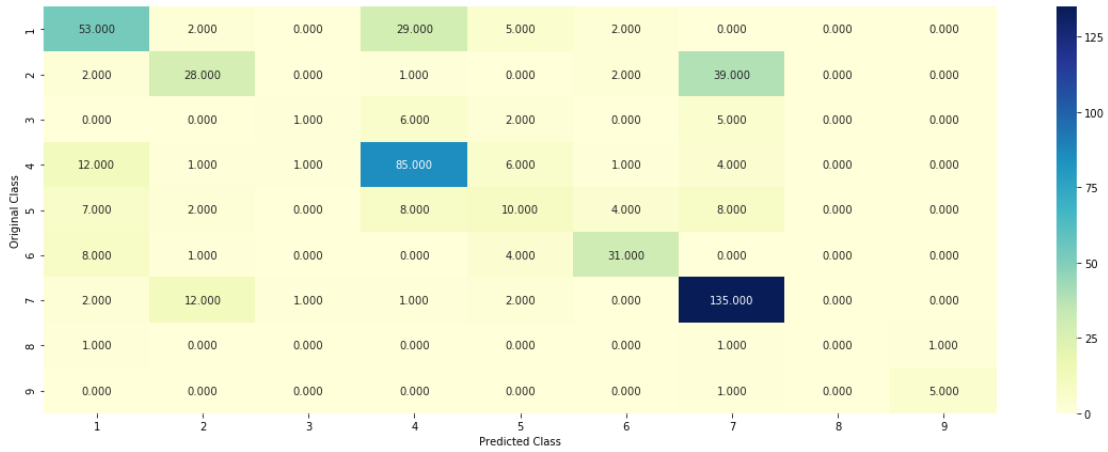
```
For values of best alpha =  0.0001 The test log loss is: 1.
0145405289168337
```

4.4.2. Testing model with best hyper parameters

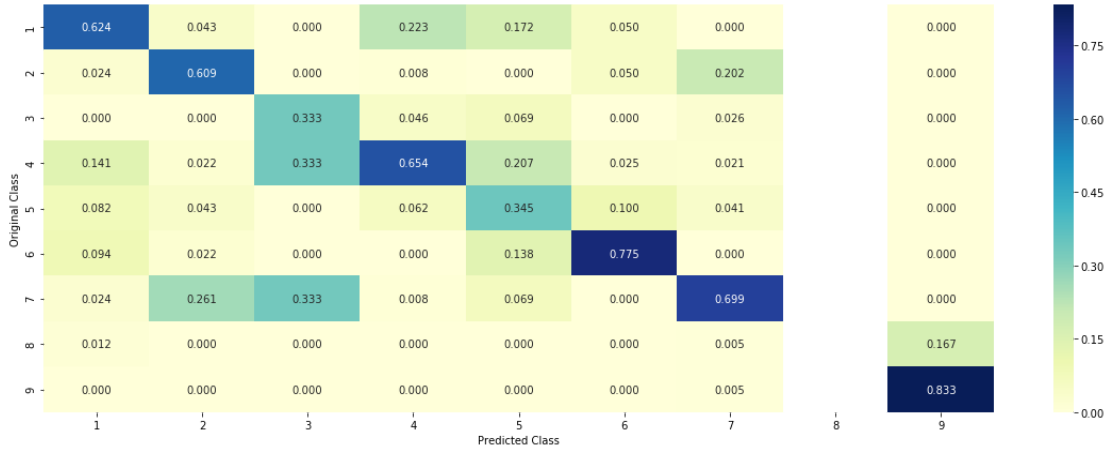
In [78]:

```
# read more about support vector machines with linear kernals here http://sci

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given trai
# predict(X)    Perform classification on samples in X.
# ------------------------------

# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weigh
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rand
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCo
```

