Personalized cancer diagnosis

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

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 [261]:
```
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 TfidfVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [262]:
```
data = pd.read_csv('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[262]:

	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 [263]:
```
# note the seprator in this file
data_text =pd.read_csv("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[263]:

	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 [264]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [265]:
```
#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 : 233.52445793654806 seconds
```

In [266]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[266]:

	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 [267]:
```
result[result.isnull().any(axis=1)]
```

Out[267]:

	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 [268]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [269]:
```
result[result['ID']==1109]
```

Out[269]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [270]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of output varaible 'y_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 [271]:
```
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])
```

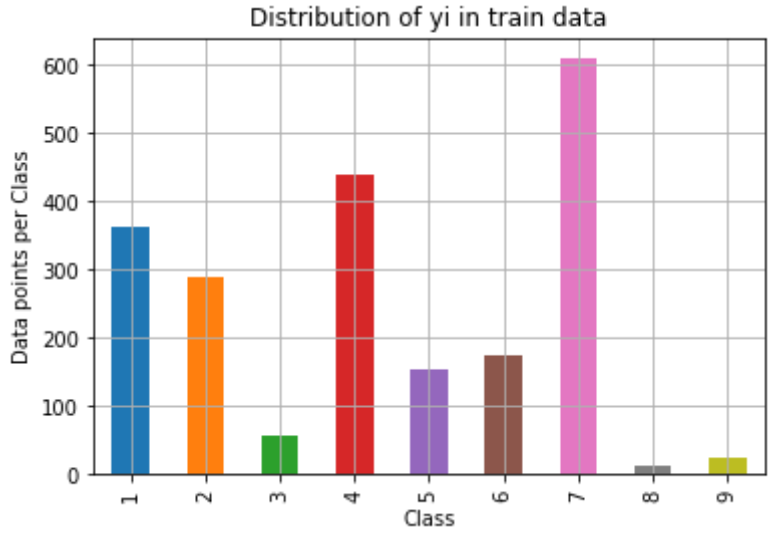
```
Number of data points in train data: 2124
Number of data points in test data: 665
```

```
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

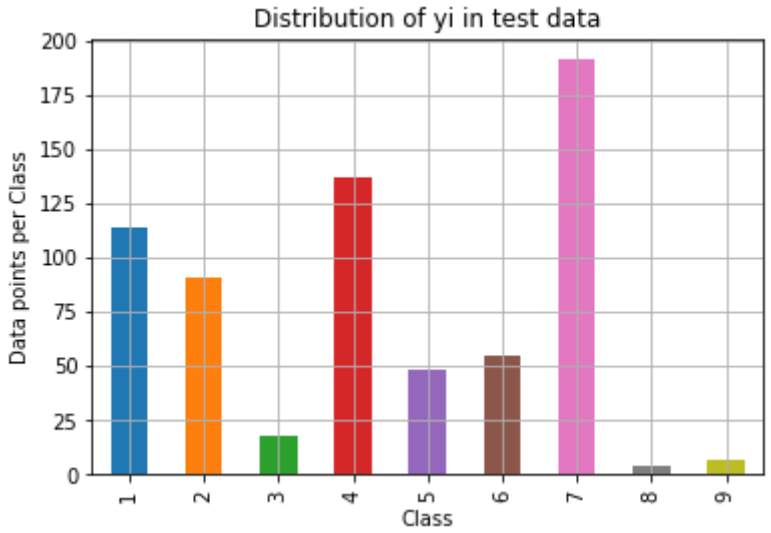
In [272]:
```
# it returns a dict, keys as class labels and values as the number of data poi
nts in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.round((train_class_distribution.values[i]/train_df.shape[0]
*100), 3), '%)')


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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.v
alues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*10
0), 3), '%)')

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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```

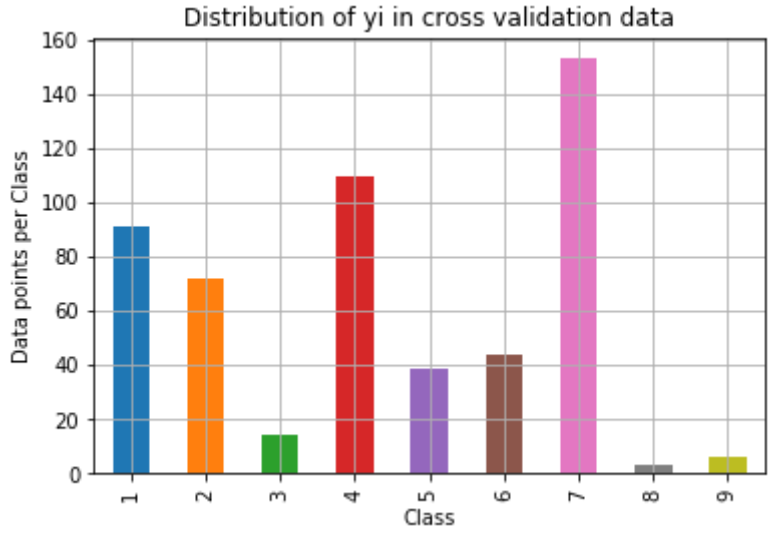



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 %)
------------------------------------------------------------------------
-----
```




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




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

3.2 Prediction using a 'Random' Model

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

In [273]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at 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, y
ticklabels=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, y
ticklabels=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, y
ticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

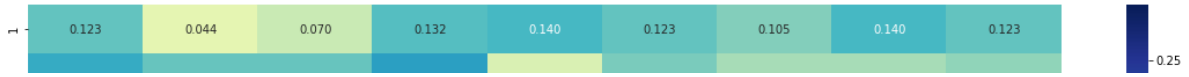
In [274]:

```
# 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 thei
r 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))


# 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_predicte
d_y, eps=1e-15))

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

