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
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. 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
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

ID,Gene,Variation,Class
0,FAM58A,Truncating Mutations,1
1,CBL,W802*,2
2,CBL,Q249E,2
...

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

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

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 [0]:

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

from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

In [2]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
```

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [3]:

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

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

Out[3]:

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

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

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

3.1.2. Reading Text Data

In [0]:

```
result.to_pickle('gdrive/My Drive/result_quora.pkl')
```

In [4]:

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

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

Out[4]:

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

3.1.3. Preprocessing of text

In [16]:

```
import nltk

from nltk.corpus import stopwords
nltk.download('stopwords')


# 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.


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

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [17]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 420.71387400000003 seconds
```

In [18]:

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

Out[18]:

	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 [19]:

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

Out[19]:

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

In [0]:

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

In [21]:

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

Out[21]:

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

In [0]:

In [0]:

```
result['TEXT'] =  result['Gene'].map(str) + ' '+ result['Variation'].map(str) + ' '+result['TEXT'].m
ap(str)
```

Word_Count of Text as Feature

In [0]:

```
word_count = result['TEXT'].str.split().apply(len).values
```

In [0]:

```
result['word_count'] = word_count
```

char_count of text as feature

In [0]:

```
result['char_count'] = result['TEXT'].apply(len)
```

In [0]:

```
result['word_density'] = result['char_count'] / (result['word_count']+1)
```

In [27]:

```
result.head()
```

Out[27]:

	ID	Gene	Variation	Class	TEXT	word_count	char_count	word_density
0	0	FAM58A	Truncating Mutations	1	FAM58A Truncating Mutations Cyclin dependent k...	4665	31703	6.794471
1	1	CBL	W802*	2	CBL W802* Abstract Background Non small cell l...	4278	28334	6.621641
2	2	CBL	Q249E	2	CBL Q249E Abstract Background Non small cell l...	4278	28334	6.621641
3	3	CBL	N454D	3	CBL N454D Recent evidence demonstrated acquire...	3963	28545	7.201060
4	4	CBL	L399V	4	CBL L399V Oncogenic mutations monomeric Casita...	4422	32230	7.286909

Total No.of.Words with Capital Letters

In [33]:

```
from string import ascii_uppercase
count_capitals = []
for i in tqdm(range(3321)):

  count_capitals.append(len(re.findall(r'[A-Z]',result['TEXT'][i])))
```

```
100%|██████████| 3321/3321 [00:02<00:00, 1111.93it/s]
```

Because words with Capital letters maybe important in medical Literature.So use them

In [ ]:

```
result['capital_count'] = count_capitals
```

In [0]:

```
result['No.of.digits'] = 0
for j in tqdm(range(3321)):
    sample = result['TEXT'][j] #sample string
    letters = 0
    numeric = 0

    for i in sample:
            if i.isdigit():
                numeric +=1
            elif i.isalpha():
                letters +=1
            else:
                pass
    result['No.of.digits'][j] = numeric
```

```
100%|██████████| 3321/3321 [08:12<00:00,  6.85it/s]
```

Also the no.of.digits inside each Text cell

In [0]:

```
gene_text_count = []
variation_text_count = []

for i in tqdm(range(3321)):


    my_words = []

    my_words = result['TEXT'][i].split()
    my_words = list(my_words)
    count_gene = 0
    count_variation = 0


    for j in range(len(my_words)):

        variation_lower = (result['Variation'][i]).lower()
        if(my_words[j] == variation_lower):

            count_variation += 1
        else:
            pass

        gene_lower = (result['Gene'][i]).lower()
        if(my_words[j] == gene_lower):

            count_gene += 1
        else:
            pass
    gene_text_count.append(count_gene)
    variation_text_count.append(count_variation)
```

```
100%|██████████| 3321/3321 [09:50<00:00,  6.47it/s]
```

Check if the Gene and Variable are present inside the Text and How many No.of.Times

In [0]:

```
result['gene_text'] = gene_text_count
result['variation_text'] = variation_text_count
```

In [0]:

```
result = pd.read_pickle('gdrive/My Drive/result_quora.pkl')
```

In [43]:

```
result.columns
```

Out[43]:

```
Index(['ID', 'Gene', 'Variation', 'Class', 'TEXT', 'word_count', 'char_count',
       'word_density', 'No.of.digits', 'gene_text', 'variation_text',
       'capital_count'],
      dtype='object')
```

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 [0]:

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

In [0]:

```
# split the data into test and train by maintaining same distribution of output varaible 'y_true'
[stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2
)
# split the train data into train and cross validation by maintaining same distribution of output
varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
)
```

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

In [46]:

```
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

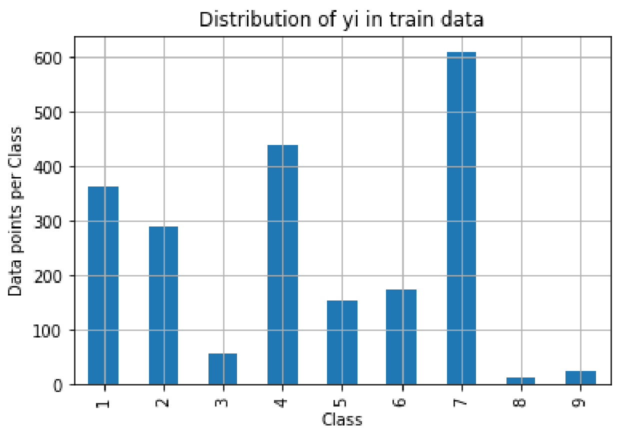
In [0]:

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

In [48]:

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
```

```
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
```

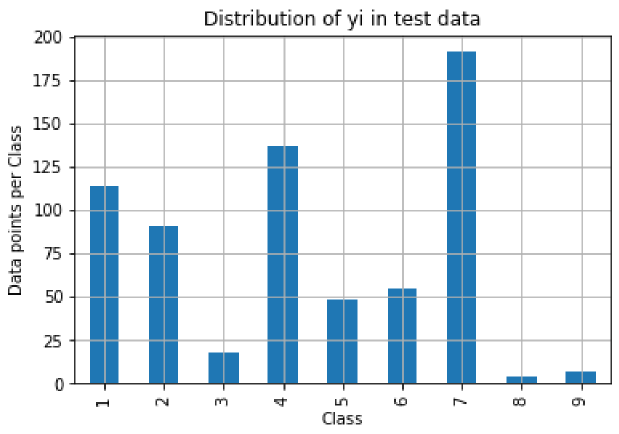



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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

In [49]:

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')
```

```
--------------------------------------------------------------------------------
```



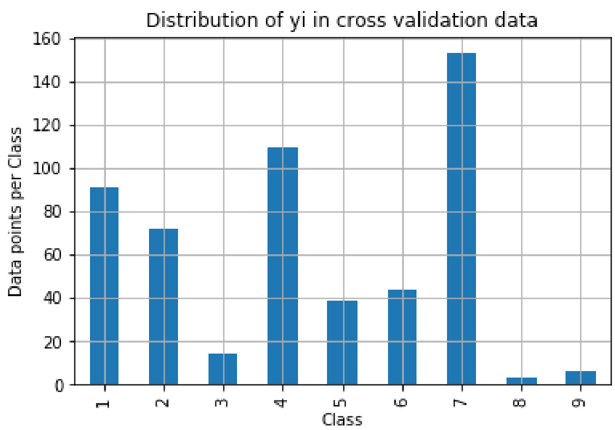

```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
```

```
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
```

In [50]:

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round
((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```

```
--------------------------------------------------------------------------------
```




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

3.2 Prediction using a 'Random' Model

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

In [0]:

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

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

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

```
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

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

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

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

In [52]:

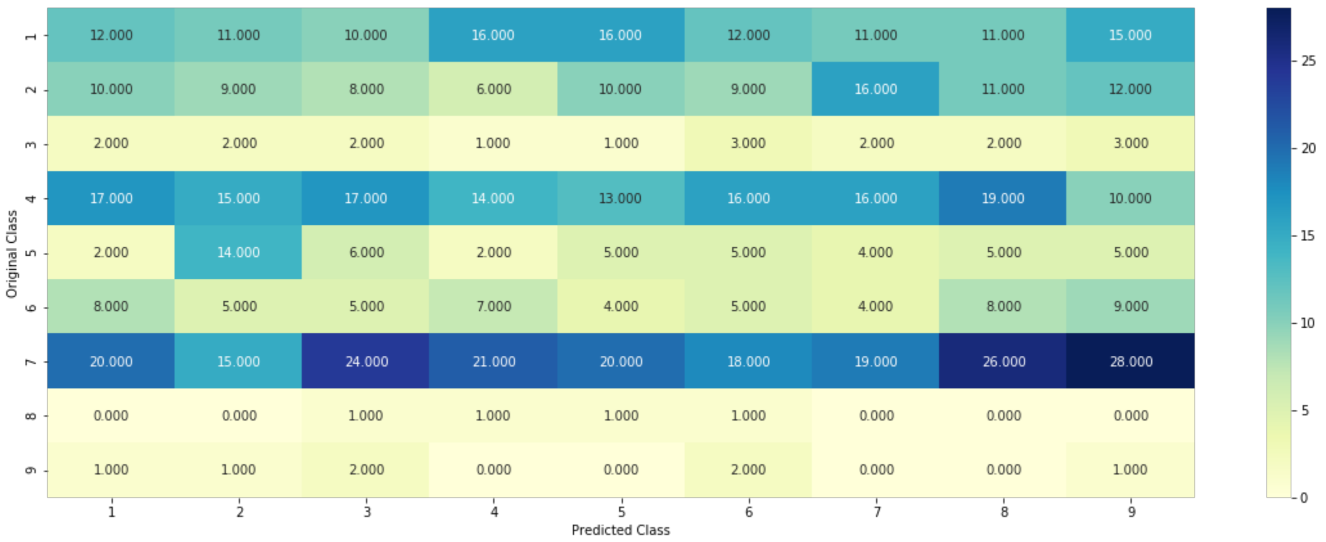
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-
15))
```

```
Log loss on Cross Validation Data using Random Model 2.4805688589412798
```

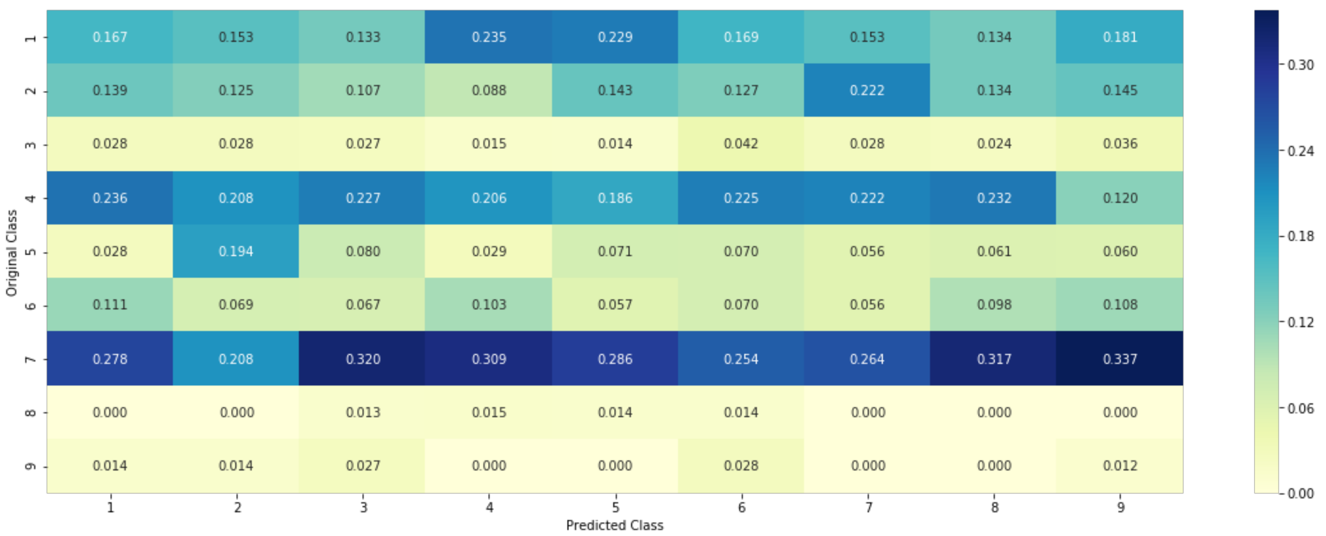
In [53]:

```
# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Test Data using Random Model 2.467797473914353
-------------------- Confusion matrix --------------------
```




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




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

3.3 Univariate Analysis

In [0]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / nu
mber of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variation
    gv_dict = dict()

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

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
```

```
        # we are adding the gene/variation to the dict as key and vec as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #       'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #       'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #       'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #       'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #       'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #       'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#           gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [55]:

```
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 : 237
BRCA1      190
TP53       108
EGFR        91
BRCA2       78
PTEN        77
BRAF        64
```

```
BRAF        64
KIT         60
ALK         41
PDGFRA      41
ERBB2       40
Name: Gene, dtype: int64
```

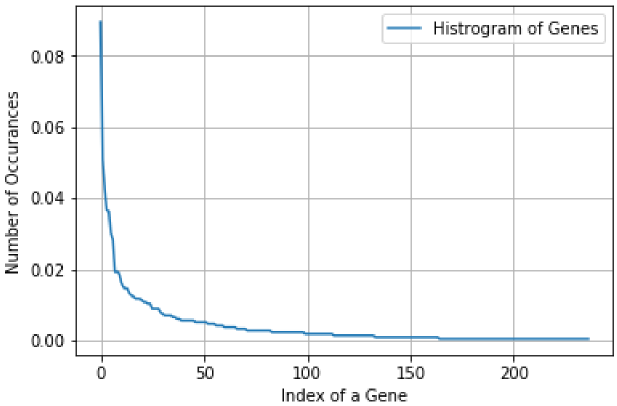
In [56]:

```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
d they are distibuted as follows",)
```

```
Ans: There are 237 different categories of genes in the train data, and they are distibuted as fol
lows
```

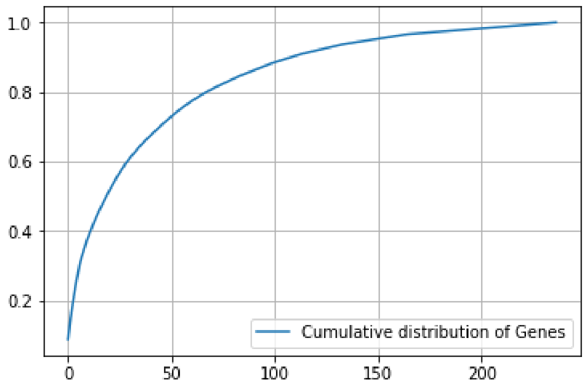
In [57]:

```
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 [58]:

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



Q3. How to featurize this Gene feature ?

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

features/

1. One hot Encoding
2. Response coding

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

In [0]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [60]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The sha
pe of gene feature:", train_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape of g
ene feature: (2124, 9)
```

Count_Vectorizer of Gene

In [0]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [62]:

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

Out[62]:

```
3047       KIT
3277       RET
1992    MAP2K1
628      FBXW7
59       PTPRT
Name: Gene, dtype: object
```

In [63]:

```
gene_vectorizer.get_feature_names()
```

Out[63]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl2',
```

```
    'atm',
    'atrx',
    'aurka',
    'aurkb',
    'axin1',
    'axl',
    'b2m',
    'bap1',
    'bard1',
    'bcl10',
    'bcl2',
    'bcor',
    'braf',
    'brca1',
    'brca2',
    'brd4',
    'brip1',
    'btk',
    'card11',
    'carm1',
    'casp8',
    'cbl',
    'ccnd1',
    'ccnd2',
    'ccnd3',
    'cdh1',
    'cdk12',
    'cdk4',
    'cdk6',
    'cdk8',
    'cdkn1a',
    'cdkn1b',
    'cdkn2a',
    'cdkn2b',
    'cdkn2c',
    'cebpa',
    'chek2',
    'cic',
    'crebbp',
    'ctcf',
    'ctla4',
    'ctnnb1',
    'ddr2',
    'dicer1',
    'dnmt3a',
    'dnmt3b',
    'dusp4',
    'egfr',
    'elf3',
    'ep300',
    'epas1',
    'erbb2',
    'erbb3',
    'erbb4',
    'ercc2',
    'ercc3',
    'ercc4',
    'erg',
    'errfi1',
    'esr1',
    'etv1',
    'etv6',
    'ewsr1',
    'ezh2',
    'fam58a',
    'fanca',
    'fancc',
    'fat1',
    'fbxw7',
    'fgf19',
    'fgf3',
    'fgf4',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'fgfr4',
    'flt1',
```

```
    'flt3',
    'foxa1',
    'foxl2',
    'foxo1',
    'foxp1',
    'gata3',
    'gli1',
    'gna11',
    'gnaq',
    'gnas',
    'h3f3a',
    'hla',
    'hnf1a',
    'hras',
    'idh1',
    'idh2',
    'igf1r',
    'ikbke',
    'ikzf1',
    'il7r',
    'inpp4b',
    'jak1',
    'jak2',
    'jun',
    'kdm5c',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'kmt2a',
    'kmt2b',
    'kmt2c',
    'kmt2d',
    'knstrn',
    'kras',
    'lats1',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mapk1',
    'mdm4',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mga',
    'mlh1',
    'mpl',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myod1',
    'nf1',
    'nf2',
    'nfe2l2',
    'nfkbia',
    'nkx2',
    'notch1',
    'notch2',
    'nras',
    'nsd1',
    'ntrk1',
    'ntrk2',
    'ntrk3',
    'nup93',
    'pax8',
    'pbrm1',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
```

```
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51c',
 'rad51d',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'rybp',
 'sdhc',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [64]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The sha
pe of gene feature:",train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 237)
```

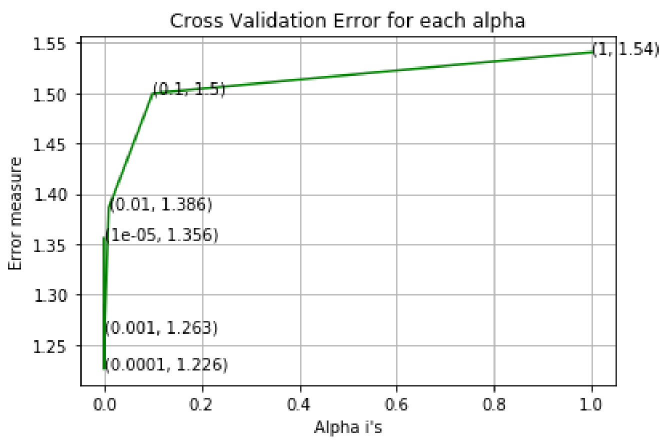
Q4. How good is this gene feature in predicting y_i?

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

In [65]:

```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link:
#------------------------------
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))

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()
```

```
For values of alpha =  1e-05 The log loss is: 1.3557995900133968
For values of alpha =  0.0001 The log loss is: 1.2260108848827873
For values of alpha =  0.001 The log loss is: 1.2627234475729803
For values of alpha =  0.01 The log loss is: 1.386264575990856
For values of alpha =  0.1 The log loss is: 1.4995946205704285
For values of alpha =  1 The log loss is: 1.540442517210596
```



In [66]:

```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of best alpha =  0.0001 The train log loss is: 1.0224667067049504
For values of best alpha =  0.0001 The cross validation log loss is: 1.2260108848827873
For values of best alpha =  0.0001 The test log loss is: 1.2327338087977802
```

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

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

In [67]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0
], " genes in train dataset?")

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  237  genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 521 out of  532 : 97.93233082706767
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [68]:

```
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 : 1945
Truncating_Mutations         56
Deletion                     45
Amplification                41
Fusions                      21
G12V                          4
Q61L                          3
E17K                          3
Promoter_Hypermethylation     2
R170W                         2
```

```
Q61H                              2
Name: Variation, dtype: int64
```

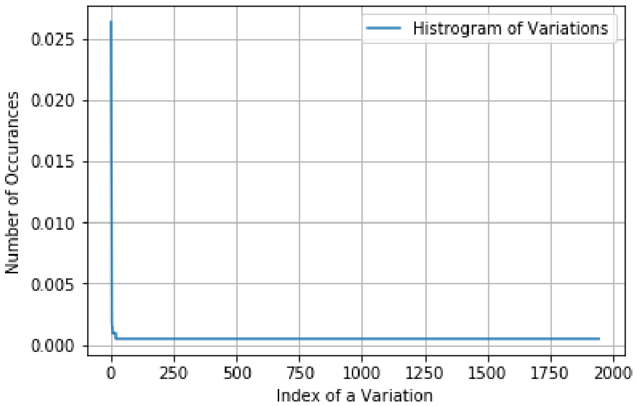
In [69]:

```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1945 different categories of variations in the train data, and they are distibuted
as follows
```

In [70]:

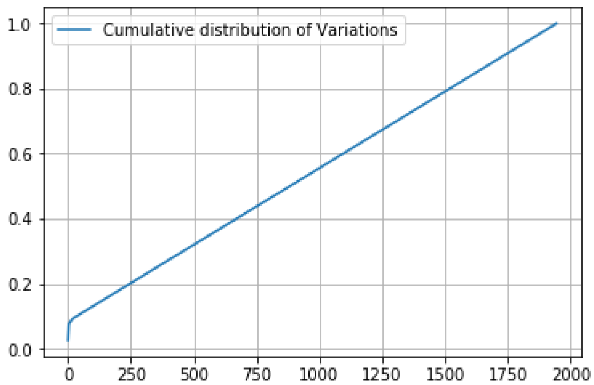
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [71]:

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

```
[0.02636535 0.04755179 0.06685499 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

features/

1. One hot Encoding
2. Response coding

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

In [0]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [73]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [0]:

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

In [75]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
od. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1972)
```

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

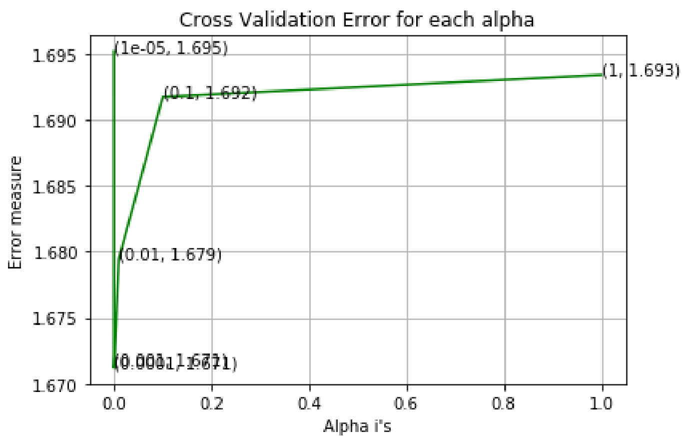
Let's build a model just like the earlier!

In [76]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
```

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

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

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.695225898568023
For values of alpha =  0.0001 The log loss is: 1.6711961754949494
For values of alpha =  0.001 The log loss is: 1.6714131993917003
For values of alpha =  0.01 The log loss is: 1.6794609358370465
For values of alpha =  0.1 The log loss is: 1.6917572897383029
For values of alpha =  1 The log loss is: 1.6934223934023502
```




```
For values of best alpha =  0.0001 The train log loss is: 0.758655380712521
For values of best alpha =  0.0001 The cross validation log loss is: 1.6711961754949494
For values of best alpha =  0.0001 The test log loss is: 1.6982610703608836
```

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

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

In [77]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
```

```
st and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
Q12. How many data points are covered by total  1945  genes in test and cross validation data
sets?
Ans
1. In test data 74 out of 665 : 11.12781954887218
2. In cross validation data 66 out of  532 : 12.406015037593985
```

3.2.3 Univariate Analysis on Text Feature

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

In [0]:

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

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

In [0]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

TfidfVectorizer on Text

In [80]:

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


# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
```

```
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

In [0]:

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

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


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

In [0]:

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

In [0]:

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

In [0]:

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

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

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

In [0]:

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

In [86]:

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

```
Counter({251.6426465593823: 1, 180.61555489453545: 1, 149.44668320842842: 1, 134.86209345199944: 1
, 128.24920418194222: 1, 121.30656782888587: 1, 120.87366515152571: 1, 115.14039425225866: 1,
112.99311435981605: 1, 111.12066064976118: 1, 107.66668856107863: 1, 90.86737452247827: 1,
90.0330286227174: 1, 87.28095880828711: 1, 82.35832704188978: 1, 81.34485083150496: 1,
79.83145530474172: 1, 79.62445804948915: 1, 78.39362768372656: 1, 78.01661528164242: 1,
77.69965970377025: 1, 77.0993752242243: 1, 71.31865610278997: 1, 70.38215058078576: 1,
69.75552780007668: 1, 69.25163263959635: 1, 68.83934959718158: 1, 67.69132129628915: 1,
66.31890168289742: 1, 65.13045158088825: 1, 65.00414372402217: 1, 64.8032146509567: 1,
64.4141231020804: 1, 62.249947214471966: 1, 60.04548946421773: 1, 56.8446032693534: 1,
56.801505319838654: 1, 56.04059987751133: 1, 54.80279307180526: 1, 52.820405613172994: 1,
52.222944649420235: 1, 52.017100672565135: 1, 49.92492103816089: 1, 49.38500457084548: 1,
49.33239872620896: 1, 47.00380090905089: 1, 46.97026986942904: 1, 46.688529128274524: 1,
46.188441224946814: 1, 44.83246488927279: 1, 44.80656327245555: 1, 44.71319069471232: 1, 44.4872605
03251946: 1, 44.40465452218241: 1, 43.54192249746686: 1, 43.375137524412665: 1, 43.36475878654876:
1, 42.808778346454: 1, 42.731495322700106: 1, 42.709413979028355: 1, 42.578429471027526: 1,
42.52996242739941: 1, 42.410894533017135: 1, 41.79651220395824: 1, 41.720796914925955: 1,
40.42682250708336: 1, 40.181408981316416: 1, 40.09978088721332: 1, 39.558919965858465: 1,
38.996393998077195: 1, 38.808159879052525: 1, 38.67742156498923: 1, 38.485880092786346: 1,
38.41429013116791: 1, 37.649100982492584: 1, 37.60539281848493: 1, 37.524313340657116: 1,
37.48726358809477: 1, 37.22494946871869: 1, 36.97254381621758: 1, 36.9549676158172: 1,
36.901466343719285: 1, 36.60427391430408: 1, 36.14545221010027: 1, 35.56019317788561: 1, 34.4771913
3229695: 1, 34.185735671374424: 1, 34.18533490041193: 1, 33.99223595943901: 1, 33.94374417298443:
1, 33.70357688535874: 1, 33.35729042204409: 1, 32.780791507320146: 1, 32.763442117321105: 1,
32.68496297610336: 1, 32.49007626102596: 1, 32.30992796046409: 1, 32.304556192717975: 1,
32.258514748178854: 1, 32.238739762439394: 1, 32.137158659873656: 1, 32.12402746367597: 1,
31.83957908167809: 1, 31.79705674563573: 1, 31.672852122382572: 1, 31.515601161353143: 1,
31.37726311212245: 1, 31.285352103026096: 1, 31.26131063624259: 1, 31.18557788665237: 1,
31.169081623882548: 1, 31.16579250707639: 1, 31.072431149616744: 1, 30.99064241739432: 1,
30.983758957615855: 1, 30.58923490940579: 1, 30.506073299105434: 1, 30.400580360334: 1,
30.396379852097937: 1, 30.296346834608745: 1, 29.990142419794: 1, 29.98896179291447: 1,
29.766568307587836: 1, 29.64191902938634: 1, 29.57459640755734: 1, 29.41218024096488: 1, 29.0986645
03350207: 1, 29.02131877321974: 1, 28.708434522203248: 1, 28.57031253288516: 1,
28.434733035850776: 1, 28.154178652270236: 1, 28.136964059667697: 1, 28.106915562558992: 1,
27.94304994237544: 1, 27.80239027449785: 1, 27.764964795627233: 1, 27.755942573683473: 1,
27.707188348316233: 1, 27.090124692603727: 1, 26.992046017066176: 1, 26.857821381082864: 1,
26.688748814010193: 1, 26.637792995215744: 1, 26.556158797878734: 1, 26.455239826277772: 1,
26.38145666539922: 1, 26.36317344847537: 1, 26.26063021776883: 1, 26.115953787128525: 1,
26.096361504778457: 1, 26.091553488971364: 1, 25.921303313307668: 1, 25.895886490298196: 1,
25.85796526724425: 1, 25.727492162694094: 1, 25.415375755975134: 1, 25.2563747547454: 1,
25.245087811985897: 1, 25.078249150601085: 1, 24.956240506127028: 1, 24.925246474345755: 1,
24.85967416057051: 1, 24.786964161412435: 1, 24.755069044002017: 1, 24.724218038663313: 1,
24.707196514868922: 1, 24.68135901876089: 1, 24.61606626898413: 1, 24.55175343731599: 1, 24.4430870
76359227: 1, 24.419705113070332: 1, 24.298094687189376: 1, 24.189808090817714: 1,
24.093224053831303: 1, 24.024383305883646: 1, 24.012235210387164: 1, 24.010296704759934: 1,
23.935616132602604: 1, 23.827256253204354: 1, 23.665694661832966: 1, 23.58565222053246: 1,
23.489126816691872: 1, 23.459475609652706: 1, 23.344810964561837: 1, 23.295427638532647: 1,
23.10262794505814: 1, 23.094486547381827: 1, 23.031457198162023: 1, 22.887938491288647: 1,
22.88110679703468: 1, 22.83804238029686: 1, 22.80747455024087: 1, 22.768836210395783: 1,
22.756815114970607: 1, 22.746309114064328: 1, 22.73508556879126: 1, 22.717539951294164: 1,
22.705587530947124: 1, 22.57195522002031: 1, 22.48182747941591: 1, 22.389703587543913: 1,
22.367015957017777: 1, 22.30729533049533: 1, 22.287529060615046: 1, 22.269066275246985: 1,
22.19211298889554: 1, 22.167867409723893: 1, 22.156634773170584: 1, 22.147228596876417: 1,
22.131506265720695: 1, 22.114293547761754: 1, 22.099743036932555: 1, 22.03544693770052: 1,
22.027062812768495: 1, 22.024049781466356: 1, 21.859595709022408: 1, 21.7828745282716: 1,
21.764295029400227: 1, 21.68540685622105: 1, 21.67602909052764: 1, 21.66684118075011: 1, 21.5561584
38205973: 1, 21.515744553033205: 1, 21.45103212179552: 1, 21.422306214096622: 1,
21.33453027790739: 1, 21.32081106378057: 1, 21.317842222050388: 1, 21.298661466171776: 1,
21.252589125954714: 1, 21.22663983589642: 1, 21.224331406282598: 1, 21.20990345194292: 1,
21.16038171079857: 1, 21.04562007834208: 1, 21.02225081163964: 1, 20.960303662871492: 1,
20.903806216796944: 1, 20.768305443739372: 1, 20.707549757314755: 1, 20.688351996023716: 1,
20.671834108716162: 1, 20.66609847179548: 1, 20.642463950306425: 1, 20.46907089173165: 1,
20.41165750432692: 1, 20.355994151909986: 1, 20.316451785130273: 1, 20.201582887978017: 1,
20.034269754792096: 1, 20.034185324289425: 1, 20.022137177602325: 1, 20.00344748388104: 1,
19.95074670004365: 1, 19.897884521659133: 1, 19.772849603075322: 1, 19.768471986421982: 1,
19.749039536166787: 1, 19.727916862316853: 1, 19.667125252518318: 1, 19.653664481086274: 1,
19.61488179163307: 1, 19.603995614879633: 1, 19.570511837976813: 1, 19.52878267364997: 1,
19.450435816842248: 1, 19.43934524209304: 1, 19.416326679911542: 1, 19.41411485104044: 1,
19.393234207765268: 1, 19.328498352008715: 1, 19.324627203247207: 1, 19.31729013669982: 1,
19.286160608296534: 1, 19.15907731007146: 1, 19.147073478964742: 1, 18.921476952152776: 1,
18.920058632809067: 1, 18.915534593775696: 1, 18.890151695740634: 1, 18.847169256917514: 1,
18.811839586399056: 1, 18.7774446213487: 1, 18.763911340623224: 1, 18.738570118842134: 1,
18.7191291487457: 1, 18.638312696559826: 1, 18.616694472476137: 1, 18.57666247396873: 1,
```