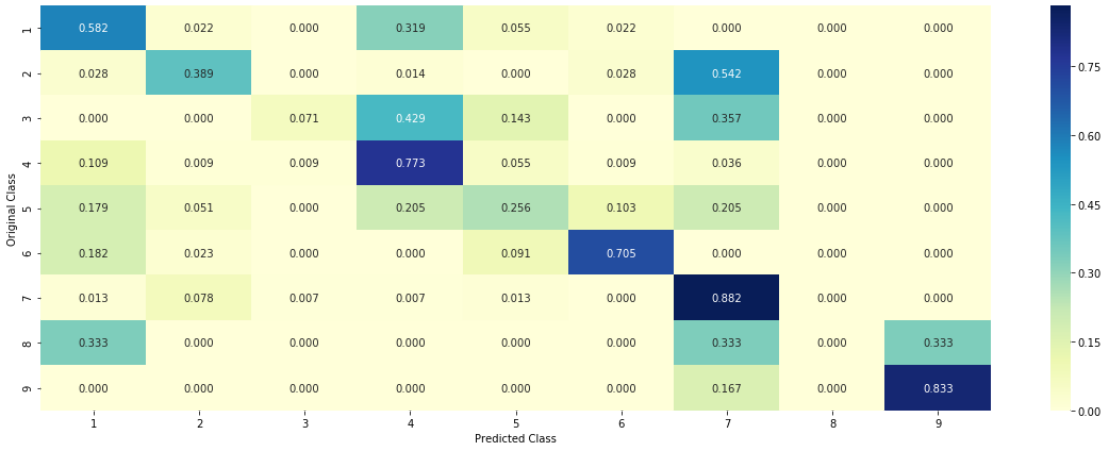
```
Log loss : 0.9930573316600035
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [79]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rand
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0597 0.7667 0.0204 0.0549
0.0189 0.0051 0.0723 0.0011 0.0009]]
Actual Class : 5
--------------------------------------------------
180 Text feature [1a] present in test data point [True]
378 Text feature [100] present in test data point [True]
398 Text feature [150] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [80]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1026 0.0366 0.094  0.5776
0.0512 0.0371 0.0947 0.0026 0.0037]]
Actual Class : 4
--------------------------------------------------
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 [82]:

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

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

# --------------------------------

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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_de
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cl
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],
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)], criterior
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The trair
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test
```