```
Log loss on Cross Validation Data using Random Model 2.473723812424362
Log loss on Test Data using Random Model 2.495248253720613
-------------------- Confusion matrix --------------------
```

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

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



3.3 Univariate Analysis

In [275]:
```
# 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 cl
ass1 + 10*alpha / number 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) representat
ion 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()
    #print(value_count)

    # 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 occu
red 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 occured 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
        #print(gv_dict)
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818
18177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.
03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.06122448979591836, 0.0612244897959
18366, 0.27040816326530615, 0.06122448979591836, 0.066326530612244902, 0.0510
20408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818
1818181818177, 0.06818181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.05
6818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060
6060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069
182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920
5295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.2715
2317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333
33334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.299
99999999999999, 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 fea
ture 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 [276]:
```
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))
```

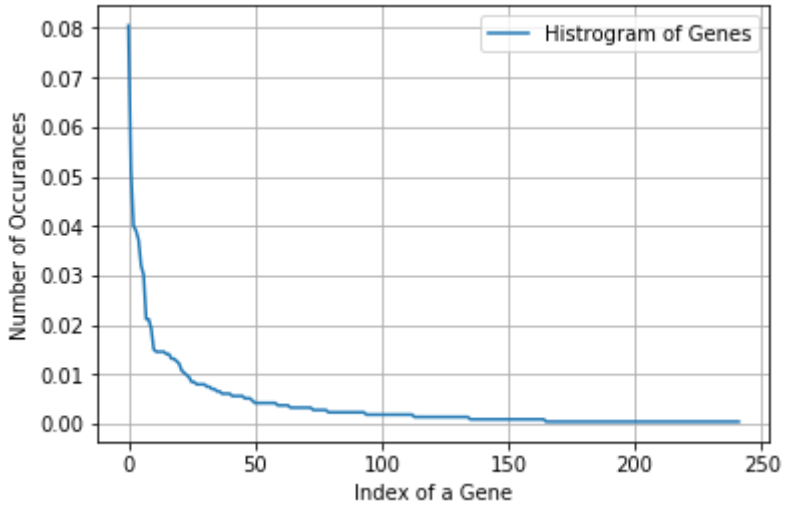
```
Number of Unique Genes : 242
BRCA1     171
TP53      109
PTEN       85
EGFR       83
BRCA2      79
KIT        68
BRAF       64
ALK        45
PDGFRA     45
ERBB2      41
Name: Gene, dtype: int64
```

In [277]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

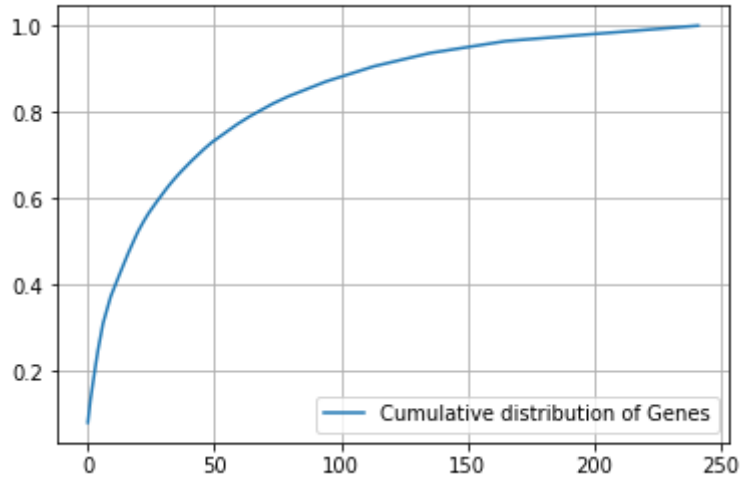
```
Ans: There are 242 different categories of genes in the train data, and the
y are distibuted as follows
```

In [278]:
```
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 [279]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [280]:
```
#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", tra
in_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 [281]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
```

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

In [282]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [283]:
```
train_df['Gene'].head()
```

Out[283]:
```
1412     FGFR3
21         CBL
504       TP53
2126     CCND1
```

```
884     PDGFRA
Name: Gene, dtype: object
```

In [284]:
```
gene_vectorizer.get_feature_names()
```

Out[284]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
```

```
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgfr1',
```

```
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
```

```
'mapk1',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
'myod1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
```

```
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'stag2',
'stat3',
'stk11',
'tert',
```

```
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [285]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 242)
```

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

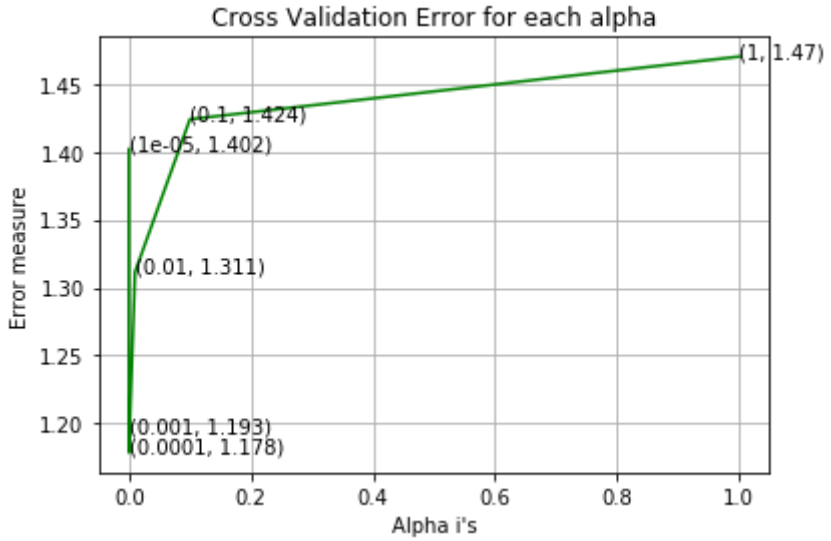
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [286]:
```
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_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 Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, 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_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",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_loss(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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.4019625605579298
For values of alpha =  0.0001 The log loss is: 1.178390947518176
For values of alpha =  0.001 The log loss is: 1.1932544784269383
For values of alpha =  0.01 The log loss is: 1.3114440889211676
For values of alpha =  0.1 The log loss is: 1.424005362413235
For values of alpha =  1 The log loss is: 1.4701532358158753
```




```
For values of best alpha =  0.0001 The train log loss is: 1.060454362813650
2
For values of best alpha =  0.0001 The cross validation log loss is: 1.1783
90947518176
For values of best alpha =  0.0001 The test log loss is: 1.192214262109507
```