18.522950306380626: 1, 18.491549207135943: 1, 18.49122553654954: 1, 18.481718483669283: 1,
18.474443890134083: 1, 18.430618670845565: 1, 18.419962676924374: 1, 18.40725345388557: 1,
18.31152376623388: 1, 18.293979016733473: 1, 18.265790197226462: 1, 18.21070989712745: 1,
18.17622420640275: 1, 18.147884314758237: 1, 18.120015961842807: 1, 18.1094816222613: 1,
18.088777071270602: 1, 18.072249793542504: 1, 18.07081889854436: 1, 18.059926277492465: 1,
18.03812089649906: 1, 18.00066732352474: 1, 17.98380583792219: 1, 17.966916079846836: 1,
17.917834388356138: 1, 17.86222251812681: 1, 17.86164977432665: 1, 17.795545941704656: 1,
17.794270120708934: 1, 17.746881029069975: 1, 17.71665013701527: 1, 17.708891108873875: 1,
17.693395158012223: 1, 17.622897909354823: 1, 17.617152576615684: 1, 17.551433333558588: 1,
17.546231063697128: 1, 17.520109030847873: 1, 17.474294518723013: 1, 17.42039383939701: 1,
17.416885007441092: 1, 17.41078415274877: 1, 17.354861897866225: 1, 17.344016824871115: 1,
17.326440060254335: 1, 17.320382972133217: 1, 17.29280841273506: 1, 17.285661075323556: 1,
17.275996547105358: 1, 17.238108355656934: 1, 17.168193063301796: 1, 17.12993654629839: 1,
17.11199396998935: 1, 17.106995656909532: 1, 17.091280246290328: 1, 17.085257237099814: 1,
17.06213958501904: 1, 17.03972806877838: 1, 16.999442577502016: 1, 16.99114703498045: 1,
16.948919790319685: 1, 16.909319247961637: 1, 16.815868779916922: 1, 16.804896554883488: 1,
16.763140962911237: 1, 16.692816792276098: 1, 16.67937265119053: 1, 16.670979705624312: 1,
16.641788569779383: 1, 16.62802705618947: 1, 16.60395235509402: 1, 16.601129660030598: 1,
16.583808051309532: 1, 16.525473421210677: 1, 16.4946242042003: 1, 16.49427579912585: 1,
16.44060161472188: 1, 16.435251393991244: 1, 16.39452258868706: 1, 16.390285022710202: 1,
16.36392792114099: 1, 16.33295509214077: 1, 16.190066727017598: 1, 16.178170684997475: 1,
16.16013945707828: 1, 16.157736194343336: 1, 16.126353043166805: 1, 16.093457469244225: 1,
16.072723062034026: 1, 16.040084876967583: 1, 16.027499668964452: 1, 16.019697762823974: 1,
16.011933912442327: 1, 15.991840282322622: 1, 15.933862279221664: 1, 15.900501882802928: 1,
15.889021093632888: 1, 15.866001846393205: 1, 15.84809422184789: 1, 15.838512007308475: 1,
15.82540185764469: 1, 15.788315010990107: 1, 15.784103830682223: 1, 15.72550576949489: 1,
15.65473976670945: 1, 15.575791652424368: 1, 15.569148255049212: 1, 15.513465508814027: 1,
15.510909605188102: 1, 15.483402290997994: 1, 15.452436959696877: 1, 15.447861854505918: 1,
15.424756119853935: 1, 15.392091081683574: 1, 15.344659281828308: 1, 15.317659598671014: 1,
15.276932910796049: 1, 15.268998707076904: 1, 15.242901675937693: 1, 15.23494767413088: 1,
15.228014211370148: 1, 15.22295934470105: 1, 15.220456518241495: 1, 15.217583446336086: 1,
15.207731671766702: 1, 15.128680293673288: 1, 15.112424409596336: 1, 15.093016952744286: 1,
15.07025138926472: 1, 15.056764443859517: 1, 15.04317102492762: 1, 15.005536644456498: 1,
14.992063608330465: 1, 14.906530478556586: 1, 14.868310754058173: 1, 14.861852353182353: 1,
14.861834887927397: 1, 14.843670227564624: 1, 14.83810915227212: 1, 14.824858943538148: 1,
14.819318002373445: 1, 14.791490076104964: 1, 14.780295959851736: 1, 14.775460043966397: 1,
14.765160247047753: 1, 14.733213413500316: 1, 14.70175687882188: 1, 14.697825864851357: 1,
14.697652776586292: 1, 14.62357933184546: 1, 14.611939588433101: 1, 14.604169978720812: 1,
14.593304131468509: 1, 14.593087257842182: 1, 14.567702731158908: 1, 14.543698443695307: 1,
14.524583930690484: 1, 14.500597742062991: 1, 14.49474124206353: 1, 14.487160667435495: 1,
14.47176936713786: 1, 14.467996348784986: 1, 14.43847164090678: 1, 14.383585775981496: 1,
14.335415239527594: 1, 14.319320156562291: 1, 14.308020930849533: 1, 14.285739067441762: 1,
14.277511577664681: 1, 14.24998806028104: 1, 14.190514662616424: 1, 14.171499426743026: 1,
14.135375734648536: 1, 14.125769294347116: 1, 14.02841990162243: 1, 14.00597697691305: 1,
13.998781406060683: 1, 13.973917889723245: 1, 13.937096507458056: 1, 13.91833069665108: 1,
13.918029120588686: 1, 13.91600322295727: 1, 13.851762619365903: 1, 13.827025239837912: 1,
13.790963286772088: 1, 13.776163305241482: 1, 13.758134676561413: 1, 13.735773702402915: 1,
13.723120336980688: 1, 13.717150416706588: 1, 13.712745103884705: 1, 13.704674432266712: 1,
13.695897919876483: 1, 13.693967694708086: 1, 13.67949811727363: 1, 13.677703467533345: 1,
13.62855244342889: 1, 13.60930878566225: 1, 13.570882049881691: 1, 13.532492253026827: 1,
13.498857992645947: 1, 13.480224613075611: 1, 13.407800481327094: 1, 13.39379416996432: 1,
13.386085119460637: 1, 13.372050699748135: 1, 13.357183522051676: 1, 13.33529171901981: 1,
13.32580141243832: 1, 13.319445980260285: 1, 13.303251668309496: 1, 13.265611049560095: 1,
13.26489121548428: 1, 13.238821314453883: 1, 13.17467307836589: 1, 13.159692063994878: 1,
13.114638763033083: 1, 13.103109677646765: 1, 13.0973303479951: 1, 13.080747672085288: 1,
13.05042284668963: 1, 13.022185925011994: 1, 12.999700820083413: 1, 12.97563559380287: 1,
12.964762163509548: 1, 12.96255706079395: 1, 12.953942558968784: 1, 12.937529260273068: 1,
12.936154585512812: 1, 12.934721022191791: 1, 12.920125445977371: 1, 12.914189190375083: 1,
12.910099127809168: 1, 12.895515739179807: 1, 12.839790113087476: 1, 12.819879968205294: 1,
12.81664190358363: 1, 12.801703184008188: 1, 12.800096926763342: 1, 12.792695734883905: 1,
12.779195969647967: 1, 12.776015003893638: 1, 12.77531747274744: 1, 12.759024141644955: 1,
12.754697082688171: 1, 12.750687568929946: 1, 12.715297021644515: 1, 12.695824884683082: 1,
12.684637167778549: 1, 12.665730897025297: 1, 12.572758941765617: 1, 12.556126782806261: 1,
12.543781684188426: 1, 12.54016422390421: 1, 12.488248850167889: 1, 12.464918117210667: 1,
12.439176994080135: 1, 12.421181619950731: 1, 12.401719880289177: 1, 12.362892088075883: 1,
12.346622018539904: 1, 12.344480878445871: 1, 12.317547761117153: 1, 12.311804283465747: 1,
12.292783033710863: 1, 12.256490723916704: 1, 12.25437794231473: 1, 12.2499486916633: 1,
12.236873906995609: 1, 12.232926603620806: 1, 12.21987852780625: 1, 12.219231271261483: 1,
12.212998918995186: 1, 12.212415130914009: 1, 12.186825838732775: 1, 12.163553078856935: 1,
12.161586470665148: 1, 12.143730848481317: 1, 12.135667536659362: 1, 12.119953809903722: 1,
12.097503887170085: 1, 12.096339288455507: 1, 12.091119331969566: 1, 12.088739944144892: 1,
12.086612894890749: 1, 12.072359200950975: 1, 12.071462973795548: 1, 12.04429646661245: 1,
12.039125181842827: 1, 12.03595680147334: 1, 12.025019901665548: 1, 11.963040095372477: 1,
11.958297990661539: 1, 11.956169601768242: 1, 11.906015377779115: 1, 11.905851605548756: 1,
11.885973047015142: 1, 11.853060754027913: 1, 11.801228634541246: 1, 11.762226653014437: 1,
11.747513577702954: 1, 11.72401720092068: 1, 11.68114904654508: 1, 11.640307070112518: 1,
11.629839958884853: 1, 11.621994903894311: 1, 11.605926502101884: 1, 11.594499754213565: 1,

11.582521238516515: 1, 11.577399718315677: 1, 11.509921900168381: 1, 11.509778963058976: 1,
11.486127065373351: 1, 11.478404470646653: 1, 11.458908796450066: 1, 11.448775660722482: 1,
11.416320691773683: 1, 11.401399500249445: 1, 11.381925662785656: 1, 11.351352102415637: 1,
11.34840799198818: 1, 11.346859748069305: 1, 11.31816014606133: 1, 11.302450945297267: 1,
11.296103840617246: 1, 11.293632397828782: 1, 11.2890863932487: 1, 11.275730974470944: 1,
11.272779397271584: 1, 11.261933339444893: 1, 11.230598470159139: 1, 11.226224881993295: 1,
11.217564694652987: 1, 11.214706629400782: 1, 11.20632318634811: 1, 11.164291660142966: 1,
11.159707351624597: 1, 11.152602155688964: 1, 11.119216167889551: 1, 11.101635386619732: 1,
11.06922912226289: 1, 11.067434122360542: 1, 11.065024588439474: 1, 11.064253319288122: 1,
11.042155585403897: 1, 11.031888577058409: 1, 11.030996733470273: 1, 11.030244322516664: 1,
10.991906257840803: 1, 10.991860779224421: 1, 10.986753480067563: 1, 10.981299472883386: 1,
10.968342176346027: 1, 10.967131051501704: 1, 10.961336888379433: 1, 10.954830527325836: 1,
10.940989799725145: 1, 10.925347519255828: 1, 10.91315296071978: 1, 10.898054872941438: 1,
10.889293065855199: 1, 10.887379313324598: 1, 10.854530823716365: 1, 10.848650107451165: 1,
10.848523170965478: 1, 10.808460565884747: 1, 10.801287272684307: 1, 10.78645181157982: 1,
10.78431890366298: 1, 10.780332727460596: 1, 10.778701631173268: 1, 10.773303341617131: 1,
10.758798026883623: 1, 10.73927581904265: 1, 10.726899704583317: 1, 10.70967521619281: 1,
10.668917806380545: 1, 10.658160928660527: 1, 10.65623318514577: 1, 10.62598908921037: 1,
10.617760080393783: 1, 10.616762745508732: 1, 10.589752325218873: 1, 10.581154684202252: 1,
10.580431002682058: 1, 10.565102157517957: 1, 10.541235652309005: 1, 10.536032273296046: 1,
10.532334133339768: 1, 10.526653453435372: 1, 10.525725474727372: 1, 10.500803197647961: 1,
10.493858790685842: 1, 10.478170504433349: 1, 10.463239759681718: 1, 10.432517119650994: 1,
10.411347232502628: 1, 10.39991090891269: 1, 10.390979787093508: 1, 10.365037674233806: 1,
10.361719528906441: 1, 10.346499749577113: 1, 10.331581570113396: 1, 10.329079966142332: 1,
10.280378394654914: 1, 10.277006285663656: 1, 10.265065622949521: 1, 10.249703422409953: 1,
10.218578007338222: 1, 10.215003828175291: 1, 10.208298158372152: 1, 10.205464709925257: 1,
10.185711832884412: 1, 10.170299415024783: 1, 10.16734671463483: 1, 10.164181912401764: 1,
10.1531867614583: 1, 10.132456305356905: 1, 10.123398294595328: 1, 10.109831728347224: 1,
10.106353605779374: 1, 10.103658615161628: 1, 10.093312869724482: 1, 10.091592109941798: 1,
10.065977128557359: 1, 10.060742548260157: 1, 10.06070937045332: 1, 10.040305466632422: 1,
10.040026240118339: 1, 10.035325135890028: 1, 10.028456303605434: 1, 10.00756657175538: 1,
10.005647937420227: 1, 9.995815940992657: 1, 9.992927166971615: 1, 9.990776617110486: 1, 9.98681124
173374: 1, 9.975420802132263: 1, 9.969638125250126: 1, 9.965169740001633: 1, 9.917042716249796: 1,
9.9168029153225: 1, 9.90033857171854: 1, 9.894937146980856: 1, 9.887690458729455: 1,
9.883566583669555: 1, 9.882815141654833: 1, 9.875962500732172: 1, 9.870917015761778: 1,
9.866635369183228: 1, 9.841448900039644: 1, 9.83634850845781: 1, 9.827494057214135: 1,
9.824740399692917: 1, 9.821699213171492: 1, 9.821084227075561: 1, 9.808932529338184: 1,
9.80615129593439: 1, 9.778273040569763: 1, 9.771154995207711: 1, 9.766183132304633: 1,
9.764541366326267: 1, 9.76396926217242: 1, 9.762033914830809: 1, 9.760439875757395: 1,
9.756565610736782: 1, 9.752549255593244: 1, 9.71776602190895: 1, 9.713453780116254: 1,
9.694789151428806: 1, 9.69287389587744: 1, 9.68597258983315: 1, 9.681339645245378: 1,
9.678843495815165: 1, 9.655712961344266: 1, 9.654457499510393: 1, 9.633860887707813: 1,
9.618893503687355: 1, 9.607503848499883: 1, 9.545348804378094: 1, 9.525105557159078: 1,
9.523166932044346: 1, 9.5193870071729: 1, 9.517023345277527: 1, 9.51681934653088: 1,
9.498057855404491: 1, 9.494014248163579: 1, 9.47220501066618: 1, 9.471804854427244: 1,
9.468937551442025: 1, 9.453937100763909: 1, 9.442936947616518: 1, 9.429896265683107: 1,
9.422140525606057: 1, 9.42097194197252: 1, 9.415618751569045: 1, 9.411698023846835: 1,
9.410195838364942: 1, 9.408872053504878: 1, 9.406771268825896: 1, 9.401648764221354: 1,
9.392511696390844: 1, 9.368130328388345: 1, 9.367331959637722: 1, 9.361492886225044: 1,
9.353222388720274: 1, 9.334342682160697: 1, 9.319392169102864: 1, 9.319296371048518: 1,
9.311261892709814: 1, 9.300592189925524: 1, 9.26782324732895: 1, 9.265521086840831: 1,
9.256296794133666: 1, 9.254102606024757: 1, 9.231236416905366: 1, 9.218233904348782: 1,
9.212143624873253: 1, 9.210565845735099: 1, 9.200267471274516: 1, 9.193356738004763: 1,
9.179920961441459: 1, 9.15141669390554: 1, 9.13404629363117: 1, 9.116117546766485: 1,
9.109598727549068: 1, 9.105125887325727: 1, 9.095603980260146: 1, 9.083422795737961: 1,
9.07125372586111: 1, 9.06981153822345: 1, 9.061665926408939: 1, 9.05350022070317: 1,
9.045748886885688: 1, 9.04249934433653: 1, 9.042247324746917: 1, 9.030226763669518: 1,
9.028447680471226: 1, 9.023960696767944: 1, 9.005973087260347: 1, 8.994940749163495: 1,
8.993943280078275: 1, 8.985945967067622: 1, 8.980816985780915: 1, 8.979180772561683: 1,
8.95844276019213: 1, 8.931644469390084: 1, 8.929794425868407: 1, 8.921235854196846: 1,
8.897562693813127: 1, 8.88221282851971: 1, 8.874044602836358: 1, 8.852158830573703: 1,
8.842222052901244: 1, 8.828714041670962: 1, 8.822892326947526: 1, 8.814319199510983: 1,
8.812219177077154: 1, 8.81014557370072: 1, 8.80951362368868: 1, 8.794851945767848: 1,
8.783911943972312: 1, 8.76582431380603: 1, 8.760600855565166: 1, 8.73330288579723: 1,
8.72152063279935: 1, 8.717959812234795: 1, 8.71597985260362: 1, 8.71595658049517: 1,
8.715262422700471: 1, 8.7083251024321: 1, 8.690883974762219: 1, 8.680643956692085: 1,
8.677888027146516: 1, 8.667907789626298: 1, 8.627734717831459: 1, 8.614644399961625: 1,
8.597045602800728: 1, 8.594016292228542: 1, 8.578119279320774: 1, 8.573852924428923: 1,
8.565966889398226: 1, 8.558158498713022: 1, 8.509581742434285: 1, 8.508799825688175: 1,
8.507142051637272: 1, 8.484080265929624: 1, 8.457569171088556: 1, 8.440404102087669: 1,
8.432681513769424: 1, 8.417910895016504: 1, 8.416407627432392: 1, 8.396498720987214: 1,
8.383816485186733: 1, 8.37702835904455: 1, 8.3730601104248: 1, 8.372450193012948: 1,
8.3670646301482: 1, 8.365371810466558: 1, 8.32267056379846: 1, 8.318779069574038: 1,
8.314761107051703: 1, 8.309536721209135: 1, 8.299076238738985: 1, 8.293731396417188: 1,
8.291642580387796: 1, 8.288348421123692: 1, 8.27863341808883: 1, 8.259446349550036: 1,
8.241506885828679: 1, 8.210469538755834: 1, 8.197899211582637: 1, 8.18210296074383: 1,
8.181785864559247: 1, 8.16823063224785: 1, 8.155968144289362: 1, 8.150646193171196: 1,

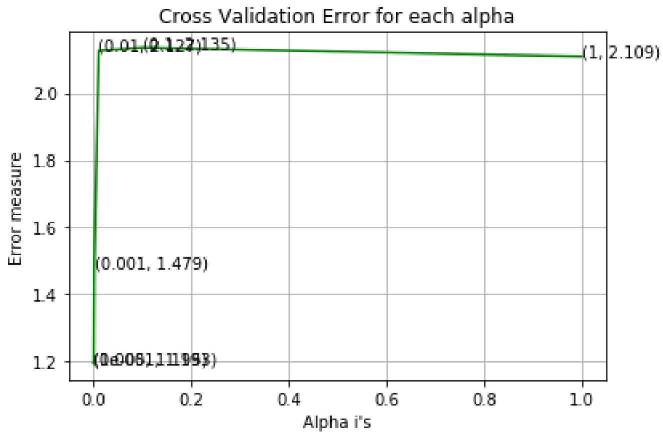
```
8.148025789346729: 1, 8.146914737635004: 1, 8.142891409187442: 1, 8.141836131229715: 1,
8.136536365881577: 1, 8.135973158828413: 1, 8.126488483698749: 1, 8.097486625333808: 1,
8.09260680153918: 1, 8.083716975056655: 1, 8.080851420685809: 1, 8.06398737609208: 1,
8.05374975592551: 1, 8.050800992510768: 1, 8.034126728010579: 1, 8.033970771217211: 1,
8.028538765200867: 1, 8.020890819412783: 1, 8.005586289768146: 1, 8.00041048463262: 1,
7.984131911945377: 1, 7.971757897040209: 1, 7.969687702554294: 1, 7.961238839257601: 1,
7.943654356788224: 1, 7.943203024331166: 1, 7.941794550065905: 1, 7.932059874489016: 1,
7.902451762062049: 1, 7.893846136129829: 1, 7.857107604749754: 1, 7.841711240356917: 1,
7.823945637500523: 1, 7.809193617298659: 1, 7.7926185737839155: 1, 7.792305672742989: 1,
7.785370782889428: 1, 7.782550829768839: 1, 7.76466294367917: 1, 7.745064701849743: 1,
7.728373272335079: 1, 7.710503231913694: 1, 7.708820053378106: 1, 7.702332336683378: 1,
7.652432431651078: 1, 7.650785176259681: 1, 7.630694489252299: 1, 7.608047256277732: 1,
7.594316942010362: 1, 7.535391127574086: 1, 7.535244380365453: 1, 7.516868742526767: 1,
7.497483929582202: 1, 7.486829773006895: 1, 7.457679136136263: 1, 7.443617689568765: 1,
7.414088838597499: 1, 7.408316483403298: 1, 7.38358084939811: 1, 7.380224929710444: 1,
7.354692973304132: 1, 7.348609941333383: 1, 7.344485504573466: 1, 7.335036211385497: 1,
7.300738081709509: 1, 7.290561589468028: 1, 7.2387363782572205: 1, 7.183192209456369: 1,
7.178779347285485: 1, 7.157979749188171: 1, 7.110575451168664: 1, 7.100869251633812: 1,
7.096236318148838: 1, 7.077845827495886: 1, 7.0665668943720235: 1, 7.063811232711551: 1,
7.0624758816127375: 1, 7.059152872275645: 1, 7.055797371398173: 1, 7.0185703704659606: 1,
7.008688087648029: 1, 6.93673815217141: 1, 6.843130043893411: 1, 6.8427070470506965: 1,
6.836720529807838: 1, 6.6590467394045: 1, 6.6554172615898395: 1, 6.651204543225183: 1,
6.622007468960565: 1, 6.547120003039487: 1, 6.505370486222713: 1, 6.379743335599716: 1,
5.991741667491071: 1})
```

In [87]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_text_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.1951053209037306
For values of alpha =  0.0001 The log loss is: 1.192558246646385
For values of alpha =  0.001 The log loss is: 1.4789213142926045
For values of alpha =  0.01 The log loss is: 2.1266383196385132
For values of alpha =  0.1 The log loss is: 2.135460760997863
For values of alpha =  1 The log loss is: 2.1090231650341518
```




```
For values of best alpha =  0.0001 The train log loss is: 0.8475477145649266
For values of best alpha =  0.0001 The cross validation log loss is: 1.192558246646385
For values of best alpha =  0.0001 The test log loss is: 1.0070771684084348
```

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

Ans. Yes, it seems like!

In [0]:

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

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

In [89]:

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

```
3.453 % of word of test data appeared in train data
3.801 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [0]:

```
#Data preparation for ML models.
#Misc. functionns for ML models
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [0]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [0]:

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

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

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

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

    print("Out of the top ",no_features," features ", word_present, "are present in query point")
```

Stacking all Text and numerical Features

Normalizing all the Numerical Features

Word_Count

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing



word_count_train = train_df['word_count']
min_max_scaler_train = preprocessing.MinMaxScaler()
word_count_train = min_max_scaler_train.fit_transform(word_count_train.values.reshape(-1,1))

word_count_test = test_df['word_count']
min_max_scaler_test = preprocessing.MinMaxScaler()
word_count_test = min_max_scaler_test.fit_transform(word_count_test.values.reshape(-1,1))

word_count_cv = cv_df['word_count']
min_max_scaler_cv = preprocessing.MinMaxScaler()
word_count_cv = min_max_scaler_cv.fit_transform(word_count_cv.values.reshape(-1,1))


import scipy
word_count_train = scipy.sparse.csr_matrix(word_count_train)
word_count_test = scipy.sparse.csr_matrix(word_count_test)
word_count_cv = scipy.sparse.csr_matrix(word_count_cv)
```

Char_Count

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing



char_count_train = train_df['char_count']
min_max_scaler_train = preprocessing.MinMaxScaler()
char_count_train = min_max_scaler_train.fit_transform(char_count_train.values.reshape(-1,1))

char_count_test = test_df['char_count']
min_max_scaler_test = preprocessing.MinMaxScaler()
char_count_test = min_max_scaler_test.fit_transform(char_count_test.values.reshape(-1,1))

char_count_cv = cv_df['char_count']
min_max_scaler_cv = preprocessing.MinMaxScaler()
char_count_cv = min_max_scaler_cv.fit_transform(char_count_cv.values.reshape(-1,1))


import scipy
char_count_train = scipy.sparse.csr_matrix(char_count_train)
char_count_test = scipy.sparse.csr_matrix(char_count_test)
char_count_cv = scipy.sparse.csr_matrix(char_count_cv)
```