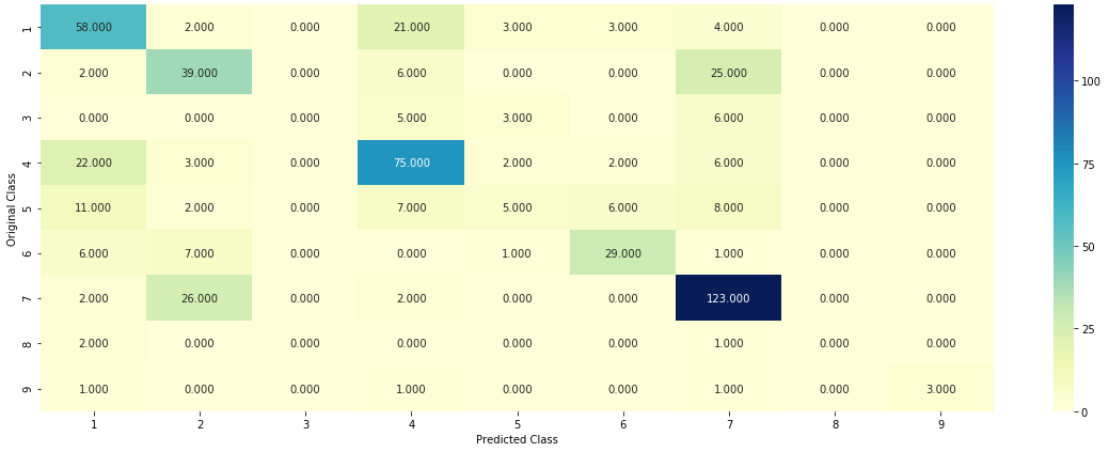
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1953388409557375
for n_estimators = 100 and max depth =  10
Log Loss : 1.1970762764301726
for n_estimators = 200 and max depth =  5
Log Loss : 1.1863370702566784
for n_estimators = 200 and max depth =  10
Log Loss : 1.1836479119698535
for n_estimators = 500 and max depth =  5
Log Loss : 1.1774324933561695
for n_estimators = 500 and max depth =  10
Log Loss : 1.1772659455213121
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1729324689807843
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1718712136996332
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1721815863878804
for n_estimators = 2000 and max depth =  10
```

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

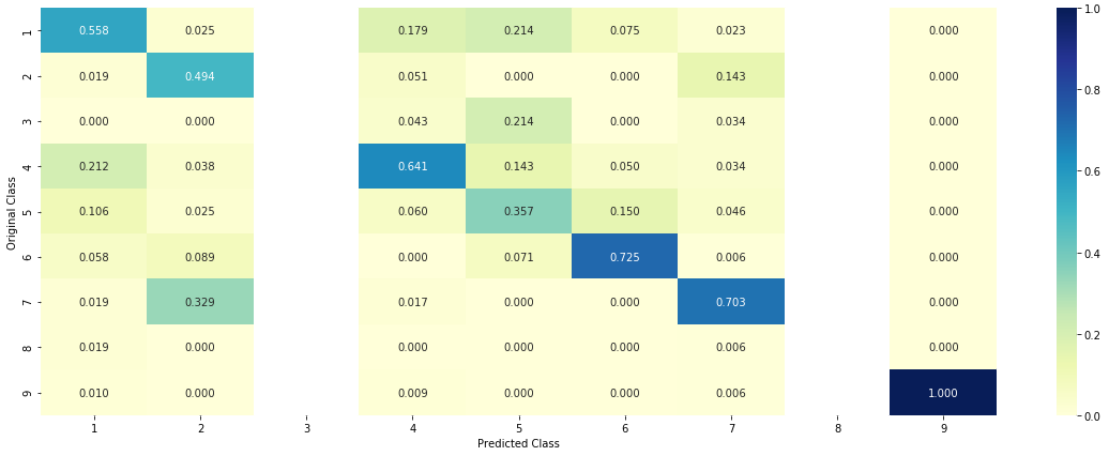
In [83]:

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

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

# ------------------------------

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

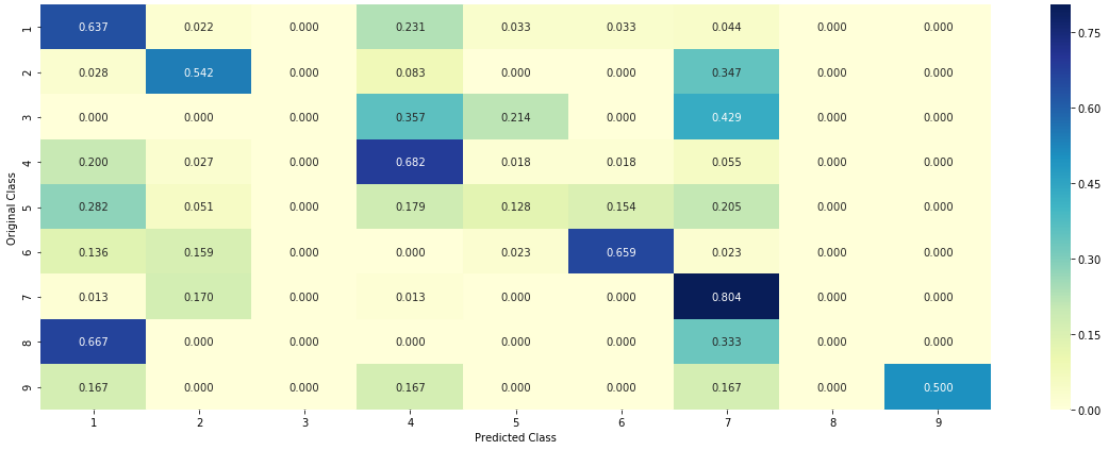
```
Log loss : 1.1708409851125832
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [84]:

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

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0738 0.654  0.0135 0.0707
0.042  0.0391 0.0956 0.0054 0.006 ]]
Actual Class : 5
--------------------------------------------------
20 Text feature [1997] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [85]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2207 0.0587 0.0225 0.4603
0.0806 0.0814 0.0582 0.0077 0.0098]]
Actuall Class : 4
--------------------------------------------------
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 [86]:

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

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

#------------------------------

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given trai
# predict(X)    Perform classification on samples in X.
# --------------------------------

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

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

# --------------------------------


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balan
clf2.fit(train_x_onehotCoding, train_y)
```

```
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.pre
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_prob
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], met
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" %
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

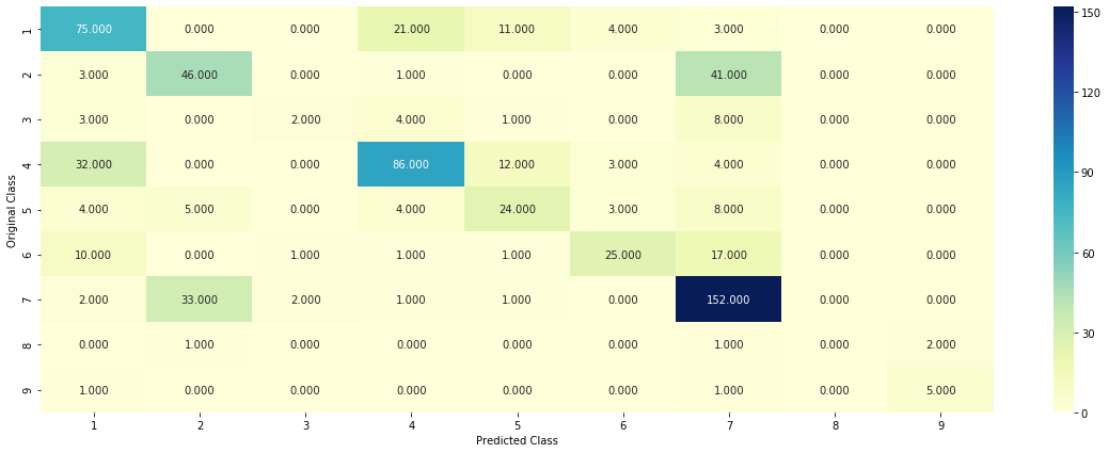
```
Logistic Regression :  Log Loss: 1.03
Support vector machines : Log Loss: 1.81
Naive Bayes : Log Loss: 1.27
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Los
s: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Los
s: 1.714
Stacking Classifer : for the value of alpha: 0.010000 Log Los
s: 1.321
Stacking Classifer : for the value of alpha: 0.100000 Log Los
s: 1.249
Stacking Classifer : for the value of alpha: 1.000000 Log Los
s: 1.607
Stacking Classifer : for the value of alpha: 10.000000 Log Los
s: 1.915
```

4.7.2 testing the model with the best hyper parameters

In [87]:

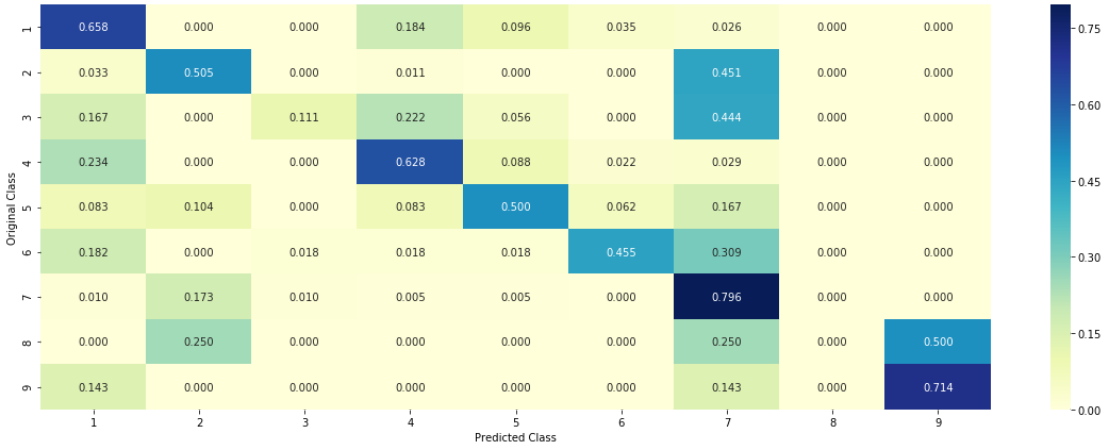
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cl
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.39754588209520
42
Log loss (CV) on the stacking classifier : 1.2489432335258686
Log loss (test) on the stacking classifier : 1.210045941489671
2
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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