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 [287]:
```
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], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  242  g
enes in train dataset?
Ans
1. In test data 655 out of 665 : 98.49624060150376
2. In cross validation data 518 out of  532 : 97.36842105263158
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

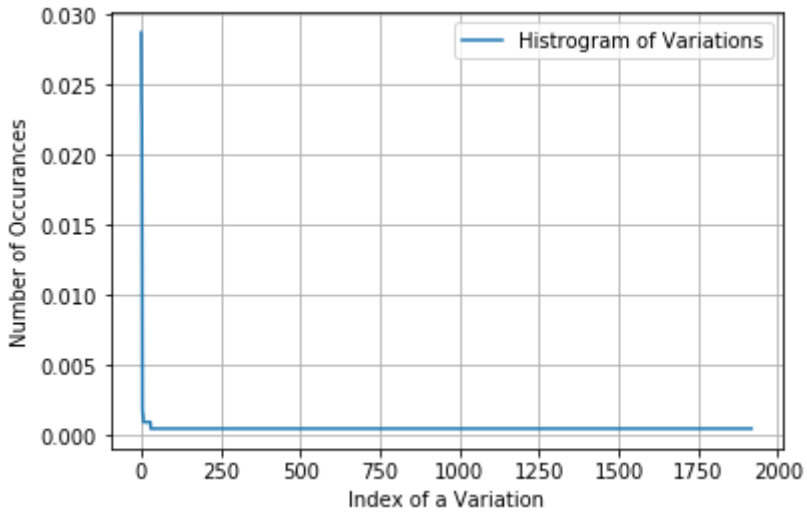
In [288]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1918
Truncating_Mutations    61
Amplification           50
Deletion                48
Fusions                 21
Overexpression           4
Q61H                     3
Q61R                     3
G12V                     2
G35R                     2
E330K                    2
Name: Variation, dtype: int64
```

In [289]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

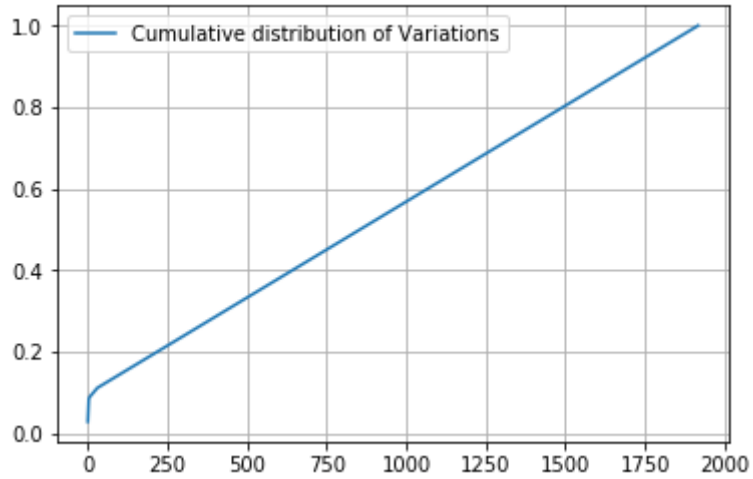
```
Ans: There are 1918 different categories of variations in the train data, a
nd they are distibuted as follows
```

In [290]:
```
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 [291]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.0287194  0.05225989 0.07485876 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [292]:
```
# 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 [293]:
```
print("train_variation_feature_responseCoding is a converted feature using the  response coding method. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

```
train_variation_feature_responseCoding is a converted feature using the res
```

```
ponse coding method. The shape of Variation feature: (2124, 9)
```

In [294]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(trai
n_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df[
'Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Vari
ation'])
```

In [295]:
```
print("train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature:", train_variation_feat
ure_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-h
ot encoding method. The shape of Variation feature: (2124, 1947)
```

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

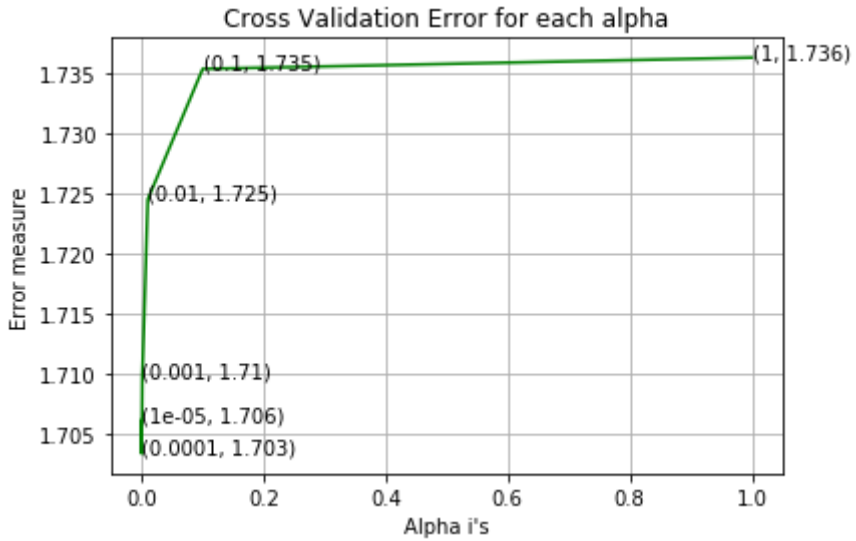
Let's build a model just like the earlier!