Word_Density

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing



word_density_count_train = train_df['word_density']
min_max_scaler_train = preprocessing.MinMaxScaler()
word_density_count_train =
min_max_scaler_train.fit_transform(word_density_count_train.values.reshape(-1,1))

word_density_count_test = test_df['word_density']
min_max_scaler_test = preprocessing.MinMaxScaler()
word_density_count_test = min_max_scaler_test.fit_transform(word_density_count_test.values.reshape
(-1,1))
```

```
(-1,1))

word_density_count_cv = cv_df['word_density']
min_max_scaler_cv = preprocessing.MinMaxScaler()
word_density_count_cv = min_max_scaler_cv.fit_transform(word_density_count_cv.values.reshape(-1,1))


import scipy
word_density_count_train = scipy.sparse.csr_matrix(word_density_count_train)
word_density_count_test = scipy.sparse.csr_matrix(word_density_count_test)
word_density_count_cv = scipy.sparse.csr_matrix(word_density_count_cv)
```

No.of.digits

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing



digits_count_train = train_df['No.of.digits']
min_max_scaler_train = preprocessing.MinMaxScaler()
digits_count_train = min_max_scaler_train.fit_transform(digits_count_train.values.reshape(-1,1))

digits_count_test = test_df['No.of.digits']
min_max_scaler_test = preprocessing.MinMaxScaler()
digits_count_test = min_max_scaler_test.fit_transform(digits_count_test.values.reshape(-1,1))

digits_count_cv = cv_df['No.of.digits']
min_max_scaler_cv = preprocessing.MinMaxScaler()
digits_count_cv = min_max_scaler_cv.fit_transform(digits_count_cv.values.reshape(-1,1))


import scipy
digits_count_train = scipy.sparse.csr_matrix(digits_count_train)
digits_count_test = scipy.sparse.csr_matrix(digits_count_test)
digits_count_cv = scipy.sparse.csr_matrix(digits_count_cv)
```

Gene_Text

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing



gene_text_count_train = train_df['gene_text']
min_max_scaler_train = preprocessing.MinMaxScaler()
gene_text_count_train = min_max_scaler_train.fit_transform(gene_text_count_train.values.reshape(-1,
1))

gene_text_count_test = test_df['gene_text']
min_max_scaler_test = preprocessing.MinMaxScaler()
gene_text_count_test = min_max_scaler_test.fit_transform(gene_text_count_test.values.reshape(-1,1))

gene_text_count_cv = cv_df['gene_text']
min_max_scaler_cv = preprocessing.MinMaxScaler()
gene_text_count_cv = min_max_scaler_cv.fit_transform(gene_text_count_cv.values.reshape(-1,1))


import scipy
gene_text_count_train = scipy.sparse.csr_matrix(gene_text_count_train)
gene_text_count_test = scipy.sparse.csr_matrix(gene_text_count_test)
gene_text_count_cv = scipy.sparse.csr_matrix(gene_text_count_cv)
```

Variation_Text

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

variation_text_count_train = train_df['variation_text']
min_max_scaler_train = preprocessing.MinMaxScaler()
variation_text_count_train = min_max_scaler_train.fit_transform(variation_text_count_train.values.
reshape(-1,1))

variation_text_count_test = test_df['variation_text']
min_max_scaler_test = preprocessing.MinMaxScaler()
variation_text_count_test =
min_max_scaler_test.fit_transform(variation_text_count_test.values.reshape(-1,1))

variation_text_count_cv = cv_df['variation_text']
min_max_scaler_cv = preprocessing.MinMaxScaler()
variation_text_count_cv = min_max_scaler_cv.fit_transform(variation_text_count_cv.values.reshape(-
1,1))


import scipy
variation_text_count_train = scipy.sparse.csr_matrix(variation_text_count_train)
variation_text_count_test = scipy.sparse.csr_matrix(variation_text_count_test)
variation_text_count_cv = scipy.sparse.csr_matrix(variation_text_count_cv)
```

Capital_Count

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing

capital_count_count_train = train_df['capital_count']
min_max_scaler_train = preprocessing.MinMaxScaler()
capital_count_count_train = min_max_scaler_train.fit_transform(capital_count_count_train.values.re
shape(-1,1))

capital_count_count_test = test_df['capital_count']
min_max_scaler_test = preprocessing.MinMaxScaler()
capital_count_count_test = min_max_scaler_test.fit_transform(capital_count_count_test.values.reshap
e(-1,1))

capital_count_count_cv = cv_df['capital_count']
min_max_scaler_cv = preprocessing.MinMaxScaler()
capital_count_count_cv = min_max_scaler_cv.fit_transform(capital_count_count_cv.values.reshape(-1,
1))


import scipy
capital_count_train = scipy.sparse.csr_matrix(capital_count_count_train)
capital_count_test = scipy.sparse.csr_matrix(capital_count_count_test)
capital_count_cv = scipy.sparse.csr_matrix(capital_count_count_cv)
```

Concatenating all the Features

In [0]:

```
# merging gene, variance and text features

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


train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)
)
```

```
)

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,
train_text_feature_onehotCoding,word_count_train,char_count_train,word_density_count_train,digits_c
ount_train,gene_text_count_train,variation_text_count_train,capital_count_train)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding,
test_text_feature_onehotCoding,word_count_test,char_count_test,word_density_count_test,digits_count
_test,gene_text_count_test,variation_text_count_test,capital_count_test)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding,word_count_cv,c
har_count_cv,word_density_count_cv,digits_count_cv,gene_text_count_cv,variation_text_count_cv,capi
tal_count_cv)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding =
np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding =
np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
cv_gene_var_responseCoding =
np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,
train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding)
)
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
```

In [101]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding
.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3216)
(number of data points * number of features) in test data =  (665, 3216)
(number of data points * number of features) in cross validation data = (532, 3216)
```

In [102]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shap
e)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_responseCoding.shape)
```

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

Assingnment-1:Tfidf with 1000 features

4.1. Base Line Model

4.1.1. Naive Bayes with Tfidf(max_features=1000)

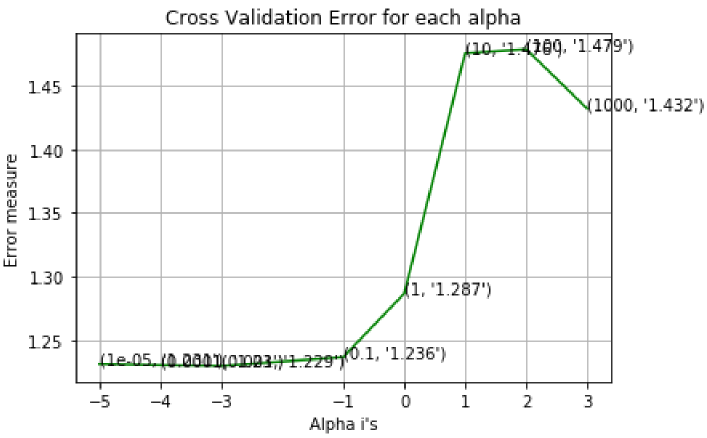
4.1.1.1. Hyper parameter tuning

In [103]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
```

```
Log Loss : 1.231024121070405
for alpha = 0.0001
Log Loss : 1.2301580949630948
for alpha = 0.001
Log Loss : 1.2294801111951885
for alpha = 0.1
Log Loss : 1.2364391116997018
for alpha = 1
Log Loss : 1.2865553391976066
for alpha = 10
Log Loss : 1.4759056555783814
for alpha = 100
Log Loss : 1.4791008174310993
for alpha = 1000
Log Loss : 1.432345055549656
```




```
For values of best alpha =  0.001 The train log loss is: 0.5062829293471407
For values of best alpha =  0.001 The cross validation log loss is: 1.2294801111951885
For values of best alpha =  0.001 The test log loss is: 1.1793586172336685
```

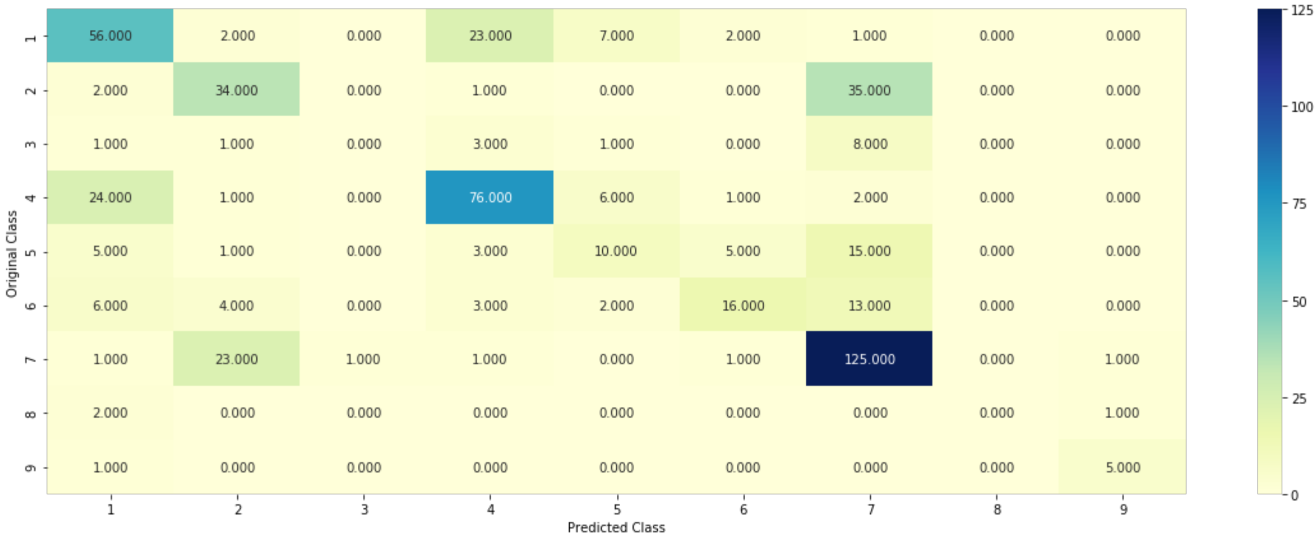
4.1.1.2. Testing the model with best hyper paramters

In [104]:

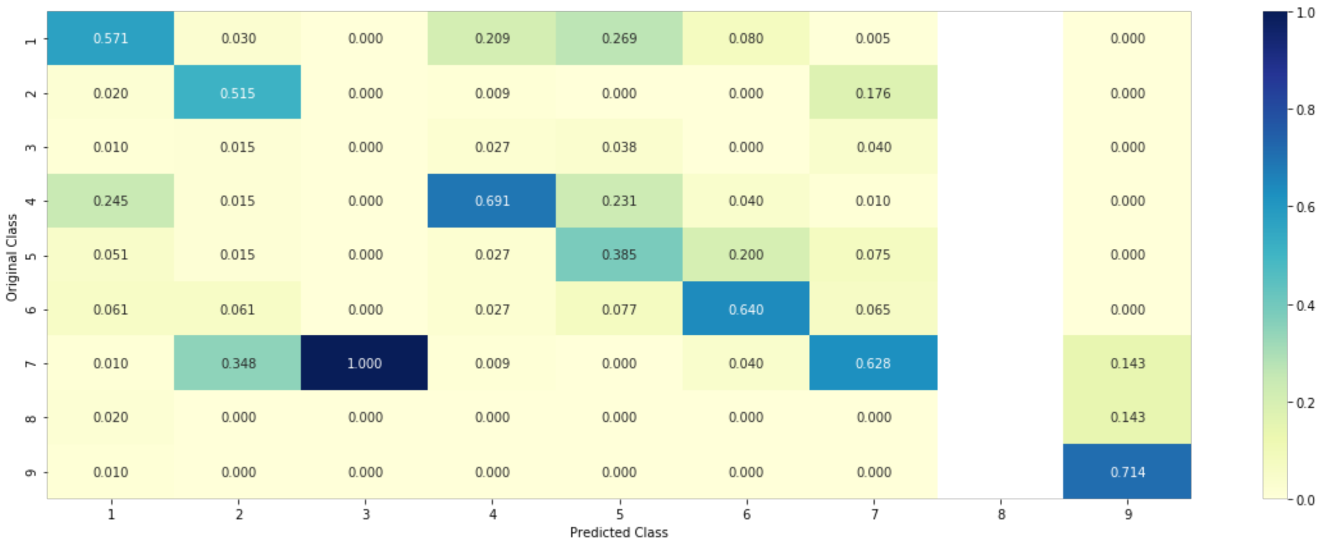
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


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

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

```
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv
_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

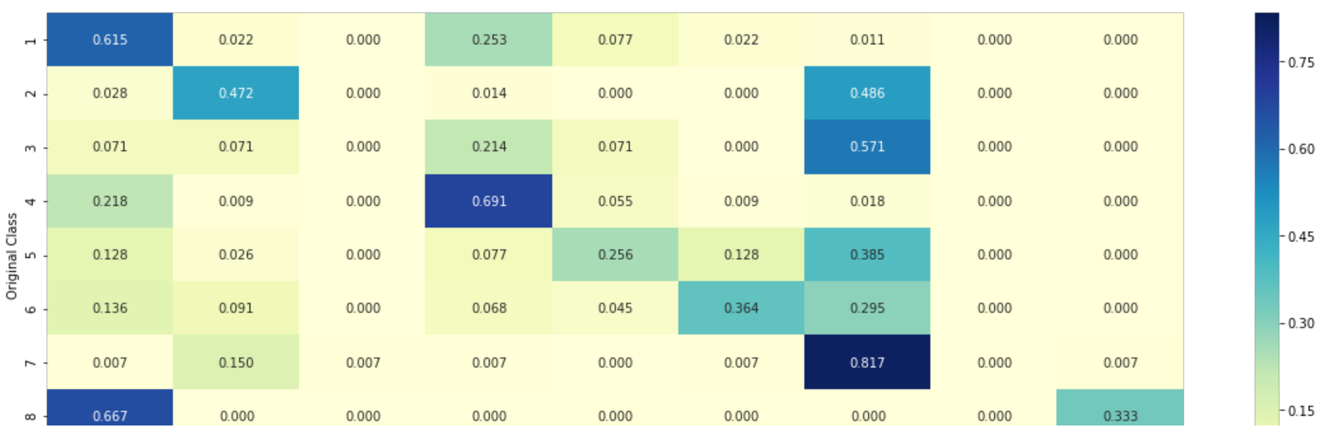
```
Log Loss : 1.2294801111951885
Number of missclassified point : 0.39473684210526316
-------------------- Confusion matrix --------------------
```

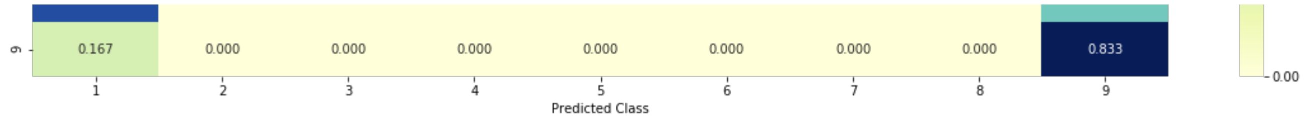



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.1.1.3. Feature Importance, Correctly classified point

In [105]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0608 0.0611 0.0105 0.0592 0.0315 0.0302 0.7407 0.0036 0.0024]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [106]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3839 0.0459 0.0123 0.3926 0.0352 0.0337 0.0895 0.0041 0.0028]]
Actual Class : 4
--------------------------------------------------
48 Text feature [005] present in test data point [True]
56 Text feature [023] present in test data point [True]
72 Text feature [004] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.2. K Nearest Neighbour Classification with Tfidf(max_features=1000)

4.2.1. Hyper parameter tuning

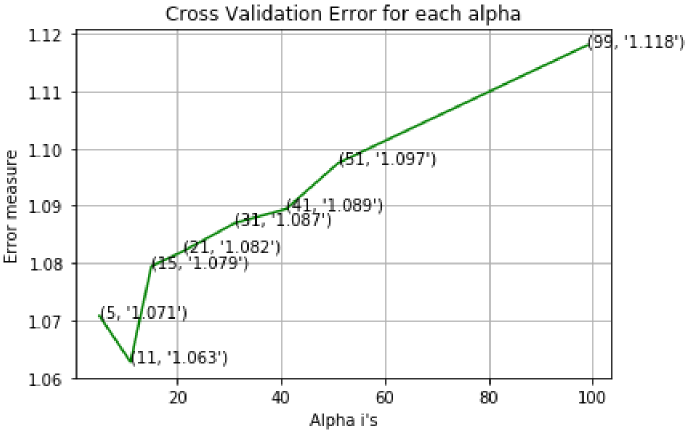
In [107]:

```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
```

```
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0707550396622976
for alpha = 11
Log Loss : 1.062728297002352
for alpha = 15
Log Loss : 1.0794394262597726
for alpha = 21
Log Loss : 1.0819931425306866
for alpha = 31
Log Loss : 1.0869382227255342
for alpha = 41
```

```
for alpha = 41
Log Loss : 1.0894380725675836
for alpha = 51
Log Loss : 1.0974509473765848
for alpha = 99
Log Loss : 1.1180215253773824
```



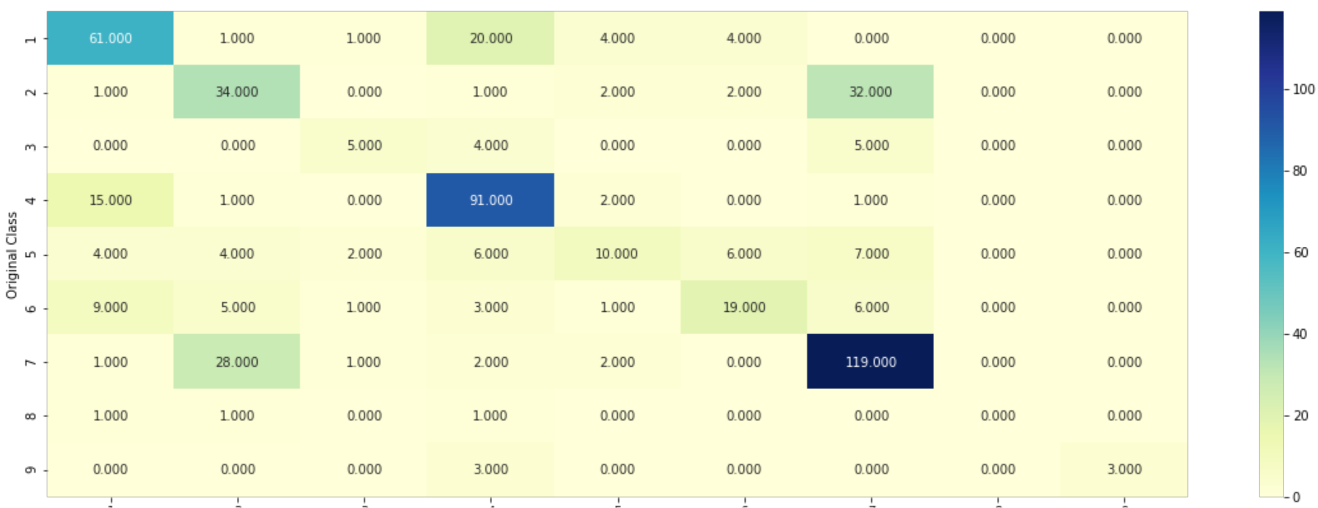

```
For values of best alpha =  11 The train log loss is: 0.6400110184426363
For values of best alpha =  11 The cross validation log loss is: 1.062728297002352
For values of best alpha =  11 The test log loss is: 1.0971646510061364
```

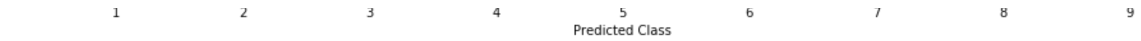
4.2.2. Testing the model with best hyper paramters

In [108]:

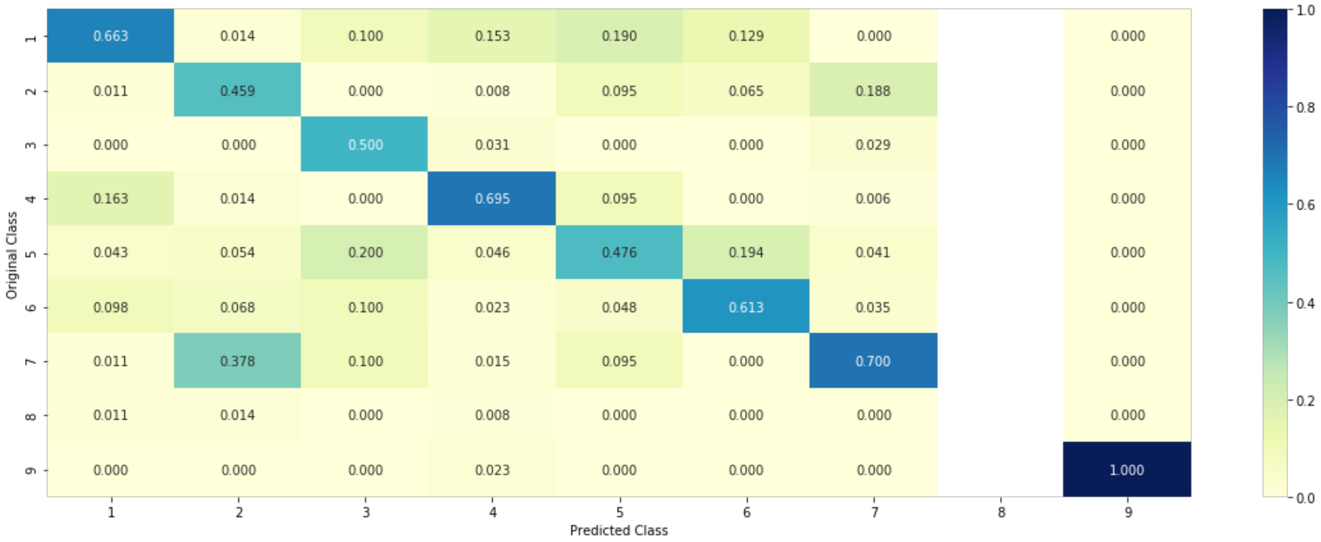
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.062728297002352
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```

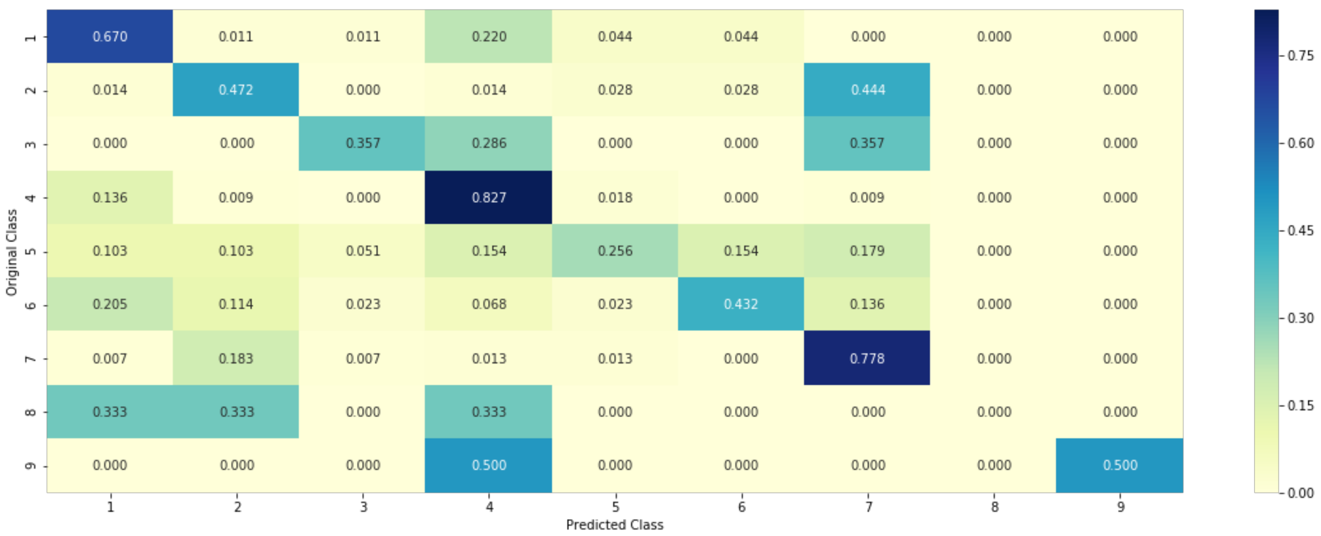





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.2.3.Sample Query point -1

In [109]:

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

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

```
Predicted Class : 4
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 7 7 7 5 5 7 7 2]
Fequency of nearest points : Counter({7: 8, 5: 2, 2: 1})
```

4.2.4. Sample Query Point-2

In [110]:

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

test_point_index = 100

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

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [4 4 4
4 3 4 4 4 4 1 1]
Fequency of nearest points : Counter({4: 8, 1: 2, 3: 1})
```

4.3. Logistic Regression with Tfidf(max_features=1000)

4.3.1. With Class balancing

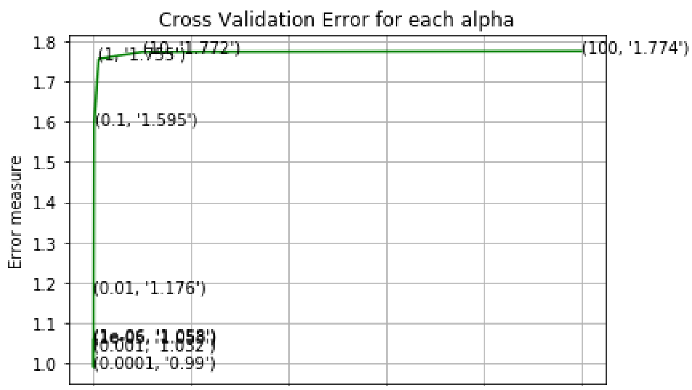
4.3.1.1. Hyper paramter tuning

In [111]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------


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

```python
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.0584006207244414
for alpha = 1e-05
Log Loss : 1.0530664029791157
for alpha = 0.0001
Log Loss : 0.9897400673645114
for alpha = 0.001
Log Loss : 1.032498525123867
for alpha = 0.01
Log Loss : 1.1756371704412885
for alpha = 0.1
Log Loss : 1.5954330152101732
for alpha = 1
Log Loss : 1.7551413782401912
for alpha = 10
Log Loss : 1.7724739176322066
for alpha = 100
Log Loss : 1.7742762749904777
```



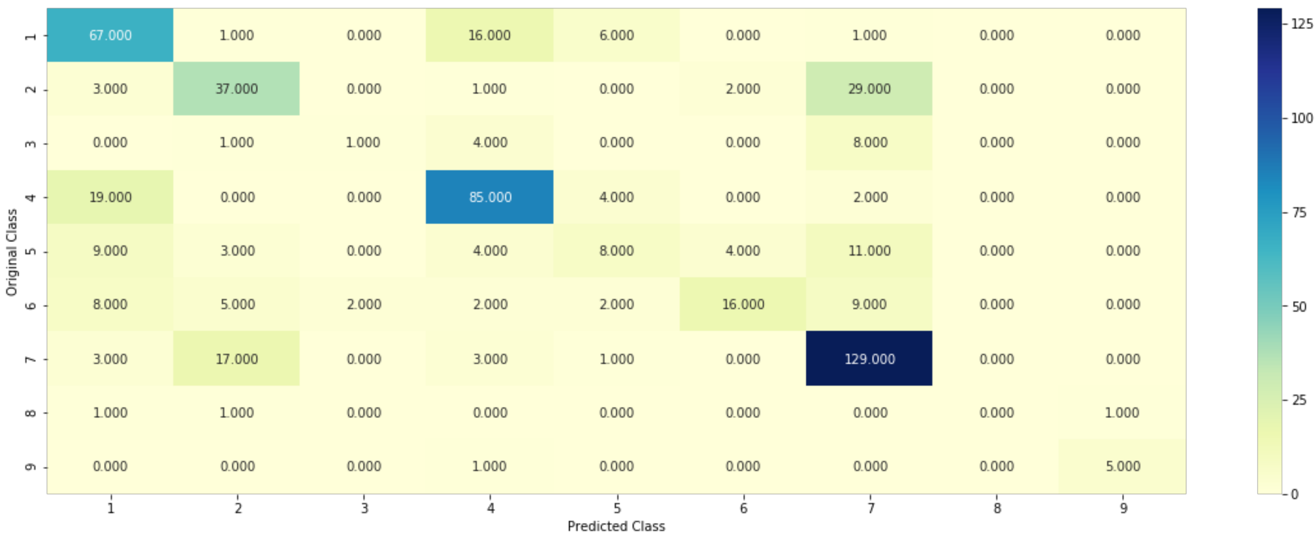

```
For values of best alpha =  0.0001 The train log loss is: 0.44275800579830693
For values of best alpha =  0.0001 The cross validation log loss is: 0.9897400673645114
For values of best alpha =  0.0001 The test log loss is: 0.9675073176722463
```