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

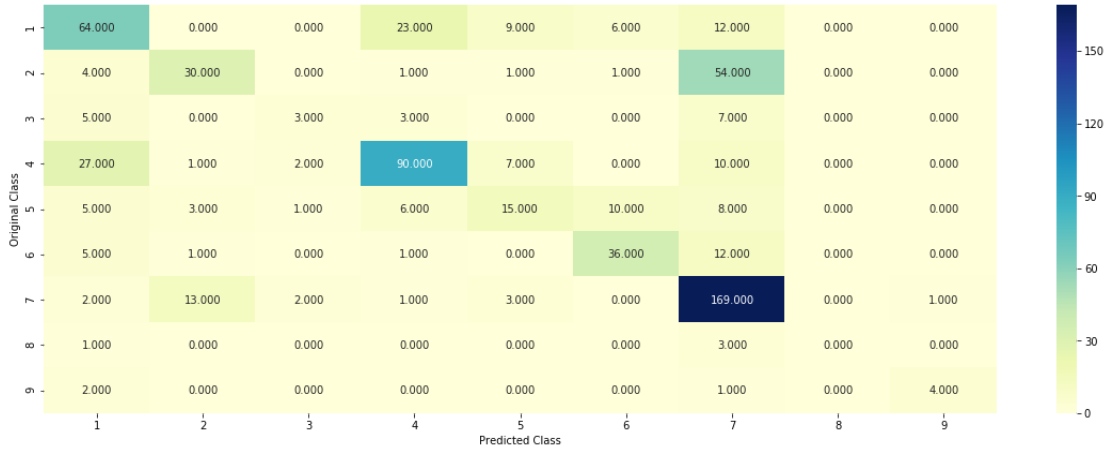


4.7.3 Maximum Voting classifier

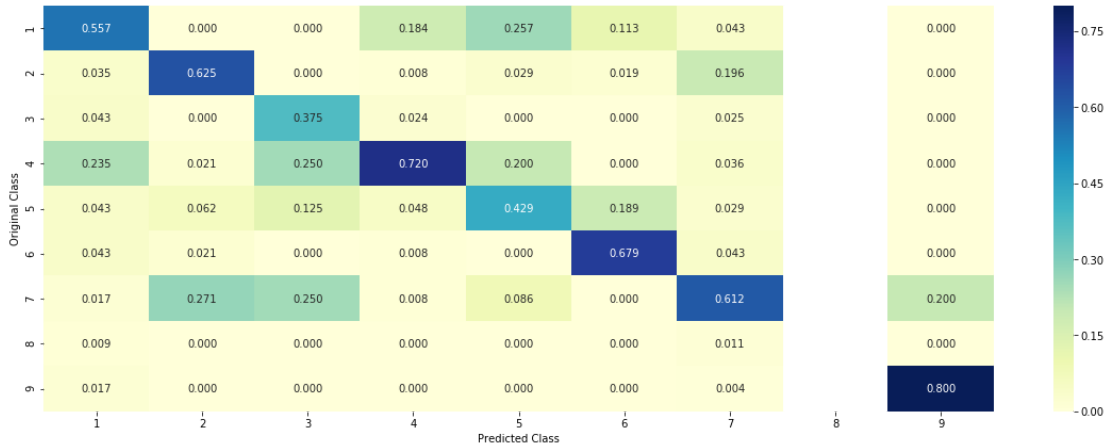
In [0]:

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

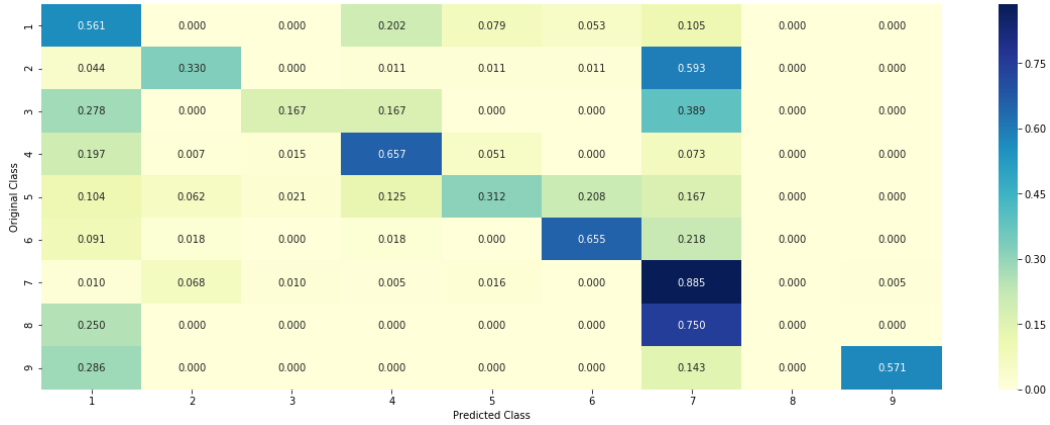
```
Log loss (train) on the VotingClassifier : 0.9407598679043604
Log loss (CV) on the VotingClassifier : 1.2835402100341697
Log loss (test) on the VotingClassifier : 1.223278167176945
Number of missclassified point : 0.3819548872180451
-------------------- Confusion matrix --------------------
```




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




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



5. Assignments

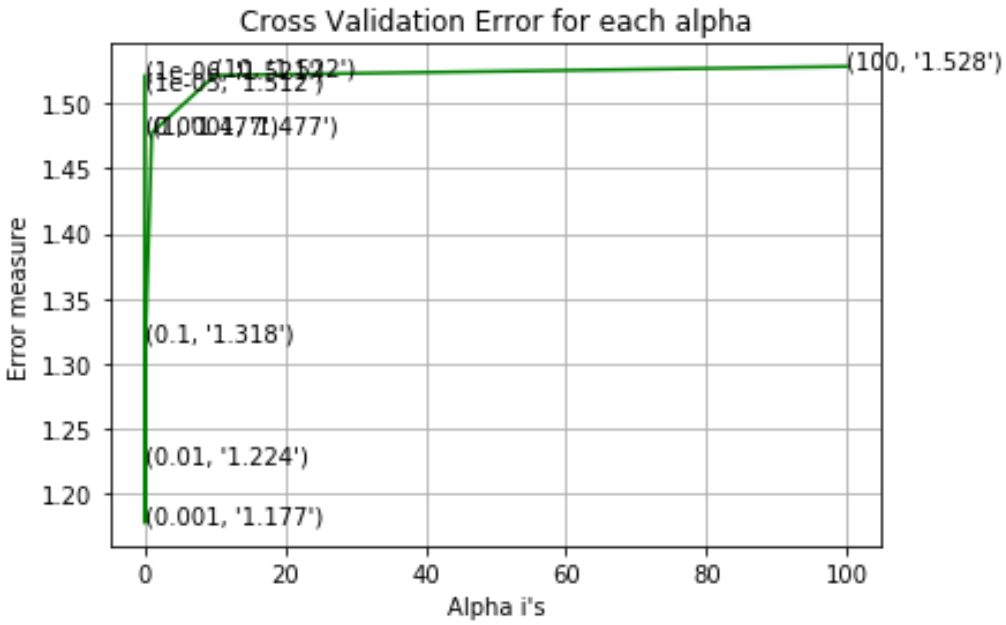
1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and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

In [99]:

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

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

#-------------------------------


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


alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss=
    clf.fit(train_x_ohe, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ohe, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ohe)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classe
    # to avoid rounding error while multiplying probabilites we use log-probo
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

predict_y = sig_clf.predict_proba(train_x_ohe)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
predict_y = sig_clf.predict_proba(cv_x_ohe)
print('For values of best alpha = ', alpha[best_alpha], "The cross validatior
predict_y = sig_clf.predict_proba(test_x_ohe)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
```