In [296]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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_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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, 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 i
s:",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_loss(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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7061033297245032
For values of alpha =  0.0001 The log loss is: 1.7033279641939136
For values of alpha =  0.001 The log loss is: 1.7096735602184494
For values of alpha =  0.01 The log loss is: 1.7245401724431388
```

```
For values of alpha =  0.1 The log loss is: 1.735425941378018
For values of alpha =  1 The log loss is: 1.7363709412512816
```




```
For values of best alpha =  0.0001 The train log loss is: 0.799892940719126
For values of best alpha =  0.0001 The cross validation log loss is: 1.7033
279641939136
For values of best alpha =  0.0001 The test log loss is: 1.7045757654046538
```

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 [297]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[0], " genes in test 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']))))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1918  genes in test and cro
ss validation data sets?
Ans
```

```
1. In test data 62 out of 665 : 9.323308270676693
2. In cross validation data 54 out of  532 : 10.150375939849624
```

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 [298]:
```
# 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 [299]:
```
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_di
ct.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
```

In [300]:
```
# building a TfidfVectorizer with all the words that occured minimum 3 times i
```

```
n train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# 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 return
s (1*number of features) 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 [301]:
```
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 [302]:
```
#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 [303]:
```
# 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_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [304]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=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, axi
s=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 [305]:
```
#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 [306]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({254.92485564185105: 1, 179.6743366077026: 1, 138.48643727952754:
1, 129.80535245216572: 1, 128.96575874866505: 1, 119.7414576102236: 1, 119.
3522720822652: 1, 115.01476649092555: 1, 111.76421625142224: 1, 109.5617425
1861837: 1, 105.76516641704632: 1, 90.55426332692564: 1, 89.52704545646593:
1, 88.9021199890866: 1, 84.94280731801689: 1, 81.66471897938335: 1, 80.5535
4104750046: 1, 80.18581330223554: 1, 79.45409353776236: 1, 75.3940276024889
6: 1, 75.11643642699688: 1, 75.08425737095087: 1, 71.05099875888212: 1, 68.
8517708320724: 1, 68.74832287501928: 1, 67.4351502441337: 1, 66.26497134486
25: 1, 66.1604432175213: 1, 64.64542410454514: 1, 64.516623063368: 1, 64.43
```

174995244321: 1, 63.519629816470804: 1, 62.940623589195624: 1, 60.489182507
91442: 1, 59.32698200783694: 1, 57.511612143086126: 1, 56.321345852853554:
1, 56.216973923270736: 1, 54.552294218017856: 1, 54.439117161411474: 1, 50.
70730809654946: 1, 49.557820346795154: 1, 49.383113608295645: 1, 49.0016117
17154674: 1, 48.42991228180798: 1, 48.23485103488938: 1, 48.22645872656446
6: 1, 46.34627359000825: 1, 46.262150699962646: 1, 45.028466083698: 1, 44.5
0598663581978: 1, 43.57949679729767: 1, 43.52860726352477: 1, 43.4740469429
5213: 1, 43.424515813504996: 1, 43.17842428586127: 1, 42.76092293218395: 1,
42.68149798663714: 1, 42.33624327888522: 1, 42.17326616916091: 1, 42.124955
794747514: 1, 41.62745353422218: 1, 41.41720665335359: 1, 41.0906331938435:
1, 40.99655066806379: 1, 40.93837459442041: 1, 40.34314408360434: 1, 40.046
032357463226: 1, 39.94721838853357: 1, 39.90548998438608: 1, 39.68820125581
4526: 1, 39.06907174726166: 1, 38.76312671991696: 1, 38.23536850861633: 1,
38.084326622511746: 1, 37.42360190054583: 1, 36.099720904045434: 1, 36.0171
5567164854: 1, 35.76838092434907: 1, 35.537566630957535: 1, 35.375371219716
72: 1, 35.275796331620846: 1, 35.20640684821937: 1, 35.144067545859095: 1,
34.87144779171174: 1, 34.86997548583771: 1, 34.601093705545665: 1, 33.79829
843768214: 1, 33.78892341023188: 1, 33.58419730836551: 1, 33.34146966346670