4.3.1.2. Testing the model with best hyper paramters

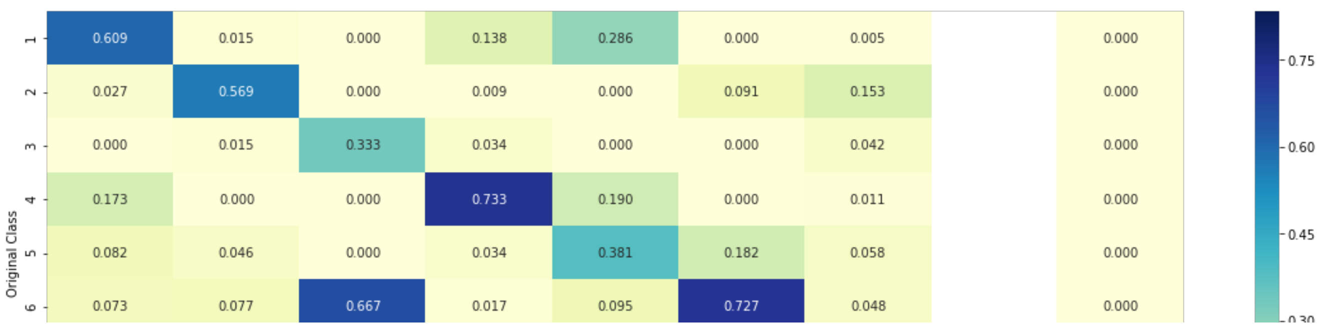
In [112]:

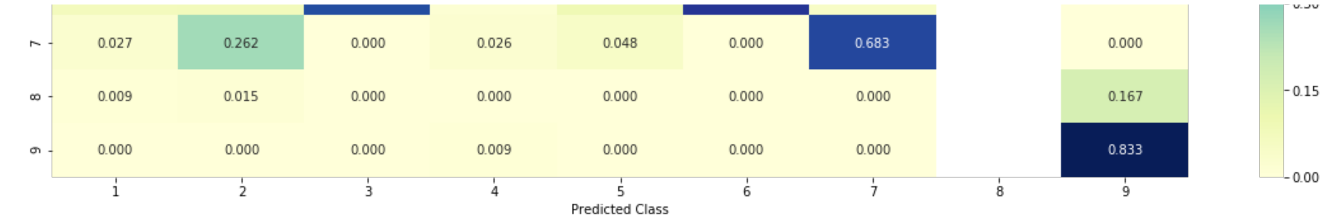
```python
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 0.9897400673645114
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```

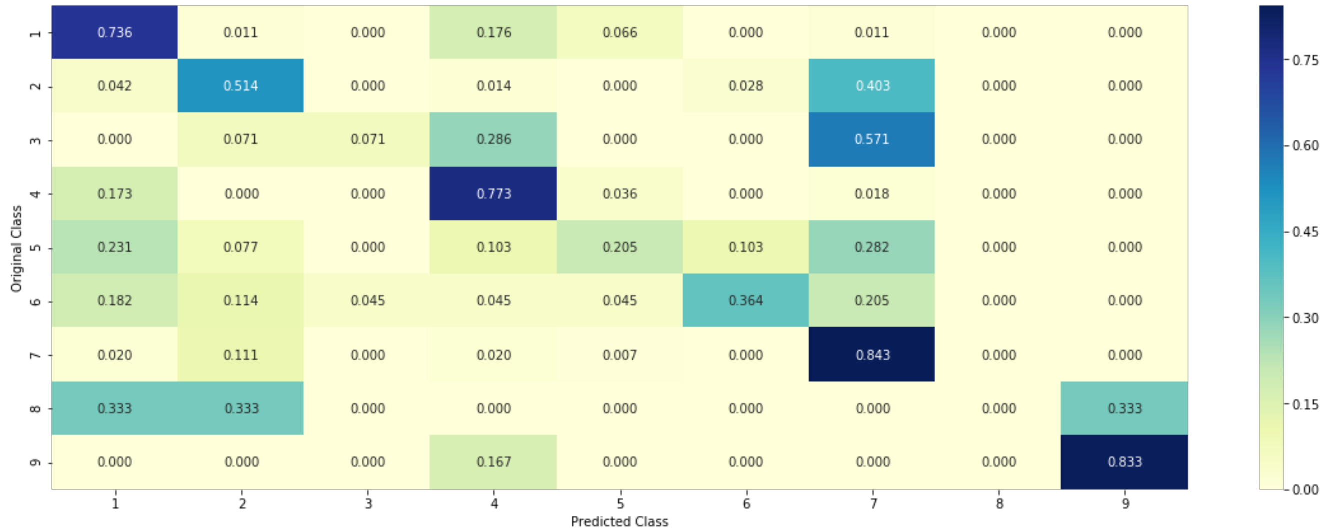



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



4.3.1.3. Feature Importance

In [0]:

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

4.3.1.3.1. Correctly Classified point

In [114]:

```python
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1056 0.1164 0.0058 0.0051 0.0724 0.1192 0.5652 0.005  0.0053]]
Actual Class : 7
--------------------------------------------------
317 Text feature [1251] present in test data point [True]
393 Text feature [00] 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 [115]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3907 0.0271 0.0361 0.4622 0.0284 0.0241 0.0177 0.0066 0.0071]]
Actual Class : 4
--------------------------------------------------
180 Text feature [100] present in test data point [True]
396 Text feature [042] present in test data point [True]
407 Text feature [01] present in test data point [True]
456 Text feature [045] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.2. Without Class balancing

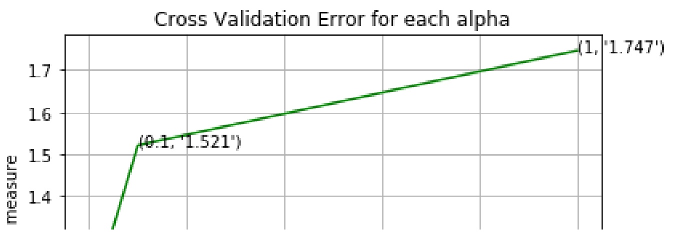
4.3.2.1. Hyper paramter tuning

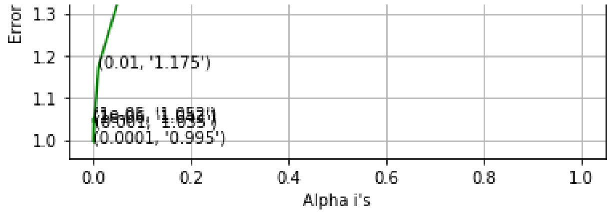
In [116]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------



# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
```

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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.0433086534462728
for alpha = 1e-05
Log Loss : 1.0516300171222936
for alpha = 0.0001
Log Loss : 0.9946436642332478
for alpha = 0.001
Log Loss : 1.0346534053912135
for alpha = 0.01
Log Loss : 1.1752409974722018
for alpha = 0.1
Log Loss : 1.5213641825236772
for alpha = 1
Log Loss : 1.7465129285724361
```





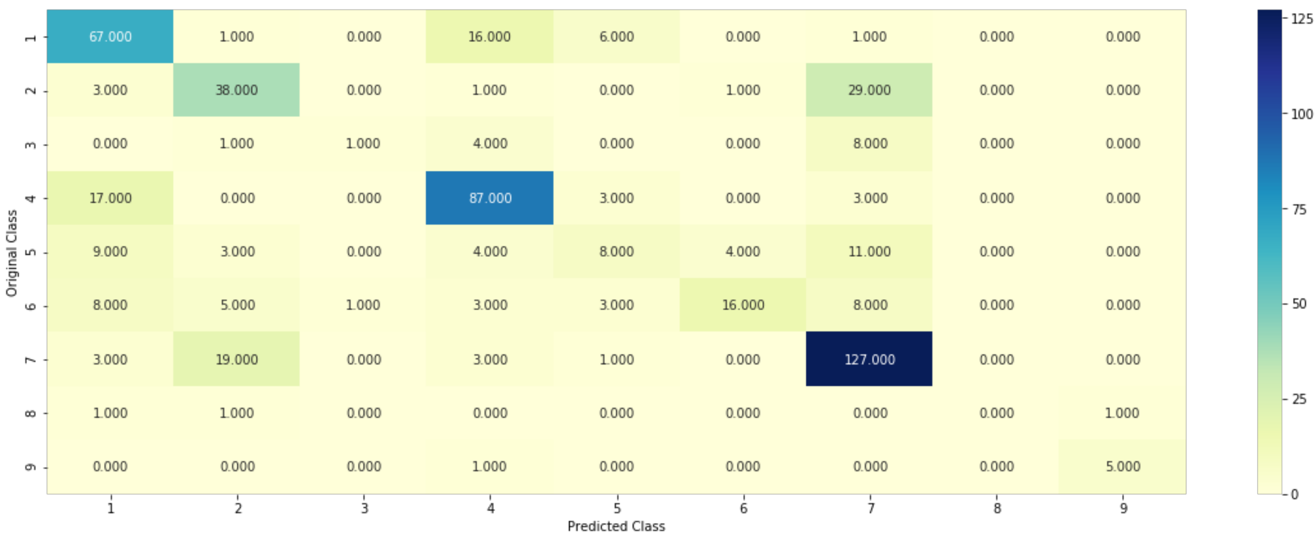

```
For values of best alpha =  0.0001 The train log loss is: 0.4344376234979843
For values of best alpha =  0.0001 The cross validation log loss is: 0.9946436642332478
For values of best alpha =  0.0001 The test log loss is: 0.9664572774164165
```

4.3.2.2. Testing model with best hyper parameters

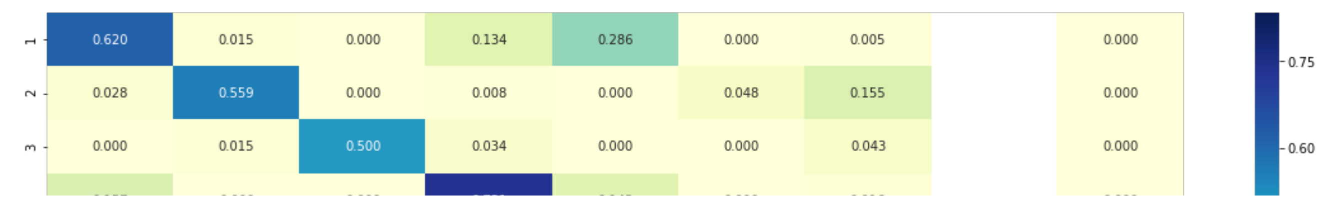
In [117]:

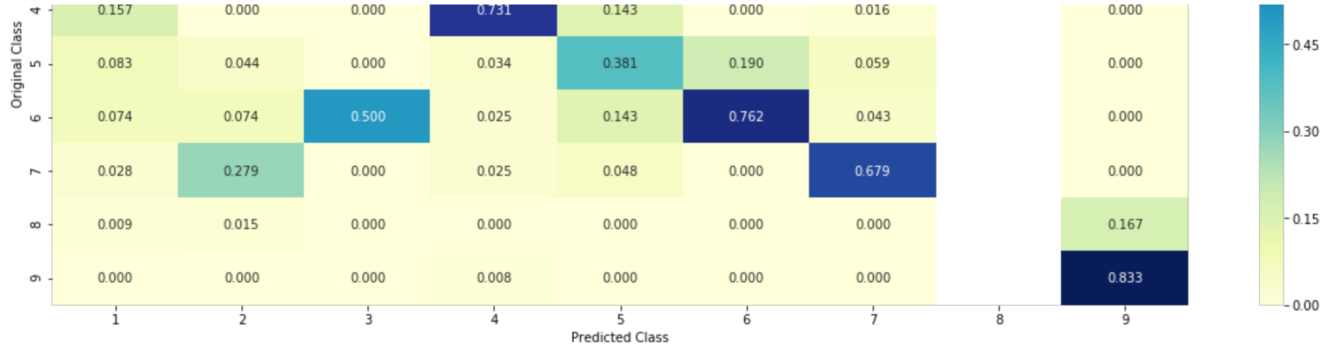
```python
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 0.9946436642332478
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```

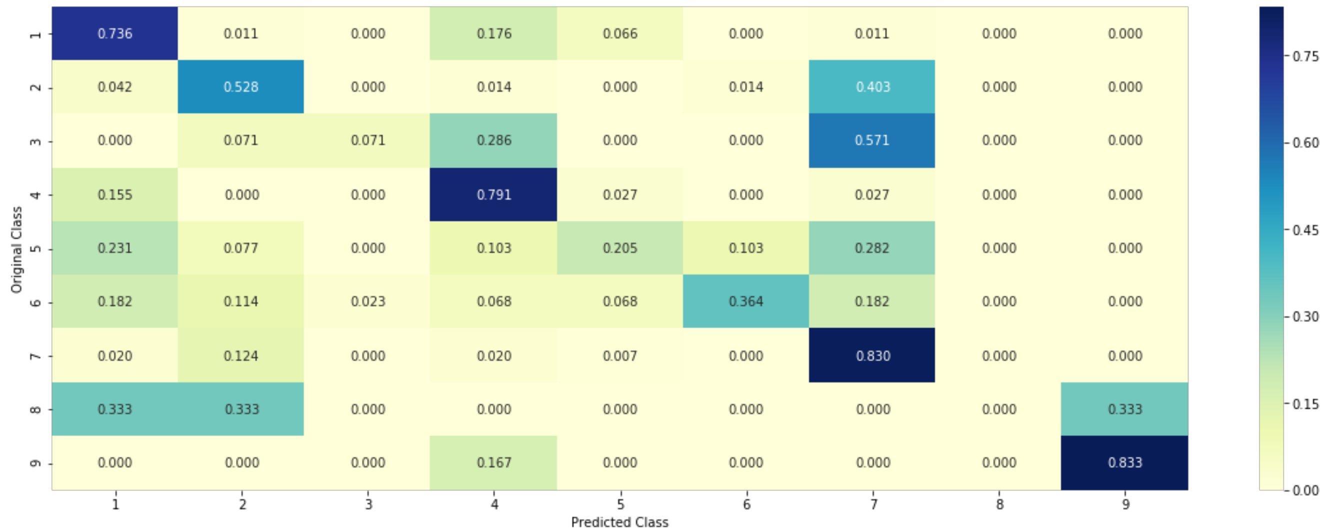



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



4.3.2.3. Feature Importance, Correctly Classified point

In [118]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1189 0.1114 0.0048 0.0051 0.0782 0.1226 0.5446 0.0071 0.0073]]
Actual Class : 7
--------------------------------------------------
333 Text feature [00] present in test data point [True]
396 Text feature [1251] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [119]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
```

```
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3832 0.0269 0.0406 0.4625 0.0283 0.0264 0.0147 0.0079 0.0097]]
Actual Class : 4
--------------------------------------------------
188 Text feature [100] present in test data point [True]
346 Text feature [042] present in test data point [True]
393 Text feature [01] present in test data point [True]
413 Text feature [045] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.4. Linear Support Vector Machines with Tfidf(max_features=1000)

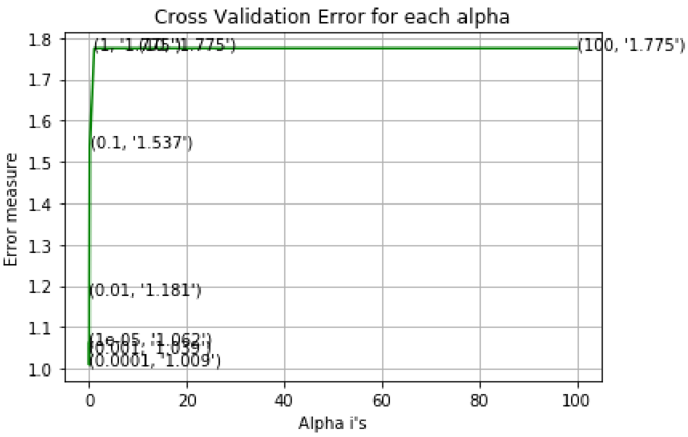
4.4.1. Hyper paramter tuning

In [120]:

```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state
=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
```

```
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', r
andom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.0622661476576638
for C = 0.0001
Log Loss : 1.0091748407500383
for C = 0.001
Log Loss : 1.0393478048942686
for C = 0.01
Log Loss : 1.1805235445994222
for C = 0.1
Log Loss : 1.5368712387342043
for C = 1
Log Loss : 1.7745975202172823
for C = 10
Log Loss : 1.774554126752811
for C = 100
Log Loss : 1.7745542302252162
```



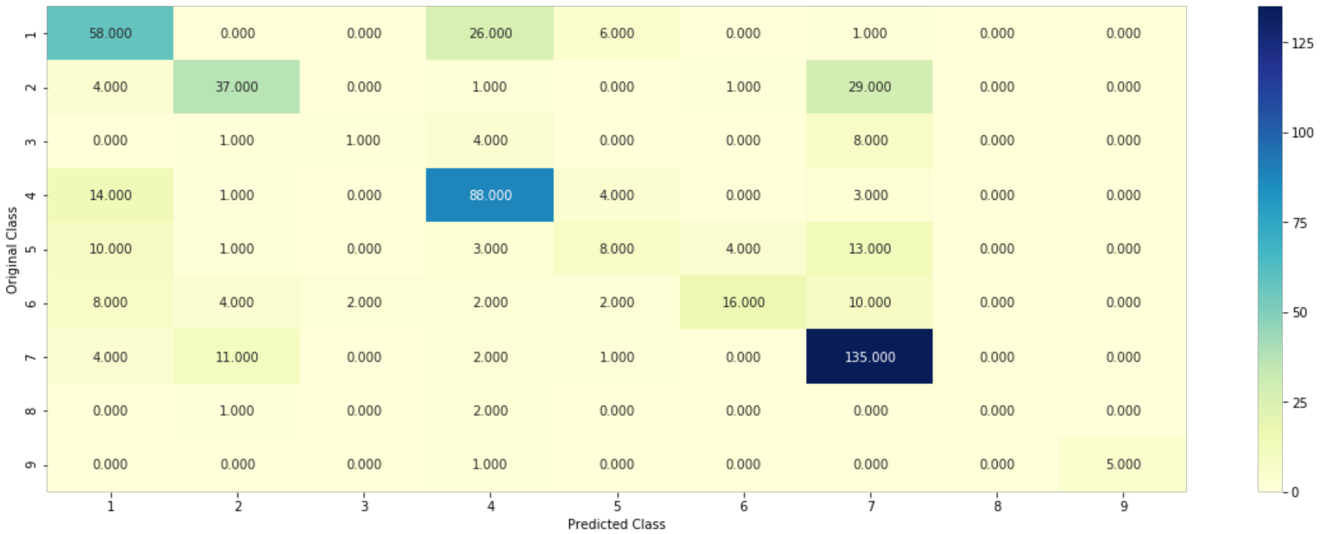

```
For values of best alpha =  0.0001 The train log loss is: 0.47411712222359886
For values of best alpha =  0.0001 The cross validation log loss is: 1.0091748407500383
For values of best alpha =  0.0001 The test log loss is: 0.9762055850897424
```

4.4.2. Testing model with best hyper parameters

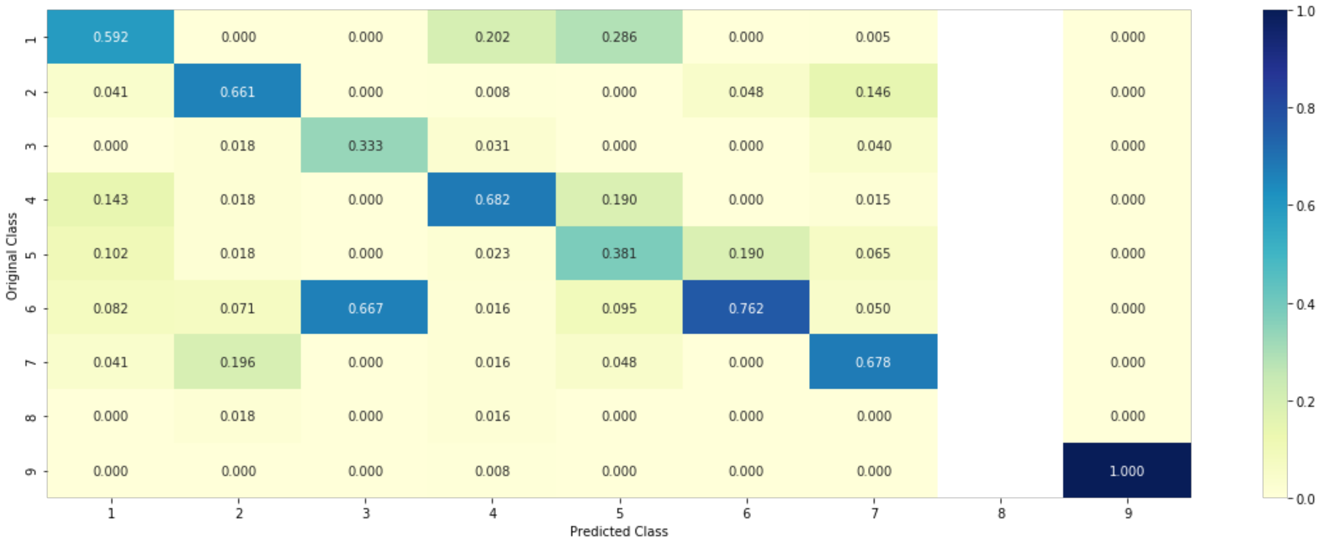
In [121]:

```python
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------


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

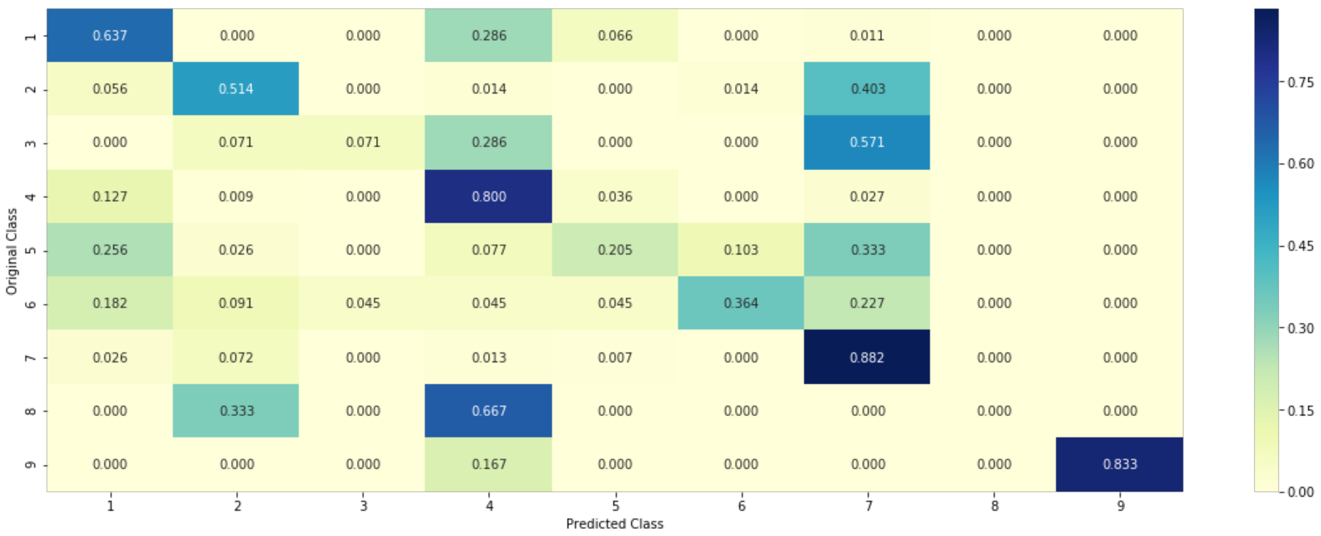
```
Log loss : 1.0091748407500383
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [122]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1181 0.0446 0.0023 0.0198 0.0614 0.2094 0.5349 0.0049 0.0047]]
Actual Class : 7
--------------------------------------------------
315 Text feature [00] present in test data point [True]
352 Text feature [121106] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [123]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
```

```
Predicted Class Probabilities: [[0.3558 0.0313 0.0275 0.4656 0.0385 0.0184 0.0481 0.0069 0.0079]]
Actual Class : 4
--------------------------------------------------
242 Text feature [042] present in test data point [True]
245 Text feature [100] present in test data point [True]
472 Text feature [045] present in test data point [True]
473 Text feature [05] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.5 Random Forest Classifier with Tfidf(max_features=1000)

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [124]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
```

```
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[i],cv_log_error_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 = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss
is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss
is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss
is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.261177671032227
for n_estimators = 100 and max depth =  10
Log Loss : 1.2784814672867353
for n_estimators = 200 and max depth =  5
Log Loss : 1.2518890869445678
for n_estimators = 200 and max depth =  10
Log Loss : 1.2720860371733866
for n_estimators = 500 and max depth =  5
Log Loss : 1.2426538763505428
for n_estimators = 500 and max depth =  10
Log Loss : 1.263740123856216
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2407897856704633
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2580550847427336
for n_estimators = 2000 and max depth =  5
Log Loss : 1.238578577381657
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2570674017637808
For values of best estimator =  2000 The train log loss is: 0.8617521615286582
For values of best estimator =  2000 The cross validation log loss is: 1.2385785773816573
For values of best estimator =  2000 The test log loss is: 1.1523738552249976
```

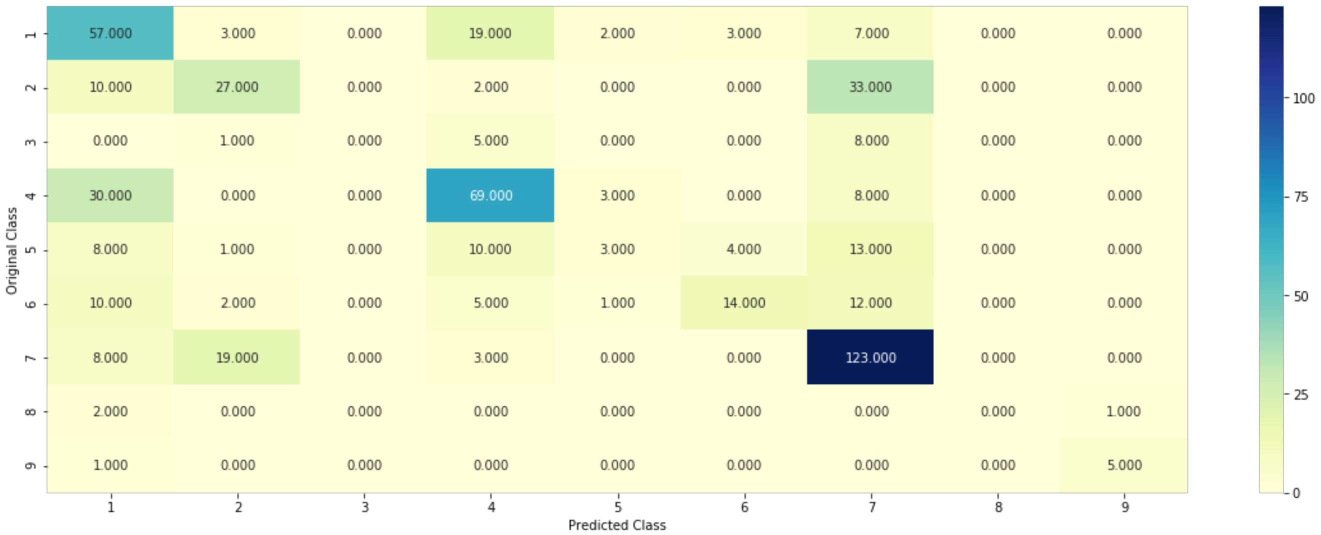
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [125]:

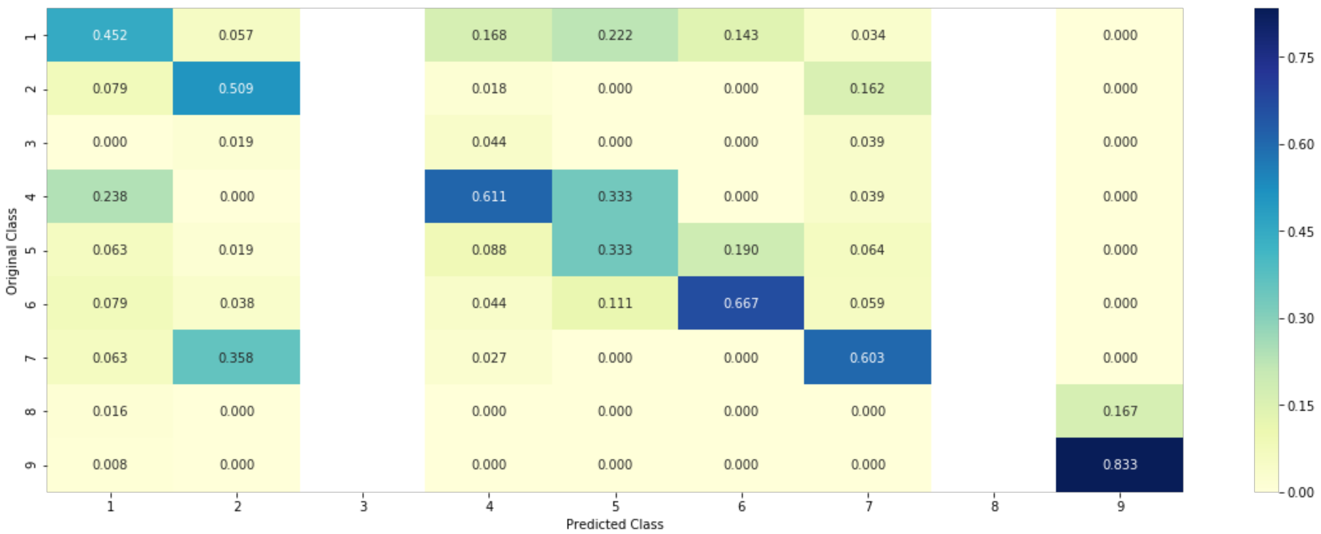
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).
```

```
# ----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# ----------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

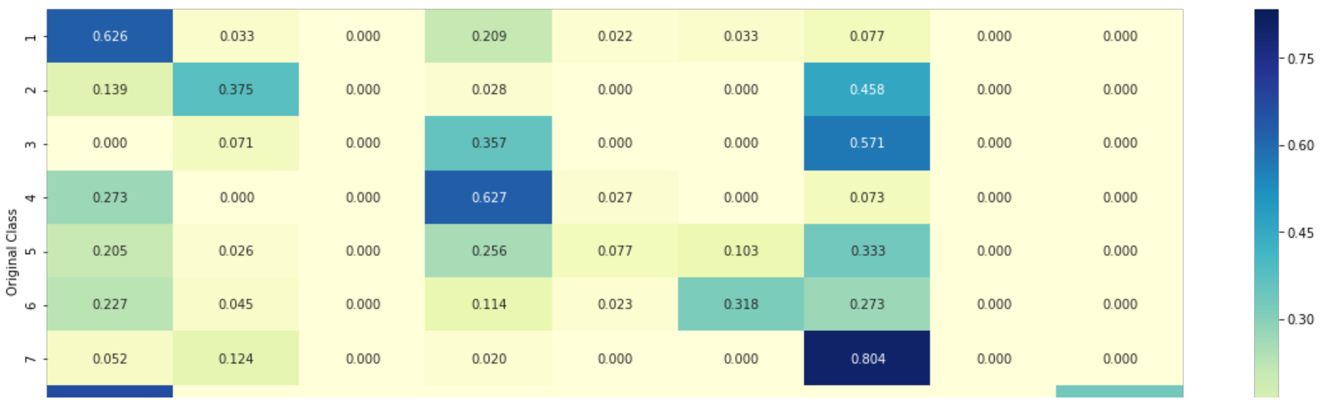
```
Log loss : 1.2385785773816573
Number of mis-classified points : 0.4398496240601504
-------------------- Confusion matrix --------------------
```

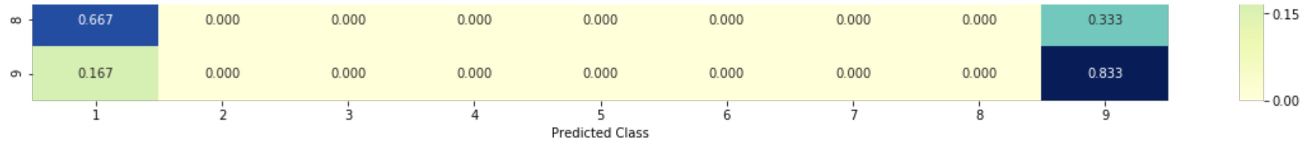



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [126]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0221 0.1937 0.0159 0.0179 0.0338 0.0394 0.6669 0.0086 0.0017]]
Actual Class : 7
--------------------------------------------------
45 Text feature [111] present in test data point [True]
91 Text feature [1000] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [127]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3043 0.0271 0.0156 0.4937 0.046  0.045  0.0563 0.0046 0.0074]]
Actuall Class : 4
--------------------------------------------------
69 Text feature [110] present in test data point [True]
98 Text feature [004] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [128]:

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

```
#
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[i],cv_log_error_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 = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max
_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
_train, predict_y, labels=clf.classes_, eps=1e-15))
```

```
_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:"
,log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_
test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.099106633787382
for n_estimators = 10 and max depth =  3
Log Loss : 1.6000217958056298
for n_estimators = 10 and max depth =  5
Log Loss : 1.509941280040656
for n_estimators = 10 and max depth =  10
Log Loss : 1.9766075314759766
for n_estimators = 50 and max depth =  2
Log Loss : 1.8057774113658889
for n_estimators = 50 and max depth =  3
Log Loss : 1.4738151802107495
for n_estimators = 50 and max depth =  5
Log Loss : 1.472096915498232
for n_estimators = 50 and max depth =  10
Log Loss : 1.8168697663300273
for n_estimators = 100 and max depth =  2
Log Loss : 1.6279915596878516
for n_estimators = 100 and max depth =  3
Log Loss : 1.4981095589111615
for n_estimators = 100 and max depth =  5
Log Loss : 1.2996399682875648
for n_estimators = 100 and max depth =  10
Log Loss : 1.755436757983853
for n_estimators = 200 and max depth =  2
Log Loss : 1.647419736881547
for n_estimators = 200 and max depth =  3
Log Loss : 1.5301738028397653
for n_estimators = 200 and max depth =  5
Log Loss : 1.344406697802887
for n_estimators = 200 and max depth =  10
Log Loss : 1.717853213147727
for n_estimators = 500 and max depth =  2
Log Loss : 1.7192397419784184
for n_estimators = 500 and max depth =  3
Log Loss : 1.5938065506668893
for n_estimators = 500 and max depth =  5
Log Loss : 1.368026905660471
for n_estimators = 500 and max depth =  10
Log Loss : 1.723188998815446
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6924510898849086
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6043583002746942
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3539353053401835
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6986329894540968
For values of best alpha =  100 The train log loss is: 0.056213064170682746
For values of best alpha =  100 The cross validation log loss is: 1.2996399682875648
For values of best alpha =  100 The test log loss is: 1.3357170626678425
```

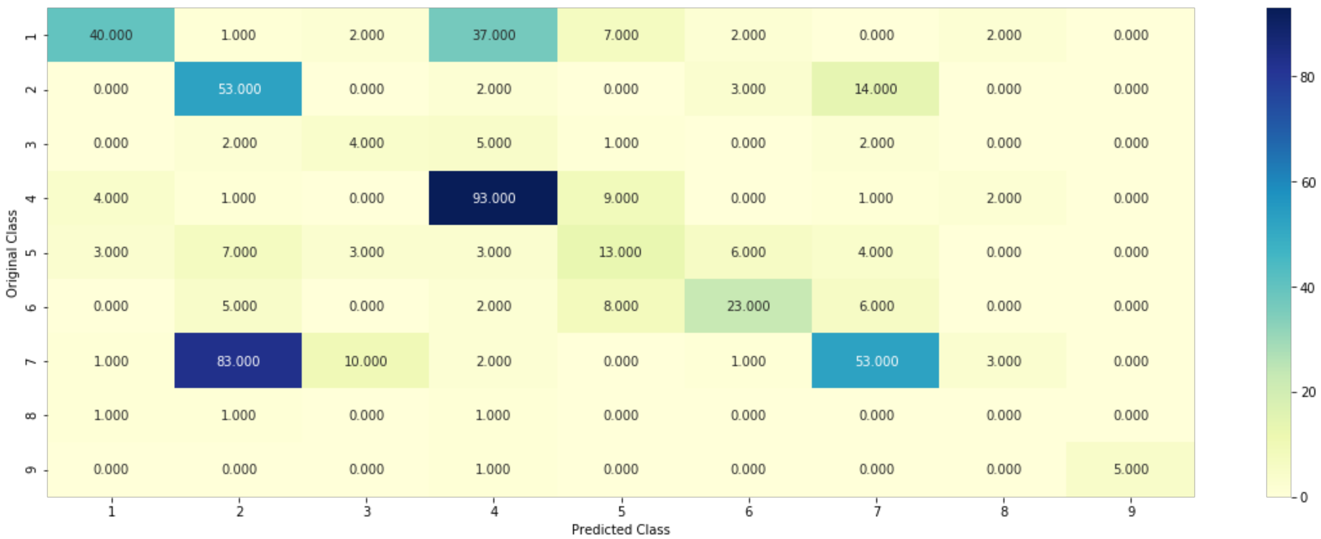
4.5.4. Testing model with best hyper parameters (Response Coding)

In [129]:

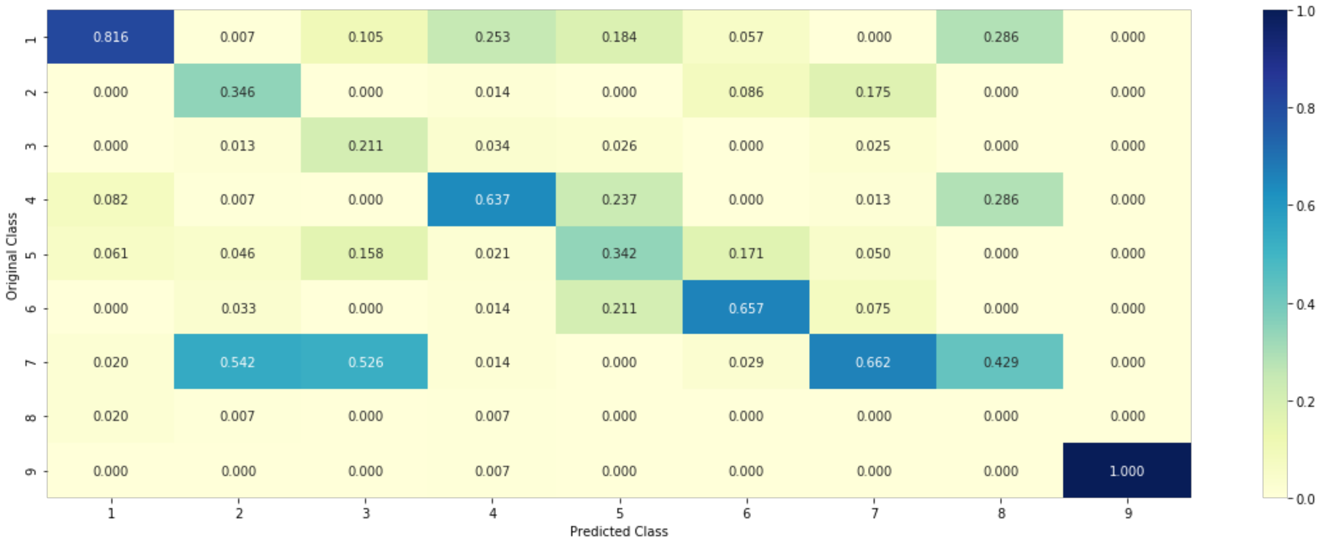
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
```

```
# some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)],
n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

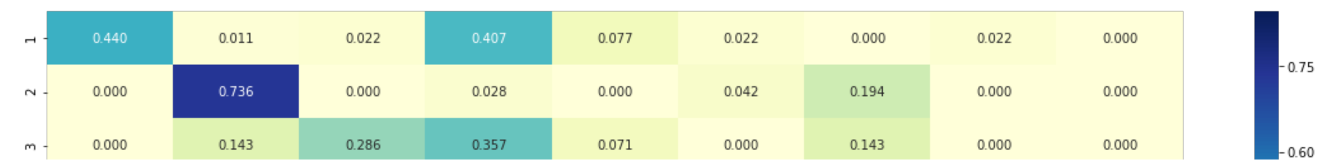
```
Log loss : 1.2996399682875648
Number of mis-classified points : 0.46616541353383456
-------------------- Confusion matrix --------------------
```

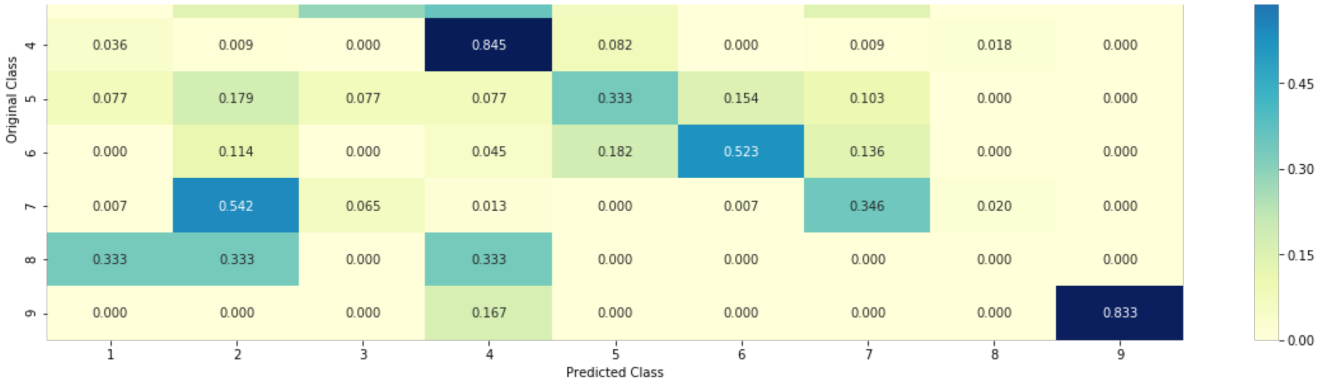



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [130]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max
_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0292 0.1852 0.1365 0.0281 0.0397 0.0879 0.3886 0.0816 0.0232]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
```

```
Gene is important feature
Text is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [131]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2182 0.0129 0.0985 0.546  0.0378 0.0412 0.0031 0.0225 0.0198]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.7 Stack the models with Tfidf(max_features=1000)

4.7.1 testing with hyper parameter tuning

In [132]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
```

```
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------


clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', class_weight='balanced', random_state=
0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.0001, penalty='l2', loss='hinge', class_weight='balanced', random_stat
e=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
```

```
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

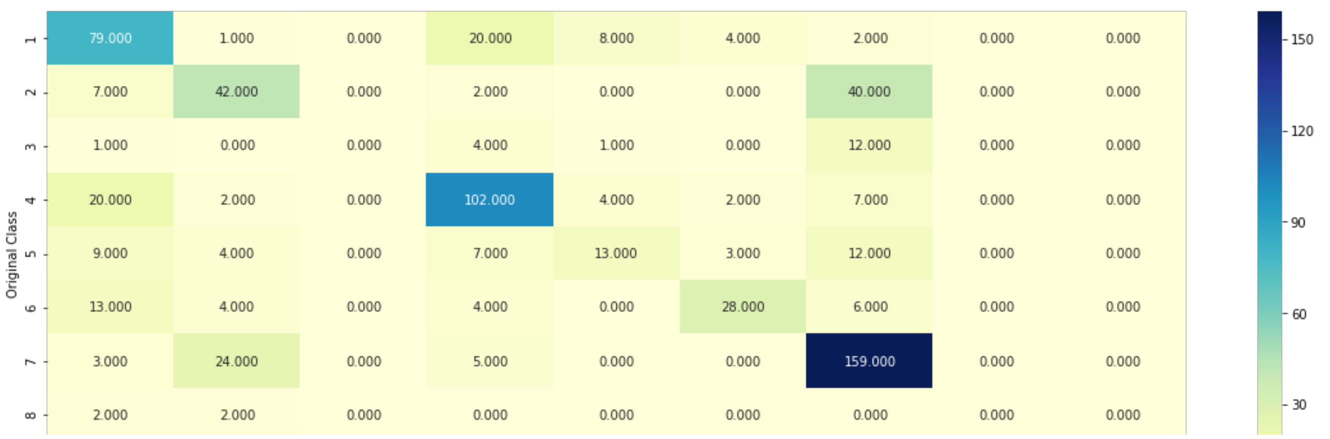
```
Logistic Regression :  Log Loss: 0.99
Support vector machines : Log Loss: 1.02
Naive Bayes : Log Loss: 1.23
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.172
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.986
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.381
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.092
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.319
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.801
```

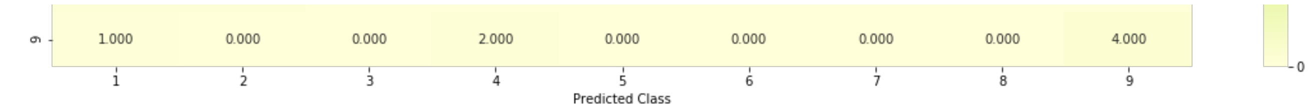
4.7.2 testing the model with the best hyper parameters

In [133]:

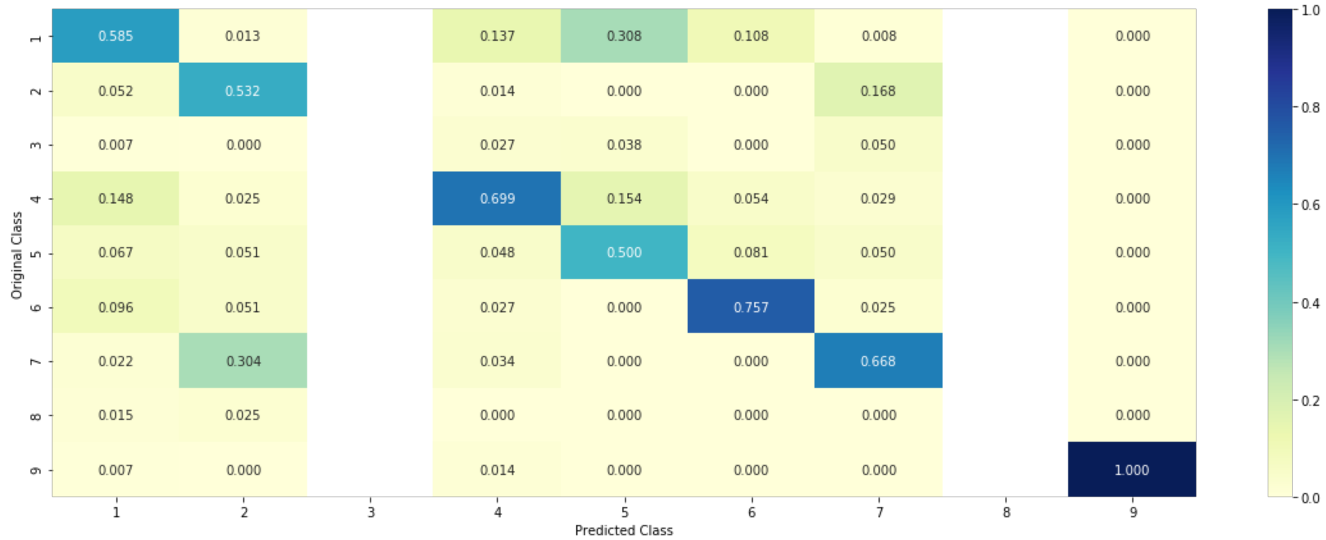
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_onehotCoding)-
test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.34466671430437557
Log loss (CV) on the stacking classifier : 1.0919171323006918
Log loss (test) on the stacking classifier : 1.0541829605386766
Number of missclassified point : 0.35789473684210527
-------------------- Confusion matrix --------------------
```

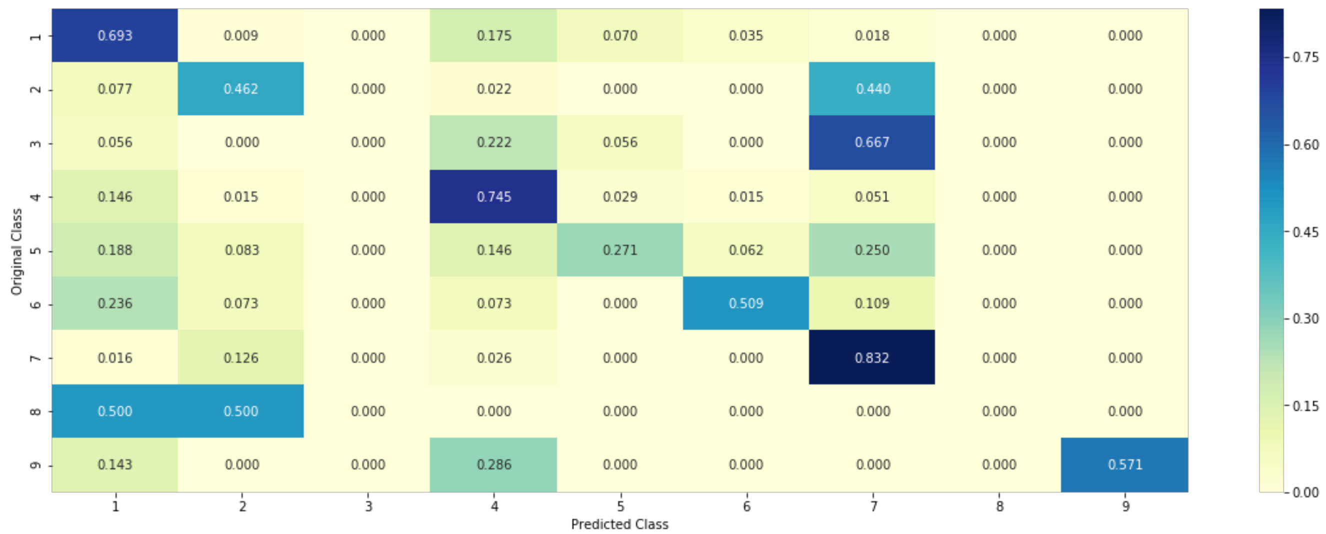





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



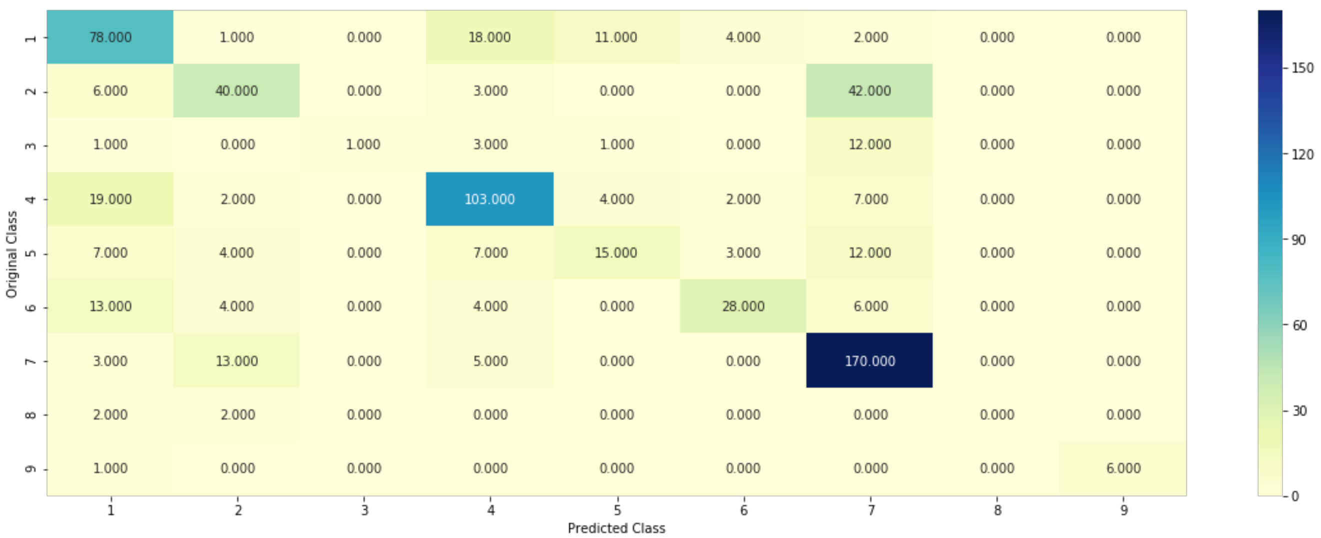
4.7.3 Maximum Voting classifier

In [134]:

```python
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y,
vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y,
vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)-
test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

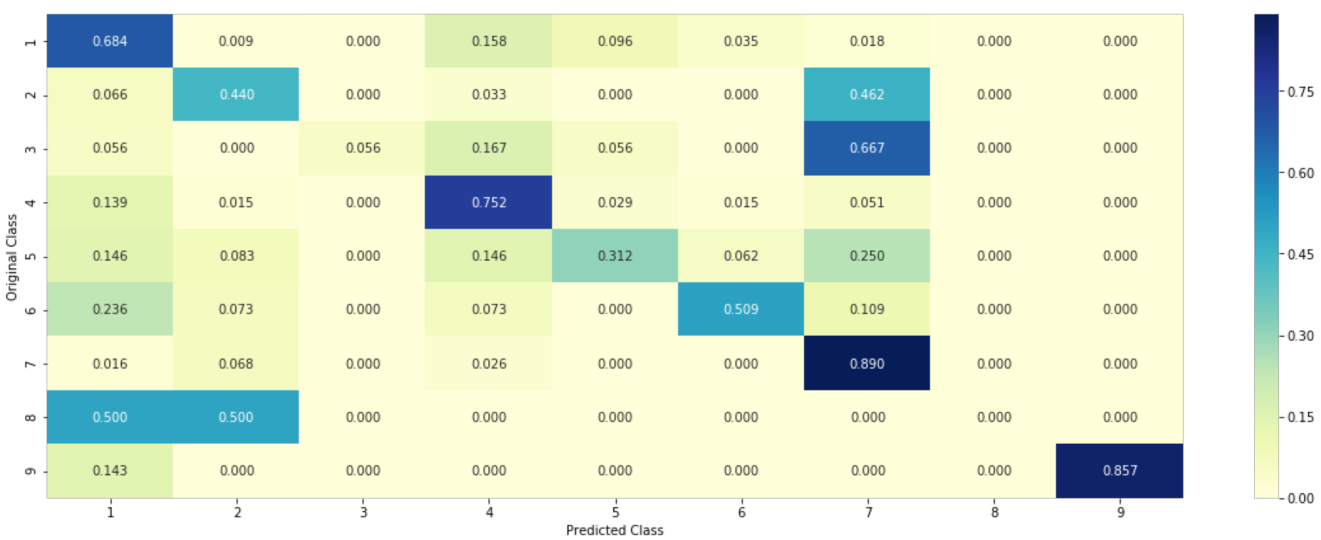
```
Log loss (train) on the VotingClassifier : 0.4798564692928693
Log loss (CV) on the VotingClassifier : 1.0224674411345462
```

```
Log loss (test) on the VotingClassifier : 1.009891390305365
Number of missclassified point : 0.3368421052631579
-------------------- Confusion matrix --------------------
```