```
for alpha = 1e-06
Log Loss : 1.5211388664225949
for alpha = 1e-05
Log Loss : 1.5115601542691313
for alpha = 0.0001
Log Loss : 1.4774475926130166
for alpha = 0.001
Log Loss : 1.1772940521360813
for alpha = 0.01
Log Loss : 1.2237140957072665
for alpha = 0.1
Log Loss : 1.3181652399447041
for alpha = 1
Log Loss : 1.4773341384440646
for alpha = 10
Log Loss : 1.5216388500543534
for alpha = 100
Log Loss : 1.5284498063545666
```




```
For values of best alpha =  0.001 The train log loss is: 0.
7517381780775437
For values of best alpha =  0.001 The cross validation log
loss is: 1.1772940521360813
```

```
For values of best alpha =  0.001 The test log loss is: 1.1
228448397354287
```

Conclusion

In [101]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Vectorizer", "Model", "Hyper Parameters", "CV log loss'

table.add_row(['TFIDF - Gene', 'Logestic Regression', 'Alpha = 0.0001',  1.2
table.add_row(['TFIDF - Variation', 'Logestic Regression', 'Alpha = 0.0001',
table.add_row(['TFIDF - Text', 'Logestic Regression', 'Alpha = 0.0001',  1.0
table.add_row(['TFIDF', 'Naive Bayes', 'Alpha = 0.1',  1.26, 1.21])
table.add_row(['TFIDF', 'K Nearest Neighbour', 'K = 5',  1.15, 1.11])
table.add_row(['TFIDF', 'Logestic Regression Class Balanced', 'Alpha = 0.000
table.add_row(['TFIDF', 'Logestic Regression', 'Alpha = 0.0001',  0.99, 1.00
table.add_row(['TFIDF', 'SVM Class Balanced', 'Alpha = 0.0001',  0.99, 1.01])
table.add_row(['TFIDF', 'Random Forest', 'n_estimators = 2000, max depth =  1
table.add_row(['TFIDF', 'Stacking', 'Alpha = 0.1',  1.24, 1.21])
table.add_row(['One hot encoding', 'Logestic Regression Class Balanced', 'Alp
print(table)
```

```
+------------------+-----------------------------------+-
-------------------------------------+-------------+-------
--------+
|     Vectorizer    |               Model                |
Hyper Parameters           | CV log loss | Test log loss |
+------------------+-----------------------------------+-
-------------------------------------+-------------+-------
--------+
|    TFIDF - Gene   |        Logestic Regression         |
Alpha = 0.0001             |     1.21    |      1.17     |
| TFIDF - Variation |        Logestic Regression         |
Alpha = 0.0001             |     1.71    |      1.69     |
|    TFIDF - Text   |        Logestic Regression         |
Alpha = 0.0001             |     1.07    |      1.11     |
|       TFIDF       |            Naive Bayes             |
Alpha = 0.1              |     1.26    |      1.21     |
|       TFIDF       |        K Nearest Neighbour         |
K = 5                  |     1.15    |      1.11     |
|       TFIDF       | Logestic Regression Class Balanced |
Alpha = 0.0001             |     0.97    |      0.97     |
|       TFIDF       |        Logestic Regression         |
Alpha = 0.0001             |     0.99    |      1.0      |
|       TFIDF       |         SVM Class Balanced         |
Alpha = 0.0001             |     0.99    |      1.01     |
|       TFIDF       |           Random Forest            |
n_estimators = 2000, max depth =  10 |     1.17    |
1.18     |
|       TFIDF       |              Stacking              |
Alpha = 0.1              |     1.24    |      1.21     |
|  One hot encoding | Logestic Regression Class Balanced |
```

```
Alpha = 0.001             |     1.17    |      1.12     |
+-------------------+-----------------------------------+-
-----------------------------------+-------------+-------
--------+
```

Summary

- Trained all the models with the TFIDF vectorizer.
- Performed hyperparameter tuning to find the best parameters.
- Trained Logestic Regression with one hot encoding of text data upto 4 grams.
- The best CV and Test log loss obtained is 0.97.
- Logestic Regression with class balance is the best suited model for this dataset
- Successful in reducing the Test and Cross Validation log loss to less than 1.