4: 1, 33.259387355602286: 1, 33.222530521638326: 1, 33.11679828432418: 1, 3
2.54441140945396: 1, 32.36954713631926: 1, 32.323087720027715: 1, 32.132880
6986928: 1, 32.07914799660596: 1, 31.894720021344842: 1, 31.79894935124427
5: 1, 31.603760520869272: 1, 31.585146968196923: 1, 31.584948668111487: 1,
31.542205872617192: 1, 31.3054838466527: 1, 31.193866593317363: 1, 31.17476
1262133078: 1, 31.048824429043474: 1, 30.86247280910974: 1, 30.688696154698
338: 1, 30.56667225071931: 1, 30.547677754891463: 1, 30.438054601236935: 1,
30.43520878110456: 1, 30.316703421526405: 1, 30.306004598731718: 1, 30.1019
43784689663: 1, 29.97116850357552: 1, 29.732574987899305: 1, 29.66102197736
163: 1, 29.577242923940634: 1, 29.143242835739695: 1, 29.115033932959175:
1, 29.081853905496963: 1, 29.011487802636086: 1, 28.934305294895374: 1, 28.
76480472266142: 1, 28.759385833166935: 1, 28.6488614423203: 1, 28.346775822
682762: 1, 28.145690618915836: 1, 28.01966357503529: 1, 27.962091102352527:
1, 27.74904859923301: 1, 27.738763520423138: 1, 27.73179503164765: 1, 27.65
9162312119417: 1, 27.657023150218805: 1, 27.650683009571566: 1, 27.15686112
597695: 1, 27.0456036934815: 1, 27.038058371607278: 1, 26.83459802898503:
1, 26.68120578825564: 1, 26.5372221001184: 1, 26.087849425107297: 1, 25.966
588230444554: 1, 25.75912856734232: 1, 25.75566618860182: 1, 25.63336307452
7498: 1, 25.5987287598036: 1, 25.55258633831879: 1, 25.500576389632904: 1,
25.471256869644115: 1, 25.439123344921583: 1, 25.22847660550384: 1, 25.1479
14879489296: 1, 24.9428138655505: 1, 24.67149022619006: 1, 24.6281631699416
58: 1, 24.557936127217822: 1, 24.55327667914324: 1, 24.54433972091654: 1, 2
4.32897555489251: 1, 24.302758926303117: 1, 24.259133186648093: 1, 24.16876
3550799664: 1, 24.151353778585083: 1, 24.04757981409335: 1, 23.951302820322
31: 1, 23.949098820753466: 1, 23.888237760972128: 1, 23.882340591894163: 1,
23.87888550517118: 1, 23.867802929114564: 1, 23.825195157685414: 1, 23.6391
55313191086: 1, 23.604568972864822: 1, 23.57296174703189: 1, 23.54885250359
238: 1, 23.494892103475372: 1, 23.40497959472797: 1, 23.150263169996073: 1,

23.104708013171948: 1, 23.07469753208968: 1, 23.0149640584942: 1, 23.001855
254536423: 1, 22.961364088659735: 1, 22.878828557416412: 1, 22.850004733567
13: 1, 22.835965303435767: 1, 22.767957922325753: 1, 22.76603661391992: 1,
22.727219058639914: 1, 22.49710843342844: 1, 22.495667972911846: 1, 22.4724
06269183754: 1, 22.32491484707407: 1, 22.276229755744563: 1, 22.22607082144
7138: 1, 22.204650550117616: 1, 22.181594960175634: 1, 22.14802759034568:
1, 22.03246881926004: 1, 21.949523998480064: 1, 21.945510852274204: 1, 21.9
42814764830494: 1, 21.941724176436: 1, 21.812054654263743: 1, 21.7400921065
4213: 1, 21.698422754063934: 1, 21.605158632174604: 1, 21.602035782361984:
1, 21.560528458829783: 1, 21.496391987854555: 1, 21.487569394615083: 1, 21.
480948275983664: 1, 21.472701901021107: 1, 21.43501166976304: 1, 21.4346217
55496376: 1, 21.30748292097423: 1, 21.251991949820273: 1, 21.2480836155974
8: 1, 21.212803772372077: 1, 21.170313699488634: 1, 21.134157959190297: 1,
21.087105533494213: 1, 21.020396324122764: 1, 21.016231108289038: 1, 20.907
001961418658: 1, 20.856073702733713: 1, 20.805932339237557: 1, 20.797833987
291984: 1, 20.616541656535677: 1, 20.543743285313305: 1, 20.51863855908486:
1, 20.511168327124917: 1, 20.505856776043192: 1, 20.425201607527516: 1, 20.
40698802130536: 1, 20.35061799124216: 1, 20.302146765176055: 1, 20.20373676
188306: 1, 20.18412580399013: 1, 20.160293831728726: 1, 20.13151242362394:
1, 20.096128480983428: 1, 20.0535136569305: 1, 19.99895689080971: 1, 19.947
240990736333: 1, 19.946788461675638: 1, 19.934124107651964: 1, 19.928843451
377283: 1, 19.910636296723776: 1, 19.90074294315914: 1, 19.848724514238697:
1, 19.848385160903387: 1, 19.757731453099854: 1, 19.73970505190609: 1, 19.6
7039076997222: 1, 19.644947076773732: 1, 19.418748892507157: 1, 19.37118549
218242: 1, 19.358119513769473: 1, 19.352541379805395: 1, 19.25977292654748:
1, 19.25320178007242: 1, 19.22340759306734: 1, 19.219077680957813: 1, 19.18
978231606735: 1, 19.17807112313711: 1, 19.10766504401224: 1, 19.10063618858
288: 1, 19.05782439729586: 1, 18.936234355444878: 1, 18.883917710694206: 1,
18.827209180253373: 1, 18.787837519372278: 1, 18.774634219025188: 1, 18.765
407519491287: 1, 18.739674465065224: 1, 18.721255765128426: 1, 18.716730586
808062: 1, 18.707323885519457: 1, 18.668833615695394: 1, 18.63148095219187
2: 1, 18.6021160907659: 1, 18.589743831142115: 1, 18.532831860595458: 1, 1
8.488393592902483: 1, 18.44392606447327: 1, 18.424955649720054: 1, 18.37765