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


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



Assignment-3.Logistic Regression with CountVectorizer

CountVectorizer(bigrams)

In [135]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
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 : 776082
```

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]:

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

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

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

In [0]:

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

In [139]:

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

```
Counter({3: 158465, 4: 90557, 6: 69650, 5: 63321, 7: 53978, 8: 35575, 9: 32887, 11: 24695, 12: 226
16, 10: 20747, 14: 14627, 13: 14378, 15: 12368, 18: 10725, 17: 9112, 16: 8531, 21: 5734, 19: 5689,
20: 5367, 22: 5233, 24: 4815, 27: 4722, 34: 4106, 26: 4021, 25: 3841, 32: 3747, 23: 3745, 28: 3217
, 30: 2787, 37: 2571, 29: 2387, 33: 2196, 35: 2180, 36: 2148, 31: 2127, 58: 2014, 38: 1723, 39: 15
13, 40: 1420, 42: 1383, 41: 1301, 45: 1182, 44: 1177, 61: 1172, 82: 1165, 43: 1120, 48: 985, 46: 9
82, 54: 880, 49: 877, 47: 857, 50: 828, 51: 824, 52: 762, 55: 735, 56: 722, 60: 717, 53: 700, 59:
691, 64: 653, 62: 648, 57: 639, 63: 624, 74: 581, 65: 579, 66: 558, 72: 509, 68: 509, 70: 492, 83:
490, 67: 486, 71: 470, 69: 466, 75: 429, 84: 425, 76: 387, 73: 373, 77: 370, 78: 369, 86: 351, 85:
350, 81: 326, 79: 323, 88: 315, 80: 312, 87: 305, 90: 297, 89: 284, 92: 266, 96: 263, 91: 263, 94:
260, 95: 254, 98: 246, 97: 244, 93: 243, 101: 239, 99: 239, 102: 227, 116: 224, 100: 216, 103: 211
, 108: 201, 105: 190, 112: 185, 109: 185, 106: 185, 107: 183, 104: 183, 111: 182, 122: 179, 117: 1
74, 114: 171, 119: 160, 110: 160, 115: 159, 126: 157, 113: 151, 129: 150, 125: 147, 121: 144, 118:
143, 120: 140, 135: 139, 130: 137, 132: 135, 124: 134, 123: 130, 127: 128, 133: 124, 137: 122, 141
: 121, 139: 120, 144: 118, 138: 118, 134: 117, 140: 116, 142: 113, 136: 111, 131: 111, 154: 106, 1
46: 106, 128: 106, 164: 105, 143: 103, 148: 97, 160: 95, 155: 95, 150: 94, 174: 91, 147: 90, 145:
90, 158: 89, 168: 88, 153: 87, 152: 87, 171: 85, 167: 84, 165: 82, 162: 82, 149: 81, 163: 78, 161:
78, 151: 78, 156: 77, 170: 75, 157: 75, 188: 74, 176: 74, 166: 72, 172: 71, 159: 70, 169: 69, 181:
68, 186: 66, 177: 66, 199: 65, 184: 65, 183: 65, 180: 64, 197: 63, 179: 63, 175: 63, 192: 62, 198:
61, 189: 61, 173: 61, 194: 60, 205: 58, 220: 57, 182: 55, 178: 55, 209: 53, 196: 53, 195: 53, 190:
53, 201: 52, 200: 52, 187: 52, 185: 52, 193: 51, 233: 50, 230: 50, 214: 50, 202: 50, 191: 50, 226:
49, 223: 49, 218: 49, 212: 47, 207: 46, 203: 45, 213: 44, 224: 43, 221: 43, 210: 43, 206: 43, 204:
43, 219: 42, 216: 42, 239: 41, 236: 41, 246: 40, 208: 40, 260: 39, 242: 39, 215: 39, 211: 39, 264:
38, 263: 38, 234: 38, 225: 38, 222: 38, 302: 37, 240: 37, 229: 37, 244: 36, 237: 36, 231: 36, 290:
35, 266: 35, 265: 35, 252: 35, 235: 35, 292: 34, 289: 34, 273: 34, 251: 34, 250: 34, 227: 34, 272:
33, 271: 33, 245: 33, 232: 33, 285: 32, 281: 32, 278: 32, 274: 32, 248: 32, 247: 32, 243: 32, 238:
32, 305: 31, 258: 31, 256: 31, 255: 31, 228: 31, 217: 31, 356: 30, 288: 30, 353: 29, 293: 29, 291:
29, 284: 29, 276: 29, 275: 29, 268: 29, 267: 29, 321: 28, 300: 28, 257: 28, 253: 28, 295: 27, 347:
26, 322: 26, 287: 26, 283: 26, 280: 26, 262: 26, 360: 25, 328: 25, 307: 25, 261: 25, 259: 25, 241:
25, 361: 24, 342: 24, 332: 24, 330: 24, 304: 24, 282: 24, 249: 24, 339: 23, 318: 23, 312: 23, 297:
23, 277: 23, 254: 23, 409: 22, 329: 22, 324: 22, 320: 22, 303: 22, 294: 22, 363: 21, 346: 21, 337:
21, 336: 21, 325: 21, 298: 21, 296: 21, 286: 21, 395: 20, 366: 20, 364: 20, 351: 20, 335: 20, 331:
20, 327: 20, 316: 20, 314: 20, 306: 20, 279: 20, 386: 19, 381: 19, 368: 19, 326: 19, 317: 19, 311:
19, 309: 19, 269: 19, 470: 18, 392: 18, 378: 18, 375: 18, 370: 18, 344: 18, 319: 18, 308: 18, 299:
18, 460: 17, 416: 17, 385: 17, 383: 17, 358: 17, 348: 17, 345: 17, 341: 17, 323: 17, 315: 17, 310:
17, 270: 17, 456: 16, 429: 16, 401: 16, 393: 16, 379: 16, 376: 16, 352: 16, 349: 16, 334: 16, 333:
16, 397: 15, 389: 15, 387: 15, 380: 15, 365: 15, 469: 14, 450: 14, 445: 14, 437: 14, 432: 14, 413:
14, 402: 14, 399: 14, 398: 14, 396: 14, 382: 14, 357: 14, 355: 14, 354: 14, 343: 14, 338: 14, 748:
13, 523: 13, 513: 13, 508: 13, 457: 13, 447: 13, 444: 13, 425: 13, 421: 13, 408: 13, 406: 13, 390:
13, 372: 13, 362: 13, 350: 13, 313: 13, 301: 13, 625: 12, 580: 12, 519: 12, 515: 12, 495: 12, 494:
12, 482: 12, 471: 12, 468: 12, 451: 12, 438: 12, 414: 12, 407: 12, 405: 12, 384: 12, 374: 12, 369:
12, 709: 11, 573: 11, 558: 11, 557: 11, 537: 11, 536: 11, 514: 11, 492: 11, 489: 11, 485: 11, 479:
11, 473: 11, 466: 11, 454: 11, 453: 11, 424: 11, 411: 11, 394: 11, 367: 11, 359: 11, 340: 11, 1068
: 10, 847: 10, 650: 10, 583: 10, 571: 10, 539: 10, 505: 10, 503: 10, 501: 10, 476: 10, 472: 10, 45
9: 10, 458: 10, 452: 10, 449: 10, 448: 10, 442: 10, 439: 10, 433: 10, 428: 10, 418: 10, 412: 10, 4
03: 10, 400: 10, 391: 10, 371: 10, 683: 9, 644: 9, 623: 9, 621: 9, 613: 9, 601: 9, 599: 9, 579: 9,
577: 9, 560: 9, 549: 9, 543: 9, 533: 9, 527: 9, 509: 9, 504: 9, 498: 9, 496: 9, 487: 9, 481: 9,
465: 9, 464: 9, 435: 9, 419: 9, 417: 9, 415: 9, 410: 9, 404: 9, 388: 9, 951: 8, 903: 8, 744: 8,
716: 8, 701: 8, 689: 8, 688: 8, 676: 8, 672: 8, 668: 8, 667: 8, 658: 8, 641: 8, 640: 8, 591: 8,
587: 8, 565: 8, 563: 8, 559: 8, 553: 8, 552: 8, 542: 8, 532: 8, 528: 8, 525: 8, 522: 8, 517: 8,
499: 8, 491: 8, 467: 8, 461: 8, 443: 8, 441: 8, 436: 8, 434: 8, 431: 8, 430: 8, 427: 8, 426: 8,
422: 8, 420: 8, 993: 7, 986: 7, 922: 7, 846: 7, 784: 7, 769: 7, 696: 7, 686: 7, 682: 7, 681: 7,
670: 7, 666: 7, 656: 7, 652: 7, 638: 7, 628: 7, 618: 7, 602: 7, 594: 7, 593: 7, 581: 7, 569: 7,
566: 7, 564: 7, 555: 7, 541: 7, 530: 7, 529: 7, 524: 7, 521: 7, 518: 7, 512: 7, 511: 7, 507: 7,
506: 7, 500: 7, 490: 7, 486: 7, 483: 7, 477: 7, 462: 7, 455: 7, 423: 7, 373: 7, 1553: 6, 1429: 6,
1230: 6, 1146: 6, 1099: 6, 1013: 6, 1001: 6, 982: 6, 962: 6, 888: 6, 885: 6, 876: 6, 863: 6, 831: 6
, 828: 6, 825: 6, 808: 6, 804: 6, 798: 6, 781: 6, 762: 6, 747: 6, 745: 6, 730: 6, 728: 6, 719: 6,
713: 6, 711: 6, 708: 6, 707: 6, 705: 6, 703: 6, 699: 6, 673: 6, 649: 6, 647: 6, 639: 6, 637: 6,
635: 6, 630: 6, 615: 6, 614: 6, 612: 6, 606: 6, 592: 6, 585: 6, 584: 6, 576: 6, 574: 6, 572: 6,
570: 6, 551: 6, 546: 6, 538: 6, 534: 6, 526: 6, 520: 6, 516: 6, 502: 6, 497: 6, 493: 6, 488: 6,
478: 6, 474: 6, 440: 6, 377: 6, 1685: 5, 1460: 5, 1350: 5, 1312: 5, 1270: 5, 1213: 5, 1112: 5, 1105
: 5, 1103: 5, 1062: 5, 1027: 5, 1008: 5, 970: 5, 964: 5, 957: 5, 948: 5, 938: 5, 932: 5, 930: 5, 90
5: 5, 897: 5, 890: 5, 871: 5, 866: 5, 854: 5, 830: 5, 823: 5, 806: 5, 805: 5, 803: 5, 800: 5, 795:
5, 788: 5, 780: 5, 776: 5, 760: 5, 757: 5, 754: 5, 753: 5, 750: 5, 742: 5, 741: 5, 739: 5, 737: 5,
736: 5, 733: 5, 710: 5, 702: 5, 698: 5, 677: 5, 675: 5, 669: 5, 657: 5, 655: 5, 654: 5, 653: 5,
631: 5, 620: 5, 619: 5, 616: 5, 608: 5, 607: 5, 605: 5, 598: 5, 597: 5, 596: 5, 588: 5, 586: 5,
578: 5, 575: 5, 567: 5, 561: 5, 556: 5, 554: 5, 540: 5, 535: 5, 484: 5, 475: 5, 2090: 4, 1863: 4,
1821: 4, 1761: 4, 1742: 4, 1622: 4, 1601: 4, 1586: 4, 1571: 4, 1548: 4, 1528: 4, 1493: 4, 1462: 4,
1407: 4, 1380: 4, 1366: 4, 1332: 4, 1320: 4, 1273: 4, 1256: 4, 1246: 4, 1238: 4, 1226: 4, 1196: 4,
1189: 4, 1174: 4, 1166: 4, 1164: 4, 1158: 4, 1141: 4, 1136: 4, 1130: 4, 1128: 4, 1094: 4, 1058: 4,
1053: 4, 1044: 4, 1035: 4, 1030: 4, 1021: 4, 1014: 4, 1007: 4, 991: 4, 990: 4, 989: 4, 985: 4, 983:
4, 976: 4, 971: 4, 968: 4, 953: 4, 941: 4, 933: 4, 926: 4, 921: 4, 918: 4, 917: 4, 915: 4, 911: 4,
906: 4, 900: 4, 881: 4, 874: 4, 870: 4, 858: 4, 841: 4, 834: 4, 820: 4, 819: 4, 817: 4, 816: 4,
```

815: 4, 813: 4, 810: 4, 797: 4, 796: 4, 791: 4, 785: 4, 783: 4, 777: 4, 772: 4, 768: 4, 765: 4,
761: 4, 759: 4, 758: 4, 752: 4, 751: 4, 746: 4, 738: 4, 735: 4, 732: 4, 727: 4, 723: 4, 722: 4,
720: 4, 715: 4, 704: 4, 693: 4, 687: 4, 685: 4, 684: 4, 671: 4, 664: 4, 662: 4, 660: 4, 659: 4,
651: 4, 648: 4, 646: 4, 643: 4, 632: 4, 626: 4, 611: 4, 609: 4, 604: 4, 568: 4, 548: 4, 545: 4,
531: 4, 510: 4, 480: 4, 463: 4, 446: 4, 3192: 3, 2722: 3, 2711: 3, 2587: 3, 2542: 3, 2530: 3, 2432:
3, 2300: 3, 2279: 3, 2200: 3, 2151: 3, 2087: 3, 2046: 3, 2041: 3, 2023: 3, 1984: 3, 1968: 3, 1829:
3, 1824: 3, 1785: 3, 1722: 3, 1678: 3, 1640: 3, 1630: 3, 1619: 3, 1614: 3, 1613: 3, 1608: 3, 1597:
3, 1589: 3, 1543: 3, 1526: 3, 1498: 3, 1449: 3, 1428: 3, 1427: 3, 1400: 3, 1383: 3, 1382: 3, 1367:
3, 1357: 3, 1355: 3, 1339: 3, 1334: 3, 1326: 3, 1316: 3, 1304: 3, 1299: 3, 1280: 3, 1278: 3, 1252:
3, 1249: 3, 1244: 3, 1240: 3, 1228: 3, 1222: 3, 1218: 3, 1217: 3, 1215: 3, 1211: 3, 1210: 3, 1209:
3, 1204: 3, 1200: 3, 1199: 3, 1186: 3, 1185: 3, 1160: 3, 1155: 3, 1150: 3, 1143: 3, 1133: 3, 1126:
3, 1124: 3, 1123: 3, 1110: 3, 1106: 3, 1102: 3, 1098: 3, 1078: 3, 1077: 3, 1072: 3, 1070: 3, 1069:
3, 1064: 3, 1052: 3, 1049: 3, 1047: 3, 1039: 3, 1019: 3, 1018: 3, 998: 3, 996: 3, 995: 3, 994: 3, 9
92: 3, 981: 3, 978: 3, 974: 3, 973: 3, 969: 3, 967: 3, 966: 3, 958: 3, 955: 3, 943: 3, 942: 3,
940: 3, 937: 3, 935: 3, 929: 3, 924: 3, 908: 3, 907: 3, 904: 3, 898: 3, 893: 3, 889: 3, 887: 3,
883: 3, 882: 3, 880: 3, 875: 3, 868: 3, 867: 3, 862: 3, 860: 3, 856: 3, 853: 3, 851: 3, 845: 3,
840: 3, 839: 3, 838: 3, 835: 3, 833: 3, 832: 3, 822: 3, 821: 3, 809: 3, 793: 3, 792: 3, 790: 3,
789: 3, 787: 3, 779: 3, 778: 3, 775: 3, 774: 3, 773: 3, 764: 3, 763: 3, 756: 3, 755: 3, 743: 3,
734: 3, 731: 3, 726: 3, 721: 3, 718: 3, 712: 3, 706: 3, 700: 3, 697: 3, 695: 3, 694: 3, 692: 3,
691: 3, 680: 3, 663: 3, 642: 3, 627: 3, 624: 3, 622: 3, 617: 3, 610: 3, 595: 3, 590: 3, 547: 3,
544: 3, 13234: 2, 9981: 2, 8462: 2, 6569: 2, 6405: 2, 5792: 2, 5741: 2, 5496: 2, 5197: 2, 5140: 2,
4998: 2, 4939: 2, 4882: 2, 4868: 2, 4573: 2, 4442: 2, 4173: 2, 4161: 2, 4109: 2, 4040: 2, 4004: 2,
3951: 2, 3950: 2, 3933: 2, 3790: 2, 3785: 2, 3581: 2, 3574: 2, 3562: 2, 3546: 2, 3467: 2, 3445: 2,
3429: 2, 3407: 2, 3392: 2, 3390: 2, 3344: 2, 3306: 2, 3295: 2, 3249: 2, 3235: 2, 3227: 2, 3205: 2,
3183: 2, 3153: 2, 3058: 2, 3045: 2, 3044: 2, 3043: 2, 3042: 2, 3035: 2, 3032: 2, 3005: 2, 2801: 2,
2791: 2, 2758: 2, 2723: 2, 2703: 2, 2685: 2, 2681: 2, 2676: 2, 2675: 2, 2668: 2, 2664: 2, 2660: 2,
2568: 2, 2567: 2, 2548: 2, 2538: 2, 2527: 2, 2524: 2, 2521: 2, 2502: 2, 2457: 2, 2436: 2, 2426: 2,
2423: 2, 2422: 2, 2416: 2, 2412: 2, 2402: 2, 2398: 2, 2395: 2, 2376: 2, 2371: 2, 2368: 2, 2350: 2,
2341: 2, 2339: 2, 2312: 2, 2308: 2, 2296: 2, 2288: 2, 2282: 2, 2269: 2, 2263: 2, 2246: 2, 2228: 2,
2207: 2, 2205: 2, 2188: 2, 2160: 2, 2153: 2, 2146: 2, 2138: 2, 2119: 2, 2115: 2, 2078: 2, 2070: 2,
2069: 2, 2067: 2, 2057: 2, 2053: 2, 2052: 2, 2042: 2, 2040: 2, 2025: 2, 2024: 2, 2013: 2, 2002: 2,
2000: 2, 1977: 2, 1959: 2, 1954: 2, 1952: 2, 1944: 2, 1936: 2, 1922: 2, 1913: 2, 1911: 2, 1910: 2,
1908: 2, 1900: 2, 1896: 2, 1890: 2, 1889: 2, 1886: 2, 1884: 2, 1878: 2, 1877: 2, 1876: 2, 1853: 2,
1845: 2, 1842: 2, 1833: 2, 1813: 2, 1808: 2, 1805: 2, 1804: 2, 1802: 2, 1797: 2, 1781: 2, 1777: 2,
1762: 2, 1758: 2, 1740: 2, 1736: 2, 1725: 2, 1718: 2, 1712: 2, 1707: 2, 1706: 2, 1703: 2, 1695: 2,
1691: 2, 1687: 2, 1681: 2, 1679: 2, 1675: 2, 1665: 2, 1662: 2, 1661: 2, 1655: 2, 1650: 2, 1642: 2,
1636: 2, 1635: 2, 1632: 2, 1629: 2, 1606: 2, 1603: 2, 1600: 2, 1592: 2, 1590: 2, 1588: 2, 1585: 2,
1584: 2, 1582: 2, 1581: 2, 1570: 2, 1569: 2, 1567: 2, 1563: 2, 1562: 2, 1560: 2, 1554: 2, 1540: 2,
1537: 2, 1535: 2, 1534: 2, 1532: 2, 1525: 2, 1515: 2, 1511: 2, 1501: 2, 1487: 2, 1485: 2, 1484: 2,
1481: 2, 1476: 2, 1469: 2, 1466: 2, 1456: 2, 1446: 2, 1444: 2, 1443: 2, 1442: 2, 1437: 2, 1433: 2,
1432: 2, 1424: 2, 1415: 2, 1411: 2, 1409: 2, 1408: 2, 1403: 2, 1401: 2, 1396: 2, 1394: 2, 1393: 2,
1391: 2, 1387: 2, 1386: 2, 1375: 2, 1369: 2, 1365: 2, 1363: 2, 1362: 2, 1360: 2, 1352: 2, 1351: 2,
1347: 2, 1346: 2, 1342: 2, 1340: 2, 1337: 2, 1333: 2, 1331: 2, 1328: 2, 1324: 2, 1317: 2, 1315: 2,
1314: 2, 1309: 2, 1308: 2, 1305: 2, 1302: 2, 1293: 2, 1291: 2, 1290: 2, 1287: 2, 1286: 2, 1283: 2,
1282: 2, 1275: 2, 1268: 2, 1263: 2, 1259: 2, 1258: 2, 1243: 2, 1241: 2, 1235: 2, 1233: 2, 1229: 2,
1220: 2, 1219: 2, 1214: 2, 1207: 2, 1197: 2, 1193: 2, 1192: 2, 1184: 2, 1183: 2, 1182: 2, 1178: 2,
1177: 2, 1176: 2, 1173: 2, 1172: 2, 1169: 2, 1165: 2, 1162: 2, 1156: 2, 1152: 2, 1151: 2, 1149: 2,
1147: 2, 1139: 2, 1138: 2, 1135: 2, 1129: 2, 1127: 2, 1120: 2, 1118: 2, 1115: 2, 1104: 2, 1100: 2,
1096: 2, 1091: 2, 1089: 2, 1086: 2, 1085: 2, 1084: 2, 1083: 2, 1082: 2, 1081: 2, 1079: 2, 1066: 2,
1063: 2, 1059: 2, 1057: 2, 1054: 2, 1048: 2, 1046: 2, 1043: 2, 1040: 2, 1038: 2, 1037: 2, 1034: 2,
1033: 2, 1025: 2, 1017: 2, 1016: 2, 1012: 2, 1002: 2, 1000: 2, 997: 2, 988: 2, 987: 2, 980: 2, 977:
2, 975: 2, 965: 2, 961: 2, 959: 2, 952: 2, 946: 2, 945: 2, 944: 2, 939: 2, 934: 2, 928: 2, 927: 2,
925: 2, 923: 2, 920: 2, 916: 2, 913: 2, 910: 2, 899: 2, 892: 2, 891: 2, 879: 2, 878: 2, 877: 2,
873: 2, 872: 2, 864: 2, 861: 2, 859: 2, 855: 2, 849: 2, 848: 2, 844: 2, 836: 2, 829: 2, 826: 2,
824: 2, 814: 2, 811: 2, 807: 2, 802: 2, 801: 2, 786: 2, 767: 2, 766: 2, 740: 2, 729: 2, 725: 2,
724: 2, 690: 2, 679: 2, 678: 2, 665: 2, 661: 2, 645: 2, 636: 2, 634: 2, 633: 2, 629: 2, 603: 2,
600: 2, 589: 2, 582: 2, 562: 2, 550: 2, 150744: 1, 117165: 1, 79006: 1, 67747: 1, 67312: 1, 64925:
1, 64738: 1, 64179: 1, 63037: 1, 62859: 1, 55385: 1, 54726: 1, 48980: 1, 48428: 1, 47154: 1, 47003
: 1, 45090: 1, 43353: 1, 43045: 1, 42395: 1, 42155: 1, 41999: 1, 40964: 1, 40944: 1, 38918: 1, 387
08: 1, 38075: 1, 37393: 1, 36993: 1, 36329: 1, 35939: 1, 35281: 1, 34538: 1, 34466: 1, 33525: 1, 3
3303: 1, 32075: 1, 31941: 1, 29644: 1, 28326: 1, 26333: 1, 26037: 1, 26008: 1, 25910: 1, 25755: 1,
25679: 1, 25623: 1, 25310: 1, 24957: 1, 24868: 1, 24554: 1, 24454: 1, 24319: 1, 23891: 1, 23659: 1
, 22490: 1, 22382: 1, 22212: 1, 22175: 1, 21673: 1, 21471: 1, 21216: 1, 20815: 1, 20518: 1, 20390:
1, 20063: 1, 19836: 1, 19699: 1, 19529: 1, 19368: 1, 19307: 1, 19175: 1, 18902: 1, 18841: 1, 18701
: 1, 18699: 1, 18275: 1, 18208: 1, 18191: 1, 18115: 1, 18113: 1, 18076: 1, 17939: 1, 17876: 1, 178
54: 1, 17822: 1, 17772: 1, 17755: 1, 17594: 1, 17592: 1, 17507: 1, 17398: 1, 17165: 1, 17059: 1, 1
7054: 1, 16814: 1, 16688: 1, 16590: 1, 16583: 1, 16568: 1, 16161: 1, 16081: 1, 16061: 1, 15997: 1,
15980: 1, 15787: 1, 15748: 1, 15690: 1, 15561: 1, 15559: 1, 15432: 1, 15369: 1, 15082: 1, 15036: 1
, 14849: 1, 14807: 1, 14707: 1, 14668: 1, 14580: 1, 14549: 1, 14513: 1, 14388: 1, 14374: 1, 14147:
1, 14030: 1, 13983: 1, 13947: 1, 13692: 1, 13476: 1, 13324: 1, 13301: 1, 13240: 1, 13208: 1, 13123
: 1, 13103: 1, 13050: 1, 13031: 1, 12981: 1, 12907: 1, 12794: 1, 12787: 1, 12561: 1, 12528: 1, 125
20: 1, 12513: 1, 12435: 1, 12412: 1, 12382: 1, 12380: 1, 12375: 1, 12352: 1, 12312: 1, 12291: 1, 1
2284: 1, 12249: 1, 12248: 1, 12217: 1, 12191: 1, 12180: 1, 12167: 1, 12164: 1, 12154: 1, 12142: 1,
12130: 1, 12087: 1, 12010: 1, 12004: 1, 11919: 1, 11810: 1, 11773: 1, 11725: 1, 11698: 1, 11692: 1
, 11677: 1, 11482: 1, 11394: 1, 11380: 1, 11371: 1, 11242: 1, 11235: 1, 11074: 1, 10957: 1, 10934:
1, 10906: 1, 10894: 1, 10870: 1, 10740: 1, 10720: 1, 10645: 1, 10575: 1, 10539: 1, 10529: 1, 10511
: 1, 10417: 1, 10404: 1, 10374: 1, 10283: 1, 10252: 1, 10246: 1, 10243: 1, 10164: 1, 10094: 1, 100

40: 1, 10017: 1, 9975: 1, 9946: 1, 9912: 1, 9886: 1, 9820: 1, 9813: 1, 9800: 1, 9770: 1, 9766: 1, 9
650: 1, 9600: 1, 9576: 1, 9572: 1, 9542: 1, 9442: 1, 9383: 1, 9374: 1, 9367: 1, 9313: 1, 9310: 1, 9
288: 1, 9281: 1, 9262: 1, 9252: 1, 9248: 1, 9215: 1, 9214: 1, 9169: 1, 9104: 1, 9076: 1, 9073: 1, 9
062: 1, 9052: 1, 9003: 1, 8977: 1, 8970: 1, 8961: 1, 8955: 1, 8818: 1, 8754: 1, 8746: 1, 8730: 1, 8
729: 1, 8686: 1, 8678: 1, 8640: 1, 8629: 1, 8600: 1, 8508: 1, 8498: 1, 8359: 1, 8318: 1, 8302: 1, 8
299: 1, 8269: 1, 8198: 1, 8168: 1, 8162: 1, 8147: 1, 8137: 1, 8135: 1, 8133: 1, 8129: 1, 8109: 1, 8
069: 1, 8052: 1, 8048: 1, 8031: 1, 8009: 1, 7991: 1, 7962: 1, 7952: 1, 7936: 1, 7930: 1, 7871: 1, 7
860: 1, 7841: 1, 7835: 1, 7822: 1, 7813: 1, 7759: 1, 7758: 1, 7743: 1, 7716: 1, 7710: 1, 7700: 1, 7
698: 1, 7687: 1, 7660: 1, 7656: 1, 7622: 1, 7619: 1, 7597: 1, 7581: 1, 7569: 1, 7550: 1, 7533: 1, 7
500: 1, 7494: 1, 7471: 1, 7465: 1, 7384: 1, 7367: 1, 7354: 1, 7335: 1, 7334: 1, 7328: 1, 7320: 1, 7
298: 1, 7296: 1, 7293: 1, 7292: 1, 7278: 1, 7239: 1, 7218: 1, 7215: 1, 7177: 1, 7170: 1, 7142: 1, 7
112: 1, 7105: 1, 7084: 1, 7081: 1, 7079: 1, 7077: 1, 7048: 1, 7043: 1, 7035: 1, 6973: 1, 6970: 1, 6
941: 1, 6936: 1, 6928: 1, 6921: 1, 6914: 1, 6902: 1, 6865: 1, 6854: 1, 6835: 1, 6831: 1, 6812: 1, 6
811: 1, 6807: 1, 6806: 1, 6789: 1, 6742: 1, 6739: 1, 6731: 1, 6713: 1, 6681: 1, 6669: 1, 6650: 1, 6
649: 1, 6637: 1, 6613: 1, 6593: 1, 6573: 1, 6565: 1, 6553: 1, 6550: 1, 6549: 1, 6546: 1, 6535: 1, 6
473: 1, 6455: 1, 6432: 1, 6403: 1, 6401: 1, 6369: 1, 6345: 1, 6328: 1, 6327: 1, 6315: 1, 6308: 1, 6
302: 1, 6296: 1, 6277: 1, 6274: 1, 6265: 1, 6264: 1, 6263: 1, 6251: 1, 6243: 1, 6226: 1, 6219: 1, 6
203: 1, 6166: 1, 6138: 1, 6129: 1, 6114: 1, 6103: 1, 6091: 1, 6077: 1, 6067: 1, 6058: 1, 6052: 1, 6
044: 1, 6037: 1, 6034: 1, 6008: 1, 6002: 1, 5976: 1, 5972: 1, 5971: 1, 5961: 1, 5956: 1, 5947: 1, 5
945: 1, 5941: 1, 5937: 1, 5886: 1, 5872: 1, 5812: 1, 5805: 1, 5803: 1, 5772: 1, 5729: 1, 5727: 1, 5
702: 1, 5698: 1, 5685: 1, 5683: 1, 5666: 1, 5659: 1, 5640: 1, 5619: 1, 5618: 1, 5609: 1, 5602: 1, 5
601: 1, 5598: 1, 5583: 1, 5578: 1, 5568: 1, 5551: 1, 5540: 1, 5533: 1, 5528: 1, 5522: 1, 5513: 1, 5
469: 1, 5444: 1, 5426: 1, 5404: 1, 5402: 1, 5393: 1, 5374: 1, 5371: 1, 5366: 1, 5360: 1, 5341: 1, 5
339: 1, 5330: 1, 5324: 1, 5318: 1, 5284: 1, 5271: 1, 5267: 1, 5266: 1, 5235: 1, 5203: 1, 5193: 1, 5
189: 1, 5182: 1, 5181: 1, 5174: 1, 5155: 1, 5152: 1, 5149: 1, 5133: 1, 5106: 1, 5105: 1, 5104: 1, 5
096: 1, 5085: 1, 5071: 1, 5070: 1, 5063: 1, 5058: 1, 5050: 1, 5031: 1, 5013: 1, 5010: 1, 5006: 1, 4
980: 1, 4970: 1, 4969: 1, 4956: 1, 4955: 1, 4943: 1, 4928: 1, 4919: 1, 4916: 1, 4911: 1, 4902: 1, 4
885: 1, 4877: 1, 4866: 1, 4861: 1, 4860: 1, 4859: 1, 4853: 1, 4844: 1, 4837: 1, 4836: 1, 4835: 1, 4
820: 1, 4809: 1, 4808: 1, 4805: 1, 4801: 1, 4790: 1, 4789: 1, 4784: 1, 4775: 1, 4753: 1, 4737: 1, 4
726: 1, 4725: 1, 4724: 1, 4700: 1, 4687: 1, 4686: 1, 4680: 1, 4663: 1, 4657: 1, 4653: 1, 4652: 1, 4
649: 1, 4641: 1, 4639: 1, 4632: 1, 4623: 1, 4622: 1, 4595: 1, 4570: 1, 4541: 1, 4525: 1, 4513: 1, 4
512: 1, 4498: 1, 4483: 1, 4481: 1, 4480: 1, 4473: 1, 4469: 1, 4465: 1, 4461: 1, 4455: 1, 4440: 1, 4
439: 1, 4422: 1, 4421: 1, 4410: 1, 4405: 1, 4402: 1, 4397: 1, 4392: 1, 4390: 1, 4387: 1, 4374: 1, 4
363: 1, 4352: 1, 4349: 1, 4341: 1, 4331: 1, 4326: 1, 4322: 1, 4317: 1, 4314: 1, 4309: 1, 4293: 1, 4
291: 1, 4289: 1, 4285: 1, 4283: 1, 4274: 1, 4268: 1, 4259: 1, 4257: 1, 4254: 1, 4250: 1, 4240: 1, 4
238: 1, 4236: 1, 4220: 1, 4217: 1, 4215: 1, 4208: 1, 4198: 1, 4195: 1, 4190: 1, 4181: 1, 4160: 1, 4
156: 1, 4152: 1, 4142: 1, 4131: 1, 4130: 1, 4129: 1, 4117: 1, 4098: 1, 4095: 1, 4093: 1, 4089: 1, 4
084: 1, 4068: 1, 4066: 1, 4064: 1, 4056: 1, 4039: 1, 4037: 1, 4036: 1, 4035: 1, 4024: 1, 4019: 1, 4
017: 1, 4011: 1, 4003: 1, 3998: 1, 3982: 1, 3980: 1, 3979: 1, 3947: 1, 3945: 1, 3925: 1, 3924: 1, 3
913: 1, 3906: 1, 3903: 1, 3901: 1, 3897: 1, 3894: 1, 3876: 1, 3875: 1, 3871: 1, 3863: 1, 3853: 1, 3
850: 1, 3848: 1, 3836: 1, 3826: 1, 3824: 1, 3823: 1, 3821: 1, 3815: 1, 3810: 1, 3806: 1, 3798: 1, 3
796: 1, 3792: 1, 3786: 1, 3784: 1, 3778: 1, 3771: 1, 3769: 1, 3764: 1, 3762: 1, 3761: 1, 3759: 1, 3
758: 1, 3748: 1, 3730: 1, 3726: 1, 3724: 1, 3709: 1, 3707: 1, 3691: 1, 3690: 1, 3687: 1, 3681: 1, 3
671: 1, 3659: 1, 3657: 1, 3651: 1, 3650: 1, 3645: 1, 3643: 1, 3639: 1, 3634: 1, 3629: 1, 3621: 1, 3
618: 1, 3616: 1, 3615: 1, 3611: 1, 3609: 1, 3606: 1, 3605: 1, 3601: 1, 3600: 1, 3596: 1, 3592: 1, 3
575: 1, 3573: 1, 3571: 1, 3569: 1, 3567: 1, 3561: 1, 3557: 1, 3553: 1, 3551: 1, 3547: 1, 3544: 1, 3
543: 1, 3527: 1, 3525: 1, 3523: 1, 3519: 1, 3516: 1, 3514: 1, 3506: 1, 3503: 1, 3501: 1, 3495: 1, 3
493: 1, 3487: 1, 3485: 1, 3483: 1, 3480: 1, 3478: 1, 3474: 1, 3464: 1, 3463: 1, 3461: 1, 3457: 1, 3
451: 1, 3449: 1, 3442: 1, 3440: 1, 3439: 1, 3438: 1, 3436: 1, 3432: 1, 3430: 1, 3428: 1, 3427: 1, 3
423: 1, 3422: 1, 3417: 1, 3411: 1, 3409: 1, 3406: 1, 3405: 1, 3381: 1, 3379: 1, 3372: 1, 3370: 1, 3
356: 1, 3355: 1, 3348: 1, 3347: 1, 3346: 1, 3342: 1, 3338: 1, 3323: 1, 3322: 1, 3319: 1, 3315: 1, 3
310: 1, 3309: 1, 3303: 1, 3299: 1, 3297: 1, 3296: 1, 3285: 1, 3283: 1, 3278: 1, 3276: 1, 3270: 1, 3
268: 1, 3260: 1, 3256: 1, 3254: 1, 3251: 1, 3250: 1, 3248: 1, 3241: 1, 3237: 1, 3232: 1, 3225: 1, 3
224: 1, 3223: 1, 3221: 1, 3220: 1, 3213: 1, 3210: 1, 3206: 1, 3202: 1, 3196: 1, 3190: 1, 3189: 1, 3
179: 1, 3177: 1, 3174: 1, 3173: 1, 3171: 1, 3169: 1, 3168: 1, 3164: 1, 3158: 1, 3150: 1, 3146: 1, 3
139: 1, 3131: 1, 3129: 1, 3117: 1, 3111: 1, 3110: 1, 3102: 1, 3100: 1, 3095: 1, 3089: 1, 3087: 1, 3
084: 1, 3078: 1, 3077: 1, 3076: 1, 3064: 1, 3061: 1, 3060: 1, 3053: 1, 3052: 1, 3049: 1, 3037: 1, 3
033: 1, 3030: 1, 3029: 1, 3017: 1, 3013: 1, 3012: 1, 3008: 1, 3007: 1, 3006: 1, 3003: 1, 3002: 1, 2
991: 1, 2988: 1, 2986: 1, 2984: 1, 2979: 1, 2977: 1, 2972: 1, 2965: 1, 2964: 1, 2955: 1, 2953: 1, 2
947: 1, 2942: 1, 2935: 1, 2933: 1, 2931: 1, 2930: 1, 2925: 1, 2923: 1, 2911: 1, 2896: 1, 2889: 1, 2
888: 1, 2884: 1, 2883: 1, 2881: 1, 2880: 1, 2879: 1, 2875: 1, 2871: 1, 2855: 1, 2854: 1, 2853: 1, 2
849: 1, 2848: 1, 2846: 1, 2839: 1, 2838: 1, 2834: 1, 2831: 1, 2828: 1, 2825: 1, 2821: 1, 2815: 1, 2
813: 1, 2805: 1, 2796: 1, 2795: 1, 2771: 1, 2759: 1, 2757: 1, 2755: 1, 2748: 1, 2745: 1, 2743: 1, 2
742: 1, 2734: 1, 2725: 1, 2714: 1, 2712: 1, 2710: 1, 2702: 1, 2701: 1, 2700: 1, 2697: 1, 2688: 1, 2
684: 1, 2683: 1, 2682: 1, 2679: 1, 2670: 1, 2666: 1, 2665: 1, 2661: 1, 2655: 1, 2649: 1, 2647: 1, 2
646: 1, 2631: 1, 2629: 1, 2625: 1, 2624: 1, 2618: 1, 2615: 1, 2610: 1, 2608: 1, 2606: 1, 2597: 1, 2
596: 1, 2591: 1, 2590: 1, 2586: 1, 2584: 1, 2582: 1, 2580: 1, 2578: 1, 2575: 1, 2574: 1, 2559: 1, 2
557: 1, 2554: 1, 2550: 1, 2549: 1, 2546: 1, 2545: 1, 2544: 1, 2543: 1, 2541: 1, 2540: 1, 2534: 1, 2
529: 1, 2523: 1, 2512: 1, 2510: 1, 2509: 1, 2507: 1, 2506: 1, 2497: 1, 2494: 1, 2493: 1, 2489: 1, 2
488: 1, 2487: 1, 2482: 1, 2479: 1, 2476: 1, 2475: 1, 2473: 1, 2470: 1, 2467: 1, 2466: 1, 2465: 1, 2
463: 1, 2462: 1, 2461: 1, 2460: 1, 2456: 1, 2454: 1, 2452: 1, 2444: 1, 2441: 1, 2439: 1, 2438: 1, 2
434: 1, 2431: 1, 2430: 1, 2424: 1, 2419: 1, 2418: 1, 2417: 1, 2415: 1, 2414: 1, 2404: 1, 2400: 1, 2
397: 1, 2394: 1, 2393: 1, 2392: 1, 2391: 1, 2390: 1, 2389: 1, 2387: 1, 2386: 1, 2381: 1, 2379: 1, 2
374: 1, 2369: 1, 2365: 1, 2364: 1, 2363: 1, 2361: 1, 2356: 1, 2349: 1, 2348: 1, 2336: 1, 2331: 1, 2
329: 1, 2328: 1, 2325: 1, 2323: 1, 2322: 1, 2321: 1, 2317: 1, 2311: 1, 2305: 1, 2303: 1, 2299: 1, 2
293: 1, 2292: 1, 2283: 1, 2276: 1, 2275: 1, 2270: 1, 2268: 1, 2267: 1, 2266: 1, 2265: 1, 2264: 1, 2
259: 1, 2258: 1, 2257: 1, 2256: 1, 2255: 1, 2254: 1, 2248: 1, 2244: 1, 2236: 1, 2233: 1, 2232: 1, 2