264313294: 1, 18.327358909078836: 1, 18.312945137784457: 1, 18.267669315257
57: 1, 18.258958047433058: 1, 18.250084301928865: 1, 18.244506445339383: 1,
18.204750065539688: 1, 18.18980148775524: 1, 18.179646814870768: 1, 18.1247
36033248475: 1, 17.928327352330292: 1, 17.91228601318607: 1, 17.90332755471
902: 1, 17.856785538885187: 1, 17.77569456652335: 1, 17.756694647500545: 1,
17.75011513466527: 1, 17.73919790967558: 1, 17.595946204736382: 1, 17.56535
5507207713: 1, 17.565079592742002: 1, 17.514315788741847: 1, 17.50676818292
5793: 1, 17.502855664659585: 1, 17.46371472219291: 1, 17.405352003720097:
1, 17.379513315743136: 1, 17.33427503182233: 1, 17.307341067607414: 1, 17.2
31661889090777: 1, 17.22154380964408: 1, 17.22077719741894: 1, 17.206535516
8085: 1, 17.18304382604871: 1, 17.154115438375122: 1, 17.143785243275218:
1, 17.134642702020198: 1, 17.12430721389193: 1, 17.08744395033757: 1, 17.08
6239498968595: 1, 17.052825182051123: 1, 17.03172256196798: 1, 16.960321814

621203: 1, 16.9155629910071 14: 1, 16.887154024442005: 1, 16.86968931805576
6: 1, 16.846489342527583: 1, 16.829388370186532: 1, 16.819855633272855: 1,
16.80136190624529: 1, 16.799337424709524: 1, 16.798031512202826: 1, 16.7756
20167559662: 1, 16.682352507582635: 1, 16.67068800059197: 1, 16.66377576003
4053: 1, 16.661737888848045: 1, 16.653330346409454: 1, 16.6518448295963: 1,
16.61487983688525: 1, 16.59428037518838: 1, 16.5616240008025: 1, 16.5363838
46389228: 1, 16.50259262587916: 1, 16.486025079930283: 1, 16.47213979288630
6: 1, 16.471666225581075: 1, 16.418402853175746: 1, 16.41697376811895: 1, 1
6.40808700503638: 1, 16.394099993594683: 1, 16.379098753330243: 1, 16.34415
071875832: 1, 16.325250000569895: 1, 16.31596568755016: 1, 16.2949043090035
2: 1, 16.234131163882253: 1, 16.19027427424369: 1, 16.174826790777153: 1, 1
6.163702634126494: 1, 16.10888788206295: 1, 16.096396646779: 1, 16.07379403
202596: 1, 16.071675376175975: 1, 16.02572547157654: 1, 15.911931438212326:
1, 15.878992604529724: 1, 15.877970056123377: 1, 15.87575706879011: 1, 15.8
09336745023092: 1, 15.785724256574927: 1, 15.783255467063583: 1, 15.7104891
02012342: 1, 15.709141290491674: 1, 15.702807025005532: 1, 15.6883778352191
25: 1, 15.669547627254017: 1, 15.662829939478229: 1, 15.64405112137133: 1,
15.472364646363074: 1, 15.40979812719529: 1, 15.403295210595395: 1, 15.3834
10737238107: 1, 15.355028338056876: 1, 15.323511110564393: 1, 15.3180138988
48548: 1, 15.283635675872173: 1, 15.230608120733448: 1, 15.223917734954874:
1, 15.217950460869448: 1, 15.177719208528147: 1, 15.137543276911323: 1, 15.
065177478189291: 1, 15.041183860749642: 1, 15.01251457966047: 1, 14.9629323
46259825: 1, 14.955337012775276: 1, 14.9498877518602: 1, 14.93217586672216
9: 1, 14.91282203887667: 1, 14.908073452465292: 1, 14.884363998720879: 1, 1
4.879731268641994: 1, 14.877818147672015: 1, 14.877026142913808: 1, 14.8393
69661142507: 1, 14.80034829971488: 1, 14.78986674039232: 1, 14.787612281656
26: 1, 14.757734815031442: 1, 14.724537933069493: 1, 14.723104227226928: 1,
14.716048619778338: 1, 14.707159068501893: 1, 14.674634891821672: 1, 14.664
040701967886: 1, 14.642662314365184: 1, 14.61449426840746: 1, 14.5987100242
77473: 1, 14.549832742218324: 1, 14.545671454732787: 1, 14.53159945269426:
1, 14.525307351408465: 1, 14.525281563092728: 1, 14.525048581252696: 1, 14.
51497702751115: 1, 14.506080490071973: 1, 14.443053565178818: 1, 14.3872788
09487144: 1, 14.373137675070359: 1, 14.360598746143209: 1, 14.3470744703265
22: 1, 14.32857477663188: 1, 14.322716307540418: 1, 14.279073923439967: 1,
14.26377762315584: 1, 14.263742901378999: 1, 14.232482056893708: 1, 14.2088
64080729033: 1, 14.205910125540967: 1, 14.193333712648133: 1, 14.1829928321
55632: 1, 14.11563317473164: 1, 14.096828149628589: 1, 14.095453701012454:
1, 14.078330585994427: 1, 14.075395042685871: 1, 14.05083548125905: 1, 14.0
10744381843507: 1, 14.001419014223481: 1, 13.958228267198765: 1, 13.9438209
01196348: 1, 13.933104296622549: 1, 13.927302871802903: 1, 13.9177259419060
12: 1, 13.8819403036504: 1, 13.864013690821322: 1, 13.837852537119765: 1, 1
3.835380881278631: 1, 13.828942126898497: 1, 13.819685889402017: 1, 13.7734
85946611151: 1, 13.734373866770154: 1, 13.714429134395509: 1, 13.6917594581
88637: 1, 13.63508736154815: 1, 13.617678471090544: 1, 13.576041023409722:
1, 13.572138795129234: 1, 13.565892621526594: 1, 13.514696105307442: 1, 13.
499633411819229: 1, 13.48402002641801: 1, 13.466998900286052: 1, 13.4579707

35063464: 1, 13.4264138620929: 1, 13.415553913969037: 1, 13.41507348768072
4: 1, 13.395506315978176: 1, 13.360525836236674: 1, 13.357702917162786: 1,
13.354819882372286: 1, 13.345246808084333: 1, 13.32289239125021: 1, 13.2853
42948269461: 1, 13.278635762133545: 1, 13.26616689594293: 1, 13.23748457297
7552: 1, 13.222282776395295: 1, 13.170902532894615: 1, 13.07401563923002:
1, 12.985449538486414: 1, 12.981759501138638: 1, 12.968055935198988: 1, 12.
965172194817967: 1, 12.918706658479785: 1, 12.904987798247754: 1, 12.891592
556099484: 1, 12.880324979546353: 1, 12.870292929596774: 1, 12.846123252958
405: 1, 12.771633604947626: 1, 12.76873329332741: 1, 12.760537935836046: 1,
12.746010782942028: 1, 12.745019325790402: 1, 12.736806743501866: 1, 12.734
345691861801: 1, 12.727433587801038: 1, 12.709278005335516: 1, 12.635452690
225083: 1, 12.624285309890166: 1, 12.604246697736313: 1, 12.59007615582016:
1, 12.571428207499974: 1, 12.558139585587334: 1, 12.55205821481785: 1, 12.5
51403399514802: 1, 12.5434361929376: 1, 12.529095171955067: 1, 12.525326131
412976: 1, 12.51499966944245: 1, 12.493445247059649: 1, 12.492524977210524:
1, 12.461394362220641: 1, 12.420521298247289: 1, 12.410255799274234: 1, 12.
37538643996547: 1, 12.364378337105899: 1, 12.360769710562025: 1, 12.3570657
62191837: 1, 12.350649198794422: 1, 12.347653232924127: 1, 12.3024365159814
23: 1, 12.2933833817331: 1, 12.27970402820999: 1, 12.279203699824532: 1, 1
2.269859812024977: 1, 12.232583720719166: 1, 12.18325322573175: 1, 12.16964
7619123307: 1, 12.145703821210923: 1, 12.11926929699875: 1, 12.079997435113
224: 1, 12.079840965591593: 1, 12.046078990623476: 1, 12.04345673711617: 1,
12.028987751473657: 1, 12.021834261947056: 1, 12.017367619512978: 1, 11.950
36287662181: 1, 11.939154822823584: 1, 11.93284762358763: 1, 11.92307719623
571: 1, 11.92153227298804: 1, 11.908934950308367: 1, 11.901084971387482: 1,
11.876658227947814: 1, 11.840243983274867: 1, 11.823568662804023: 1, 11.818
348250497866: 1, 11.775049609963355: 1, 11.771618347218393: 1, 11.760973630
906845: 1, 11.755940553588717: 1, 11.754981862258877: 1, 11.74831105844252
7: 1, 11.726789290008286: 1, 11.675267461790925: 1, 11.675159152650112: 1,
11.6632131612883: 1, 11.63339149142656: 1, 11.624641979844862: 1, 11.621963
505433548: 1, 11.620732083971266: 1, 11.594315498977801: 1, 11.587551685058
957: 1, 11.577272063648968: 1, 11.568822337615854: 1, 11.558012709498199:
1, 11.550301235083412: 1, 11.537975115349916: 1, 11.532240102916159: 1, 11.
515432238192192: 1, 11.48946997237123: 1, 11.471514643235643: 1, 11.4664032
51488494: 1, 11.464430579057863: 1, 11.444636890487196: 1, 11.4430697125073
54: 1, 11.427368274836518: 1, 11.415776573256505: 1, 11.411778134038673: 1,
11.405113543846188: 1, 11.357858747737039: 1, 11.307970520139088: 1, 11.304
471240692068: 1, 11.296865496762855: 1, 11.294136511712086: 1, 11.291276618
853436: 1, 11.248330064373215: 1, 11.236373715533858: 1, 11.21681139124769
2: 1, 11.216631740960308: 1, 11.213250642008749: 1, 11.187618033414635: 1,
11.187026840297445: 1, 11.176501887153426: 1, 11.157625336204543: 1, 11.140
29636412567: 1, 11.135152142086746: 1, 11.127686408883232: 1, 11.1260674943
99257: 1, 11.111188842069017: 1, 11.0891985739184: 1, 11.084660763831707:
1, 11.079527959309582: 1, 11.077846467695842: 1, 11.052010617138635: 1, 11.
041424472137516: 1, 11.032178282425674: 1, 11.022509038709758: 1, 10.989599
073555949: 1, 10.979375948242083: 1, 10.971941839292972: 1, 10.962638352095

347: 1, 10.95694912855443: 1, 10.936042108809238: 1, 10.930329725491891: 1,
10.910947045084098: 1, 10.879589100789525: 1, 10.868951621110016: 1, 10.857
382124106326: 1, 10.85097375620149: 1, 10.842949873701317: 1, 10.8416021606
26492: 1, 10.818151934818443: 1, 10.808789955232289: 1, 10.80607295686753:
1, 10.777432984390309: 1, 10.767721199490941: 1, 10.765868197745545: 1, 10.
764961788391288: 1, 10.763654831694348: 1, 10.7436270510075: 1, 10.72366348
008869: 1, 10.716927337643689: 1, 10.713259660169303: 1, 10.71020642258158
5: 1, 10.705486512758648: 1, 10.700941892953177: 1, 10.687603081396272: 1,
10.653839643231453: 1, 10.626369035970265: 1, 10.615257468232299: 1, 10.582
576221117213: 1, 10.561301583730438: 1, 10.554489293102444: 1, 10.536948935
585409: 1, 10.52650270256909: 1, 10.521160019254397: 1, 10.49906666443971:
1, 10.47396076493444: 1, 10.473702584352496: 1, 10.466708643829005: 1, 10.4
65608621330794: 1, 10.45213714030905: 1, 10.441847118655144: 1, 10.42053587
5490462: 1, 10.419898994008678: 1, 10.408207859137926: 1, 10.39146940759848
2: 1, 10.374322895813467: 1, 10.366308664550877: 1, 10.3427755095869: 1, 1
0.330914173170187: 1, 10.307680053241763: 1, 10.29560999020188: 1, 10.29450
5358133367: 1, 10.264564838316748: 1, 10.255069332163147: 1, 10.23414703391