```
231: 1, 2225: 1, 2221: 1, 2216: 1, 2213: 1, 2206: 1, 2202: 1, 2197: 1, 2195: 1, 2192: 1, 2191: 1, 2
187: 1, 2186: 1, 2184: 1, 2179: 1, 2178: 1, 2176: 1, 2174: 1, 2170: 1, 2169: 1, 2168: 1, 2167: 1, 2
166: 1, 2161: 1, 2158: 1, 2157: 1, 2149: 1, 2144: 1, 2140: 1, 2134: 1, 2131: 1, 2129: 1, 2128: 1, 2
127: 1, 2125: 1, 2124: 1, 2122: 1, 2116: 1, 2114: 1, 2112: 1, 2110: 1, 2106: 1, 2105: 1, 2104: 1, 2
102: 1, 2100: 1, 2097: 1, 2089: 1, 2086: 1, 2085: 1, 2084: 1, 2080: 1, 2079: 1, 2076: 1, 2075: 1, 2
074: 1, 2071: 1, 2066: 1, 2064: 1, 2063: 1, 2061: 1, 2059: 1, 2058: 1, 2055: 1, 2054: 1, 2051: 1, 2
049: 1, 2043: 1, 2038: 1, 2033: 1, 2032: 1, 2030: 1, 2029: 1, 2028: 1, 2020: 1, 2016: 1, 2014: 1, 2
012: 1, 2008: 1, 2007: 1, 2005: 1, 2004: 1, 2001: 1, 1999: 1, 1998: 1, 1997: 1, 1993: 1, 1992: 1, 1
989: 1, 1987: 1, 1986: 1, 1985: 1, 1982: 1, 1979: 1, 1978: 1, 1976: 1, 1974: 1, 1970: 1, 1967: 1, 1
966: 1, 1965: 1, 1963: 1, 1961: 1, 1955: 1, 1953: 1, 1951: 1, 1950: 1, 1949: 1, 1947: 1, 1946: 1, 1
942: 1, 1941: 1, 1940: 1, 1939: 1, 1935: 1, 1931: 1, 1930: 1, 1929: 1, 1928: 1, 1925: 1, 1923: 1, 1
921: 1, 1919: 1, 1918: 1, 1917: 1, 1912: 1, 1909: 1, 1906: 1, 1904: 1, 1903: 1, 1901: 1, 1897: 1, 1
894: 1, 1893: 1, 1888: 1, 1885: 1, 1881: 1, 1879: 1, 1875: 1, 1871: 1, 1867: 1, 1865: 1, 1864: 1, 1
860: 1, 1855: 1, 1854: 1, 1851: 1, 1847: 1, 1844: 1, 1843: 1, 1841: 1, 1840: 1, 1839: 1, 1834: 1, 1
832: 1, 1830: 1, 1827: 1, 1826: 1, 1822: 1, 1816: 1, 1812: 1, 1810: 1, 1807: 1, 1803: 1, 1800: 1, 1
796: 1, 1794: 1, 1793: 1, 1790: 1, 1789: 1, 1787: 1, 1783: 1, 1782: 1, 1779: 1, 1775: 1, 1773: 1, 1
770: 1, 1769: 1, 1766: 1, 1765: 1, 1764: 1, 1760: 1, 1759: 1, 1757: 1, 1756: 1, 1752: 1, 1750: 1, 1
747: 1, 1746: 1, 1745: 1, 1744: 1, 1737: 1, 1733: 1, 1730: 1, 1728: 1, 1727: 1, 1724: 1, 1720: 1, 1
719: 1, 1716: 1, 1714: 1, 1713: 1, 1709: 1, 1705: 1, 1702: 1, 1701: 1, 1696: 1, 1692: 1, 1689: 1, 1
688: 1, 1684: 1, 1676: 1, 1674: 1, 1673: 1, 1670: 1, 1668: 1, 1667: 1, 1666: 1, 1657: 1, 1654: 1, 1
653: 1, 1652: 1, 1651: 1, 1647: 1, 1645: 1, 1644: 1, 1643: 1, 1637: 1, 1628: 1, 1624: 1, 1623: 1, 1
620: 1, 1617: 1, 1616: 1, 1611: 1, 1610: 1, 1605: 1, 1604: 1, 1602: 1, 1599: 1, 1598: 1, 1595: 1, 1
593: 1, 1583: 1, 1579: 1, 1578: 1, 1576: 1, 1575: 1, 1572: 1, 1568: 1, 1564: 1, 1561: 1, 1556: 1, 1
552: 1, 1551: 1, 1549: 1, 1547: 1, 1546: 1, 1545: 1, 1542: 1, 1541: 1, 1539: 1, 1538: 1, 1531: 1, 1
530: 1, 1529: 1, 1523: 1, 1522: 1, 1520: 1, 1516: 1, 1513: 1, 1512: 1, 1510: 1, 1509: 1, 1508: 1, 1
507: 1, 1506: 1, 1505: 1, 1504: 1, 1503: 1, 1502: 1, 1500: 1, 1499: 1, 1497: 1, 1496: 1, 1494: 1, 1
492: 1, 1490: 1, 1489: 1, 1488: 1, 1486: 1, 1482: 1, 1480: 1, 1479: 1, 1478: 1, 1477: 1, 1475: 1, 1
474: 1, 1473: 1, 1465: 1, 1464: 1, 1463: 1, 1461: 1, 1454: 1, 1452: 1, 1451: 1, 1448: 1, 1447: 1, 1
445: 1, 1441: 1, 1440: 1, 1439: 1, 1436: 1, 1431: 1, 1426: 1, 1425: 1, 1413: 1, 1412: 1, 1410: 1, 1
406: 1, 1404: 1, 1402: 1, 1399: 1, 1397: 1, 1395: 1, 1388: 1, 1385: 1, 1381: 1, 1377: 1, 1376: 1, 1
374: 1, 1370: 1, 1368: 1, 1361: 1, 1358: 1, 1356: 1, 1354: 1, 1349: 1, 1348: 1, 1344: 1, 1341: 1, 1
338: 1, 1336: 1, 1335: 1, 1330: 1, 1322: 1, 1321: 1, 1319: 1, 1318: 1, 1313: 1, 1307: 1, 1306: 1, 1
303: 1, 1301: 1, 1300: 1, 1298: 1, 1297: 1, 1296: 1, 1295: 1, 1294: 1, 1292: 1, 1289: 1, 1284: 1, 1
281: 1, 1279: 1, 1274: 1, 1267: 1, 1266: 1, 1265: 1, 1264: 1, 1261: 1, 1260: 1, 1257: 1, 1255: 1, 1
254: 1, 1248: 1, 1242: 1, 1239: 1, 1237: 1, 1236: 1, 1225: 1, 1223: 1, 1221: 1, 1216: 1, 1212: 1, 1
206: 1, 1205: 1, 1203: 1, 1202: 1, 1201: 1, 1198: 1, 1195: 1, 1194: 1, 1187: 1, 1180: 1, 1179: 1, 1
171: 1, 1161: 1, 1157: 1, 1154: 1, 1153: 1, 1145: 1, 1144: 1, 1142: 1, 1140: 1, 1132: 1, 1121: 1, 1
119: 1, 1117: 1, 1116: 1, 1113: 1, 1109: 1, 1108: 1, 1097: 1, 1093: 1, 1092: 1, 1088: 1, 1087: 1, 1
080: 1, 1076: 1, 1074: 1, 1071: 1, 1067: 1, 1061: 1, 1060: 1, 1056: 1, 1055: 1, 1045: 1, 1041: 1, 1
032: 1, 1029: 1, 1028: 1, 1024: 1, 1022: 1, 1011: 1, 1010: 1, 1009: 1, 1005: 1, 1004: 1, 999: 1, 98
4: 1, 979: 1, 972: 1, 960: 1, 956: 1, 954: 1, 950: 1, 949: 1, 947: 1, 919: 1, 914: 1, 912: 1, 909:
1, 902: 1, 901: 1, 895: 1, 886: 1, 884: 1, 865: 1, 857: 1, 850: 1, 843: 1, 842: 1, 827: 1, 818: 1,
812: 1, 794: 1, 782: 1, 770: 1, 749: 1, 717: 1, 714: 1})
```

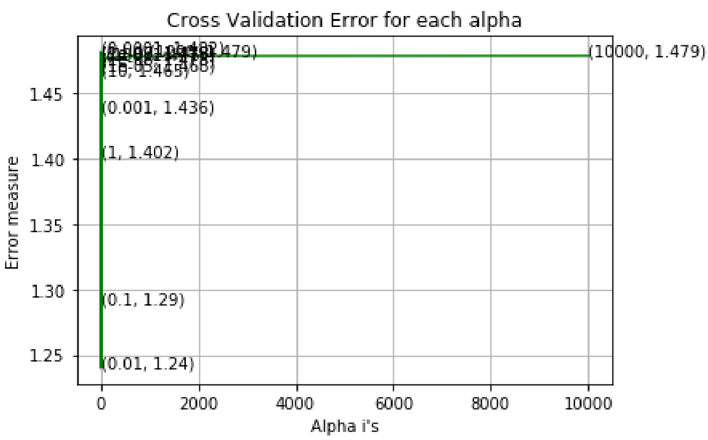
Logistic Regression of TEXT

In [140]:

```python
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-8, 5)]

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

```
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-08 The log loss is: 1.4727888305336334
For values of alpha =  1e-07 The log loss is: 1.4776048454940192
For values of alpha =  1e-06 The log loss is: 1.4778253172597984
For values of alpha =  1e-05 The log loss is: 1.4678620121224368
For values of alpha =  0.0001 The log loss is: 1.4820202233231972
For values of alpha =  0.001 The log loss is: 1.4360882588926793
For values of alpha =  0.01 The log loss is: 1.2398587151810045
For values of alpha =  0.1 The log loss is: 1.2897157356937814
For values of alpha =  1 The log loss is: 1.4018170339132392
For values of alpha =  10 The log loss is: 1.464565078438739
For values of alpha =  100 The log loss is: 1.4772165779131732
For values of alpha =  1000 The log loss is: 1.478618290680127
For values of alpha =  10000 The log loss is: 1.4787689577716876
```




```
For values of best alpha =  0.01 The train log loss is: 0.8464733539488416
For values of best alpha =  0.01 The cross validation log loss is: 1.2398587151810045
For values of best alpha =  0.01 The test log loss is: 1.1880600474877914
```

Stacking all the features Train,Test,CV

In [0]:

```
word_count_train = train_df['word_count']
min_max_scaler_train = preprocessing.MinMaxScaler()
word_count_train = min_max_scaler_train.fit_transform(word_count_train.values.reshape(-1,1))

word_count_test = test_df['word_count']
min_max_scaler_test = preprocessing.MinMaxScaler()
word_count_test = min_max_scaler_test.fit_transform(word_count_test.values.reshape(-1,1))

word_count_cv = cv_df['word_count']
min_max_scaler_cv = preprocessing.MinMaxScaler()
word_count_cv = min_max_scaler_cv.fit_transform(word_count_cv.values.reshape(-1,1))


import scipy
word_count_train = scipy.sparse.csr_matrix(word_count_train)
word_count_test = scipy.sparse.csr_matrix(word_count_test)
word_count_cv = scipy.sparse.csr_matrix(word_count_cv)
```

In [0]:

```
train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)
)

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,
train_text_feature_onehotCoding,word_count_train,char_count_train,word_density_count_train,digits_c
ount_train,gene_text_count_train,variation_text_count_train,capital_count_train)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding,
test_text_feature_onehotCoding,word_count_test,char_count_test,word_density_count_test,digits_count
_test,gene_text_count_test,variation_text_count_test,capital_count_test)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding,word_count_cv,c
har_count_cv,word_density_count_cv,digits_count_cv,gene_text_count_cv,variation_text_count_cv,capi
tal_count_cv)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [142]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding
.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 778298)
(number of data points * number of features) in test data =  (665, 778298)
(number of data points * number of features) in cross validation data = (532, 778298)
```

In [143]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shap
e)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_responseCoding.shape)
```

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

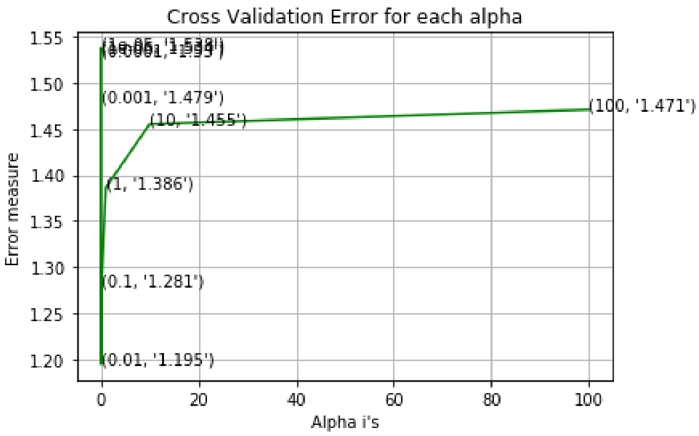
Logistic Regression with CountVectorizer(bigrams) with Class_weight=balanced

In [144]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

```
for alpha = 1e-06
Log Loss : 1.5341048035829712
for alpha = 1e-05
Log Loss : 1.5376131715054318
for alpha = 0.0001
Log Loss : 1.5295732418671029
for alpha = 0.001
Log Loss : 1.4792586182147747
for alpha = 0.01
Log Loss : 1.1949286764798583
for alpha = 0.1
Log Loss : 1.2805395131320085
for alpha = 1
Log Loss : 1.3862896203158113
for alpha = 10
Log Loss : 1.4552222764169176
for alpha = 100
Log Loss : 1.4709471343184406
```

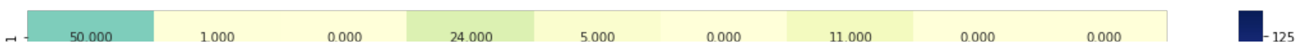



```
For values of best alpha =  0.01 The train log loss is: 0.8119633685513874
For values of best alpha =  0.01 The cross validation log loss is: 1.1949286764798583
For values of best alpha =  0.01 The test log loss is: 1.1570845075534
```

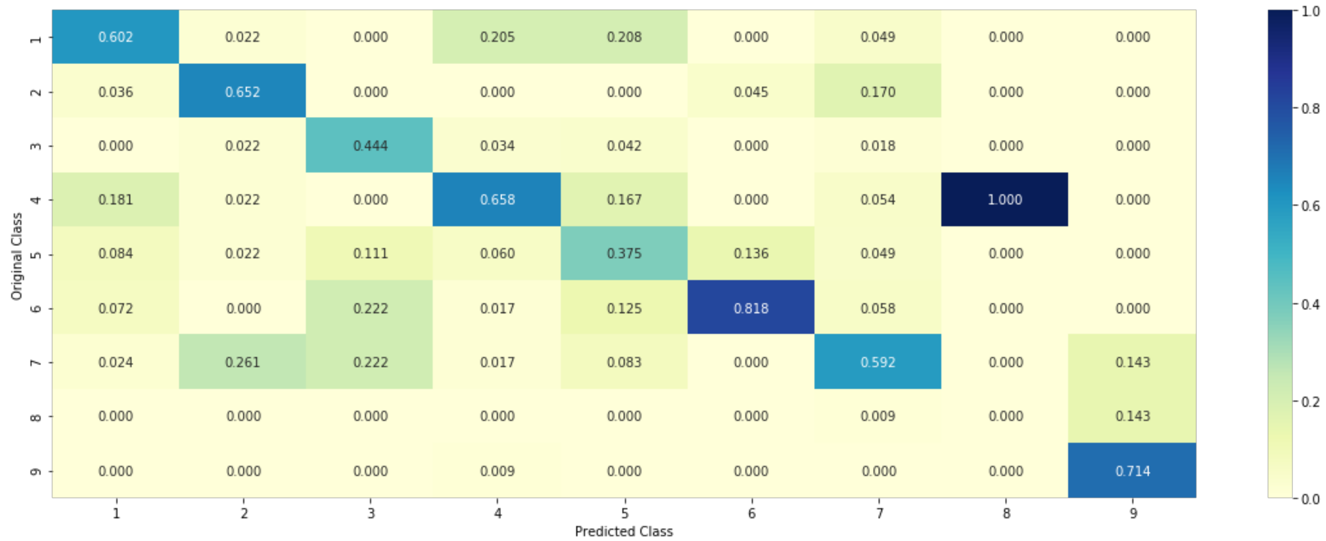
In [145]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

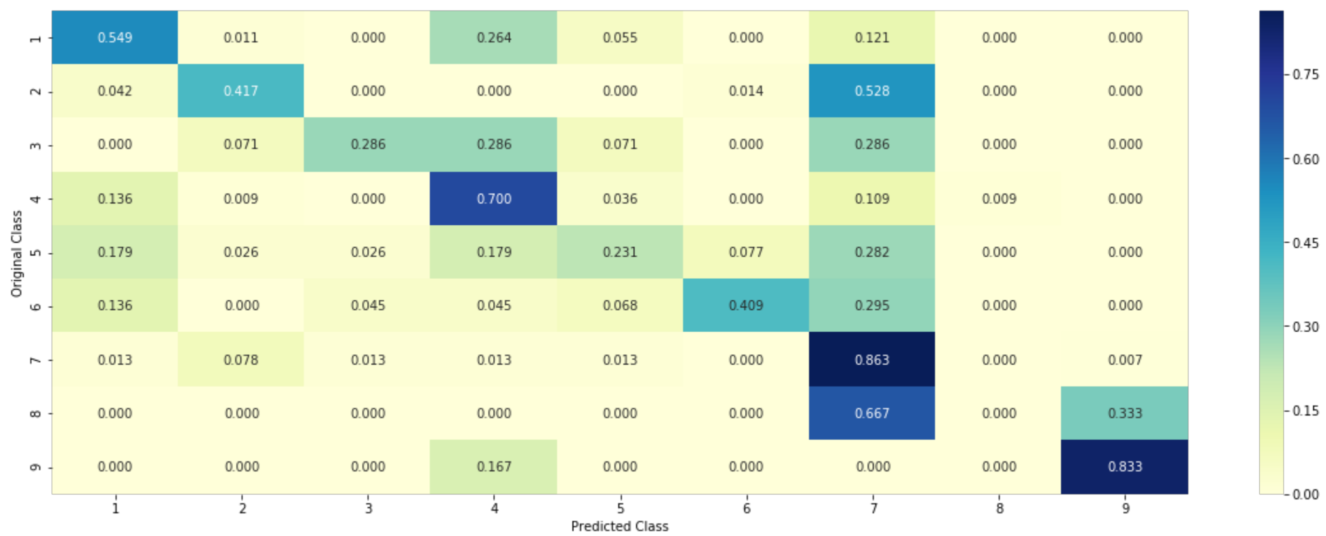
```
Log loss : 1.1949286764798583
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```




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




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



In [146]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
```

```
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.02   0.0377 0.0099 0.0148 0.0151 0.0067 0.886  0.0039 0.0059]]
Actual Class : 7
--------------------------------------------------
```

Logistic Regression with CountVectorizer(bigrams)

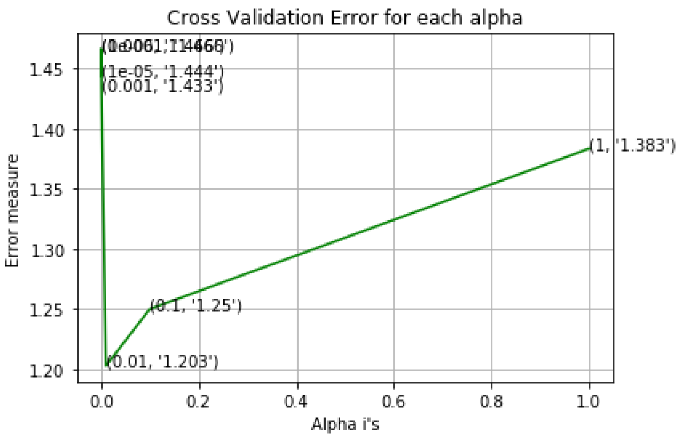
Class_weight= "not balanced"

In [148]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------



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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.4662161613566327
for alpha = 1e-05
Log Loss : 1.4441979670587377
for alpha = 0.0001
Log Loss : 1.4659962788508494
for alpha = 0.001
Log Loss : 1.4329148354191141
for alpha = 0.01
Log Loss : 1.2027265110088705
for alpha = 0.1
Log Loss : 1.2500862405364295
for alpha = 1
Log Loss : 1.3834396411374439
```



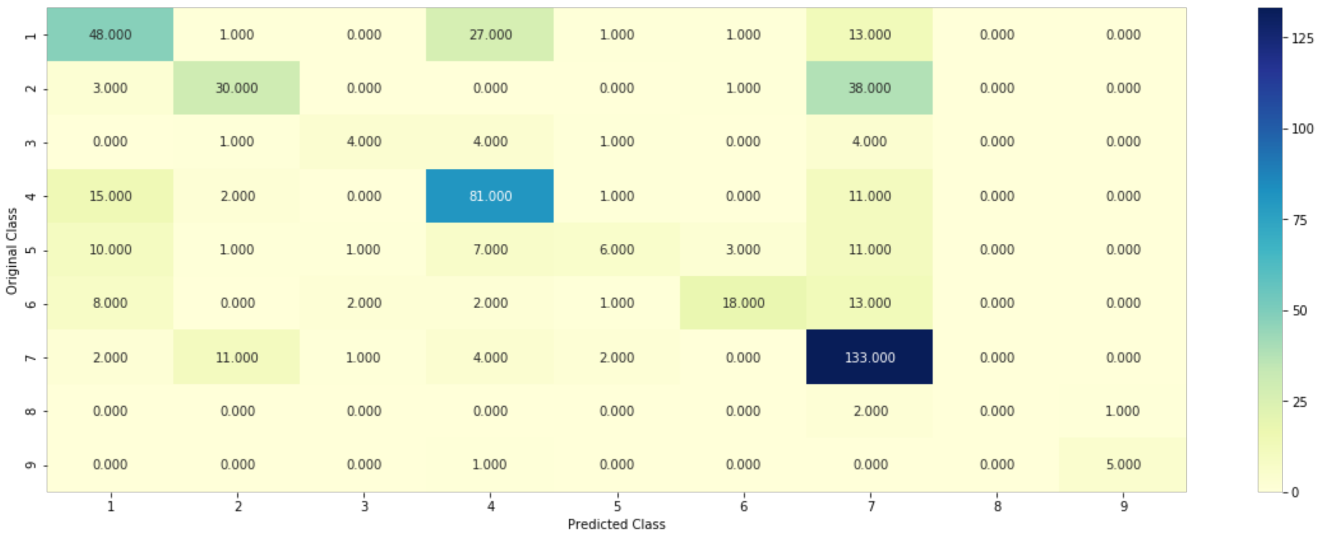

```
For values of best alpha =  0.01 The train log loss is: 0.8044444928632346
For values of best alpha =  0.01 The cross validation log loss is: 1.2027265110088705
For values of best alpha =  0.01 The test log loss is: 1.1655697860608085
```

In [149]:

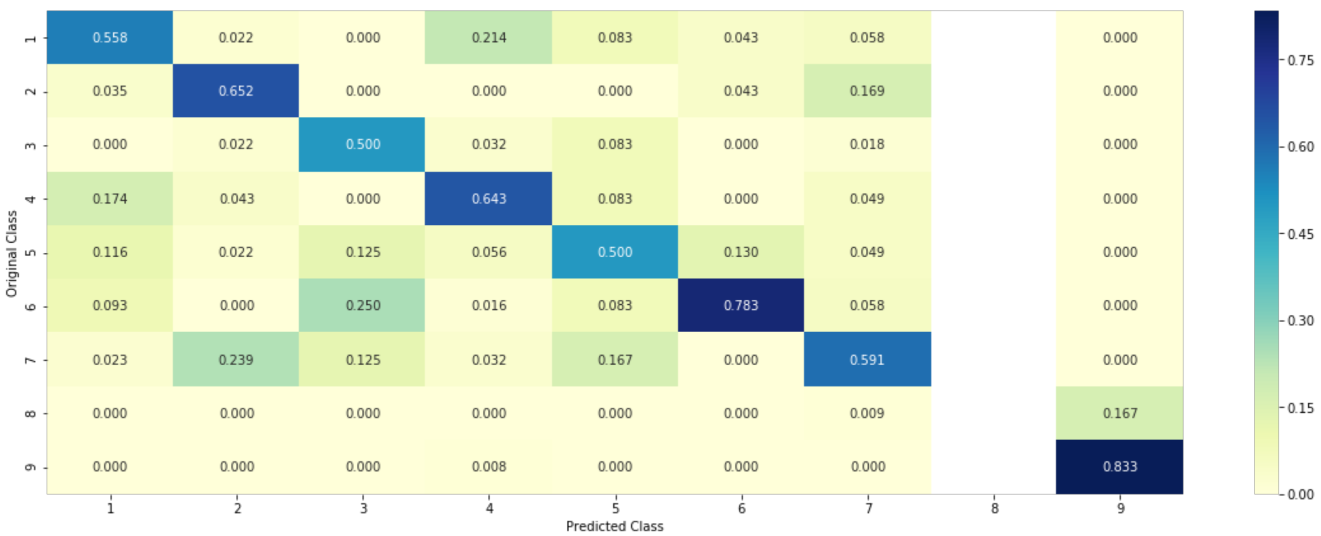
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

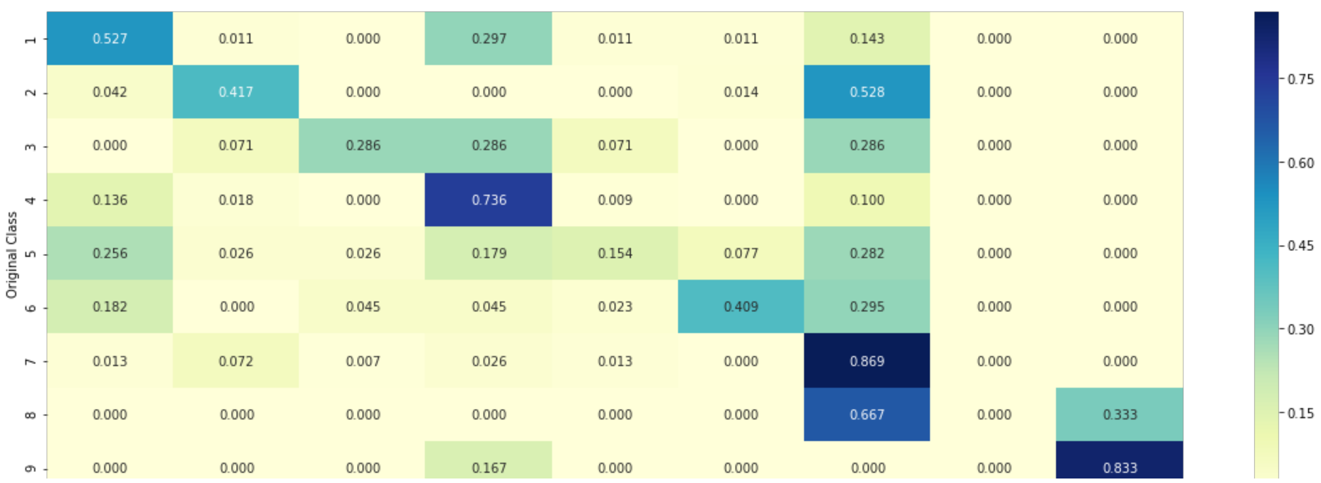
```
Log loss : 1.2027265110088705
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```

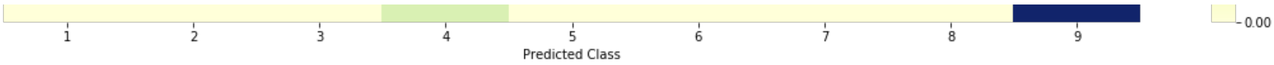


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





In [150]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0261 0.0425 0.004  0.0216 0.0147 0.0054 0.8818 0.0027 0.0012]]
Actual Class : 7
--------------------------------------------------
```

In [151]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.123  0.0387 0.0045 0.7765 0.0167 0.0052 0.0314 0.0028 0.0013]]
Actual Class : 4
--------------------------------------------------
```

In [0]:

```
## Assignment-4  Feature Engineering to reduce log loss
```

Assignment4.TfidfVectorizer

In [0]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,ngram_range=(1,6),max_features= 30000)
train_text_feature_onehotCodingdf = text_vectorizer.fit_transform(train_df['TEXT'])
```

In [0]:

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

In [155]:

```
train_text_feature_onehotCoding.shape
```

Out[155]:

```
(2124, 30000)
```

In [0]:

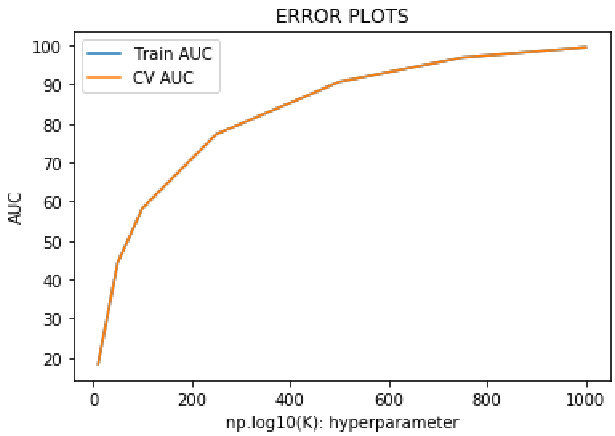
```
import numpy as np
```

```
from sklearn.decomposition import TruncatedSVD
z = list([10,50,100,250,500,750,1000])

explained_variances = []
for j in tqdm(z):

  model = TruncatedSVD(n_components=j)
  X_proj = model.fit_transform(train_text_feature_onehotCoding)
  explained_variances.append(model.explained_variance_ratio_.sum() * 100)
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
 14%|█         | 1/7 [00:02<00:15,  2.64s/it]
 29%|██        | 2/7 [00:11<00:22,  4.52s/it]
 43%|███       | 3/7 [00:29<00:34,  8.60s/it]
 57%|████      | 4/7 [01:07<00:52, 17.42s/it]
 71%|█████     | 5/7 [02:13<01:03, 31.91s/it]
 86%|██████    | 6/7 [03:52<00:52, 52.08s/it]
100%|███████   | 7/7 [06:03<00:00, 75.81s/it]
```

In [0]:

```
plt.plot(z, explained_variances, label='Train AUC')
plt.plot(z, explained_variances, label='CV AUC')
plt.legend()
plt.xlabel("np.log10(K): hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:

```
model = TruncatedSVD(n_components=2000).fit(train_text_feature_onehotCoding)
train_text_feature_onehotCoding = model.transform(train_text_feature_onehotCoding)

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])

test_text_feature_onehotCoding = model.transform(test_text_feature_onehotCoding)

cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])

cv_text_feature_onehotCoding = model.transform(cv_text_feature_onehotCoding)
```

Stack train,test,cv

In [0]:

```
train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding)
)

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,
```

```
train_text_feature_onehotCoding,word_count_train,char_count_train,word_density_count_train,digits_c
ount_train,gene_text_count_train,variation_text_count_train,capital_count_train)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding,
test_text_feature_onehotCoding,word_count_test,char_count_test,word_density_count_test,digits_count
_test,gene_text_count_test,variation_text_count_test,capital_count_test)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding,word_count_cv,c
har_count_cv,word_density_count_cv,digits_count_cv,gene_text_count_cv,variation_text_count_cv,capi
tal_count_cv)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [158]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding
.shape)
```

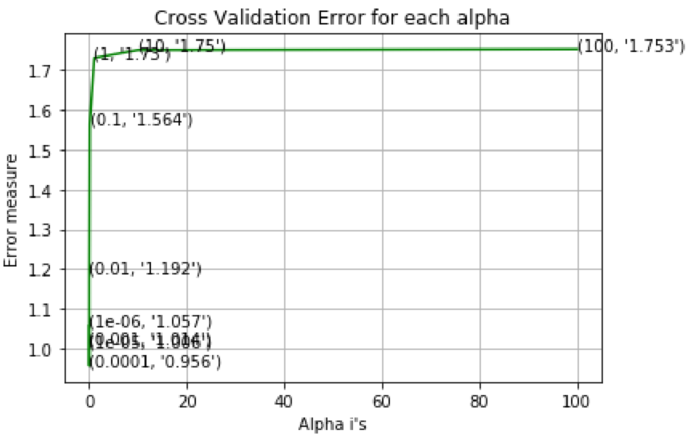
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 4216)
(number of data points * number of features) in test data =  (665, 4216)
(number of data points * number of features) in cross validation data = (532, 4216)
```

Logistic-Regression with class_weight="balanced"

In [159]:

```
alpha = [10 ** x for x in range(-8, 5)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.0573302426915048
for alpha = 1e-05
Log Loss : 1.0062605852484772
for alpha = 0.0001
Log Loss : 0.9562182449847768
for alpha = 0.001
Log Loss : 1.0137424683646532
for alpha = 0.01
Log Loss : 1.1919773198732155
for alpha = 0.1
Log Loss : 1.5641332744700065
for alpha = 1
Log Loss : 1.7304884495064505
for alpha = 10
Log Loss : 1.750489531888565
for alpha = 100
Log Loss : 1.7525509361615015
```

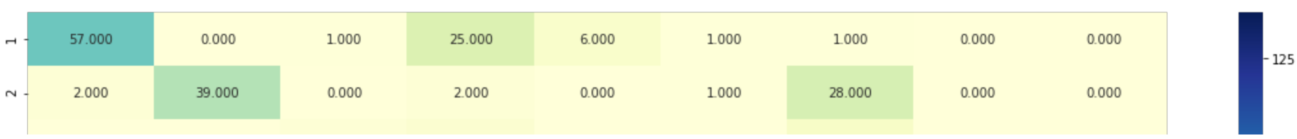



```
For values of best alpha =  0.0001 The train log loss is: 0.42334635210635213
For values of best alpha =  0.0001 The cross validation log loss is: 0.9562182449847768
For values of best alpha =  0.0001 The test log loss is: 0.9892512002742894
```

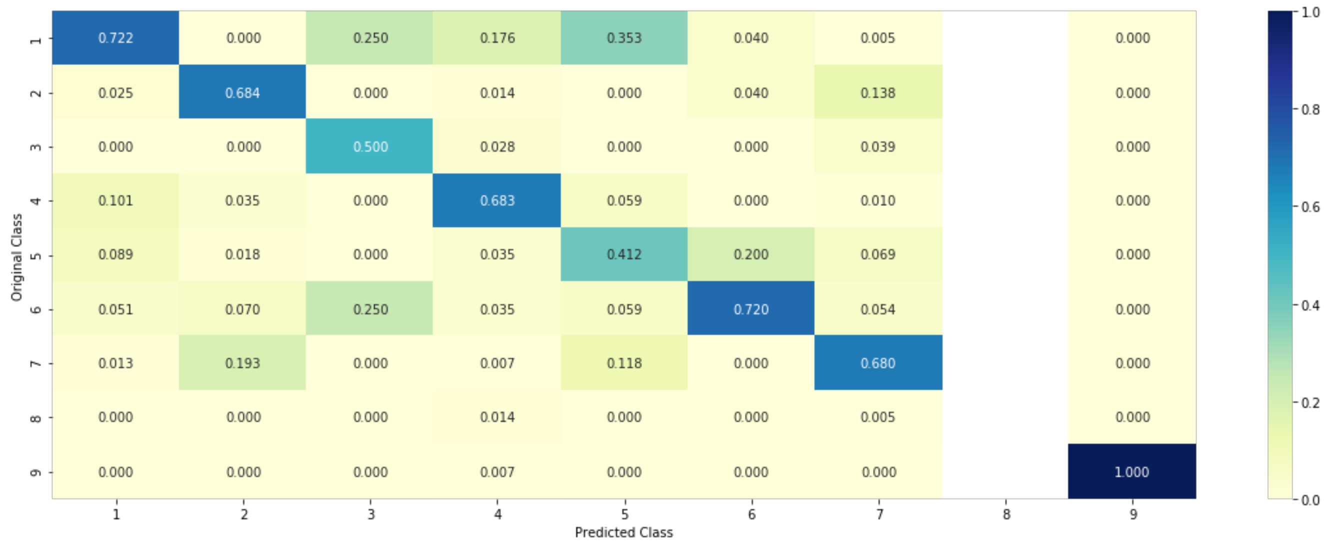
In [160]:

```python
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

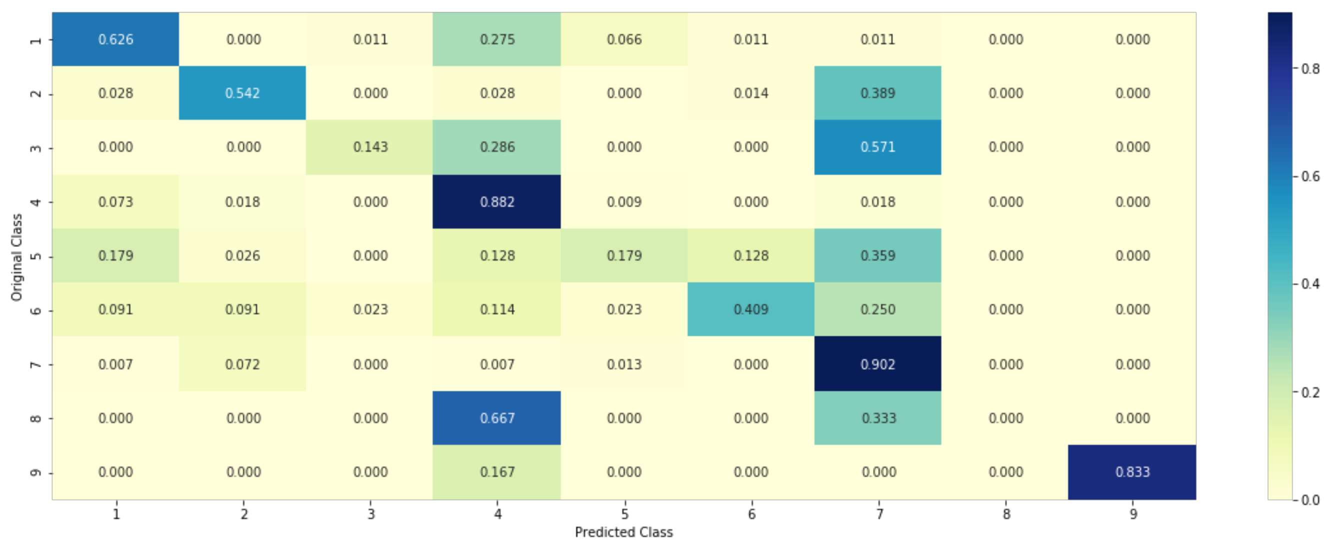
```
Log loss : 0.9562182449847768
Number of mis-classified points : 0.3176691729323308
-------------------- Confusion matrix --------------------
```




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




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



Pretty Table

1.Assignment- Tfidf with 1000 features

In [16]:

```
import numpy as np
from prettytable import PrettyTable
```

```
x = PrettyTable()
x.field_names = ["Model", "alpha","Train-LL","CV-LL","Test-LL","MisClassified%"]


x.add_row(["Naive-Bayes-OneHotEncoding",0.001,0.506,1.229,1.179,39.4])
x.add_row(["KNN-ResponseCoding",11,0.640,1.062,1.097,35.7])
x.add_row(["Logistic-Regression-Balanced-OneHot",0.0001,0.442,0.989,0.967,34.5])
x.add_row(["Logistic-Regression-UnBalanced-OneHot",0.0001,0.434,0.994,0.966,34.3])
x.add_row(["SVM-OneHotEncoding",0.0001,0.474,1.009,0.976,34.5])
x.add_row(["Stacking",0.1,0.344,1.091,1.054,35.7])

print(x)
```

```
+---------------------------------------+--------+----------+-------+---------+----------------+
|                 Model                 | alpha  | Train-LL | CV-LL | Test-LL | MisClassified% |
+---------------------------------------+--------+----------+-------+---------+----------------+
|       Naive-Bayes-OneHotEncoding       | 0.001  |  0.506   | 1.229 |  1.179  |      39.4      |
|           KNN-ResponseCoding           |   11   |   0.64   | 1.062 |  1.097  |      35.7      |
|  Logistic-Regression-Balanced-OneHot  | 0.0001 |  0.442   | 0.989 |  0.967  |      34.5      |
| Logistic-Regression-UnBalanced-OneHot | 0.0001 |  0.434   | 0.994 |  0.966  |      34.3      |
|           SVM-OneHotEncoding           | 0.0001 |  0.474   | 1.009 |  0.976  |      34.5      |
|                Stacking                |  0.1   |  0.344   | 1.091 |  1.054  |      35.7      |
+---------------------------------------+--------+----------+-------+---------+----------------+
```

In [21]:

```
x1 = PrettyTable()
x1.field_names = ["Model", "n_estimators","Depth","Train-LL","CV-LL","Test-LL","MisClassified%"]
x1.add_row(["RandomForest(one-hot-Encoding)",2000,5,0.8617,1.238,1.152,43.9])
x1.add_row(["RandomForest(ResponseCoding)",100,5,0.0562,1.299,1.335,46.6])
print(x1)
```

```
+--------------------------------+--------------+-------+----------+-------+---------+-------------
+
|             Model              | n_estimators | Depth | Train-LL | CV-LL | Test-LL |
MisClassified% |
+--------------------------------+--------------+-------+----------+-------+---------+-------------
+
| RandomForest(one-hot-Encoding) |     2000     |   5   |  0.8617  | 1.238 |  1.152  |      43.9
|
|  RandomForest(ResponseCoding)  |     100      |   5   |  0.0562  | 1.299 |  1.335  |      46.6
|
+--------------------------------+--------------+-------+----------+-------+---------+-------------
+
```

2.Assignment with CountVectorizer with Bigrams

In [17]:

```
y = PrettyTable()
y.field_names = ["Model", "alpha","Train-LL","CV-LL","Test-LL","MisClassified%"]
y.add_row(["Logistic-Regression-Balanced-OneHot",0.01,0.8119,1.194,1.157,38.9])
y.add_row(["Logistic-Regression-UnBalanced-OneHot",0.01,0.8044,1.2027,1.1655,38.9])
print(y)
```

```
+---------------------------------------+-------+----------+--------+---------+----------------+
|                 Model                 | alpha | Train-LL | CV-LL  | Test-LL | MisClassified% |
+---------------------------------------+-------+----------+--------+---------+----------------+
|  Logistic-Regression-Balanced-OneHot  |  0.01 |  0.8119  | 1.194  |  1.157  |      38.9      |
| Logistic-Regression-UnBalanced-OneHot |  0.01 |  0.8044  | 1.2027 |  1.1655 |      38.9      |
+---------------------------------------+-------+----------+--------+---------+----------------+
```

3.Assignment with Feature_Engineering-(Tfidf with ngrams = 6 and TruncatedSVD and more features)

In [19]:

```
z = PrettyTable()
z.field_names = ["Model", "alpha","Train-LL","CV-LL","Test-LL","MisClassified%"]
z.add_row(["Logistic-Regression-Balanced-OneHot",0.0001,0.423,0.956,0.989,31.7])
```

```
print(z)
```

```
+-------------------------------------+---------+----------+-------+---------+-----------------+
|                Model                |  alpha  | Train-LL | CV-LL | Test-LL | MisClassified% |
+-------------------------------------+---------+----------+-------+---------+-----------------+
| Logistic-Regression-Balanced-OneHot | 0.0001 |  0.423   | 0.956 |  0.989  |       31.7      |
+-------------------------------------+---------+----------+-------+---------+-----------------+
```