2656: 1, 10.217746261824603: 1, 10.214957347997377: 1, 10.202758973947606:
1, 10.181992135609326: 1, 10.170597715187979: 1, 10.164595210101405: 1, 10.
126886236907641: 1, 10.104911732369274: 1, 10.104475088162847: 1, 10.098732
399154358: 1, 10.069294610435636: 1, 10.053671327797373: 1, 10.046574641989
798: 1, 10.042275770898716: 1, 10.040180975006733: 1, 10.023101133102355:
1, 10.009767944421348: 1, 10.006585138912754: 1, 9.990262728285659: 1, 9.98
5168458611406: 1, 9.954787280015433: 1, 9.952209287240773: 1, 9.93568984685
3393: 1, 9.931191640582504: 1, 9.924310694243818: 1, 9.906834759602031: 1,
9.898576344067898: 1, 9.869644310787114: 1, 9.865864655443795: 1, 9.8424637
7357499: 1, 9.84099390884438: 1, 9.837298334596197: 1, 9.836727763669057:
1, 9.801878369669021: 1, 9.801049320351751: 1, 9.782378532885085: 1, 9.7659
0408011598: 1, 9.758179491900503: 1, 9.74686174428267: 1, 9.72901944281047
9: 1, 9.728335722635576: 1, 9.719132754975368: 1, 9.707734843391504: 1, 9.7
01524151831968: 1, 9.699307890299757: 1, 9.693642792382862: 1, 9.6793004451
69309: 1, 9.665409543132993: 1, 9.661212010243228: 1, 9.652417505085245: 1,
9.644202746420627: 1, 9.635165714430268: 1, 9.619429207910636: 1, 9.6174038
5852346: 1, 9.613598749474363: 1, 9.60698201068276: 1, 9.592806409763387:
1, 9.578725254217517: 1, 9.576017341700805: 1, 9.555927200535757: 1, 9.5485
03157947486: 1, 9.537851756772456: 1, 9.537234660146998: 1, 9.5336297590226
8: 1, 9.530949484819748: 1, 9.526813024392276: 1, 9.515741845846799: 1, 9.5
12231831423188: 1, 9.50086646479669: 1, 9.499771393274223: 1, 9.48370067301
1628: 1, 9.477109556212667: 1, 9.473152281211027: 1, 9.4696931441501: 1, 9.
466960753110303: 1, 9.466337535153095: 1, 9.445804849884837: 1, 9.436166776
469642: 1, 9.4205884057132: 1, 9.412515894152873: 1, 9.412436171886068: 1,
9.409355170097927: 1, 9.405616067368047: 1, 9.401556800633918: 1, 9.4012524
77023025: 1, 9.395066038257417: 1, 9.38927155607966: 1, 9.3812241787546: 1,
9.377297679479161: 1, 9.377033551845058: 1, 9.375137963004208: 1, 9.3670487
4772864: 1, 9.364368777661824: 1, 9.360629269284322: 1, 9.356227499856095:
1, 9.355182251890165: 1, 9.329115143886126: 1, 9.325475436009022: 1, 9.3196

8917622846: 1, 9.305233581962906: 1, 9.287332916401148: 1, 9.272048078621868: 1, 9.268990674620946: 1, 9.234164989788047: 1, 9.230310311970731: 1, 9.216130858200655: 1, 9.215721466482599: 1, 9.210959596063512: 1, 9.17770506267047: 1, 9.13224488856717: 1, 9.128259293274272: 1, 9.11166632743824: 1, 9.109182600146221: 1, 9.107551894571946: 1, 9.088368818477381: 1, 9.084702090751621: 1, 9.063351639833417: 1, 9.046186303608152: 1, 9.01632466903311: 1, 9.00986706768968: 1, 9.003616242432845: 1, 9.002243474955405: 1, 9.00036305763591: 1, 9.00023299436: 1, 8.997244700272745: 1, 8.975783632986412: 1, 8.975566326443202: 1, 8.970027286006298: 1, 8.935309631182855: 1, 8.930150979046235: 1, 8.927728678800246: 1, 8.902521043198336: 1, 8.896113606085537: 1, 8.891500477733205: 1, 8.883732530393685: 1, 8.862059198849668: 1, 8.853988525805626: 1, 8.836706759011797: 1, 8.829132662718806: 1, 8.822861719219956: 1, 8.799420538982483: 1, 8.790557381263318: 1, 8.763339522880347: 1, 8.755933244795994: 1, 8.737811591240773: 1, 8.729776403285562: 1, 8.727240799775442: 1, 8.716456811641851: 1, 8.706558809906948: 1, 8.696950648708533: 1, 8.678139244416256: 1, 8.676800630029632: 1, 8.670488020242557: 1, 8.66513393186726: 1, 8.663567237102198: 1, 8.663273588332347: 1, 8.662655206579322: 1, 8.661475802952904: 1, 8.658407109705468: 1, 8.657883992803686: 1, 8.650400271113499: 1, 8.645112027678985: 1, 8.638711272599995: 1, 8.633170217523 6: 1, 8.631848292228659: 1, 8.629861509998994: 1, 8.592584847574155: 1, 8.58677987930129: 1, 8.563920927044085: 1, 8.555167223227684: 1, 8.511798004703627: 1, 8.502851369811932: 1, 8.475881118613863: 1, 8.475203668762148: 1, 8.473574277938118: 1, 8.435999564140447: 1, 8.429559921000141: 1, 8.418717475686845: 1, 8.410897508840456: 1, 8.410347895810682: 1, 8.406646532767363: 1, 8.40246637408029: 1, 8.40232404582253: 1, 8.40116557066916: 1, 8.388474897606715: 1, 8.382695092310366: 1, 8.351221388812085: 1, 8.342637594030538: 1, 8.308320809529313: 1, 8.300097814557077: 1, 8.294145821605023: 1, 8.278521796029269: 1, 8.265734005863132: 1, 8.23753353327631: 1, 8.232289860485 39: 1, 8.193939750643576: 1, 8.188575084640357: 1, 8.186947635978973: 1, 8.179066503727517: 1, 8.15510962585566: 1, 8.144741049816616: 1, 8.119449051670383: 1, 8.115090775934537: 1, 8.110349392325732: 1, 8.109785083773408: 1, 8.107676963758434: 1, 8.084965972931982: 1, 8.083668508876416: 1, 8.066417572576158: 1, 8.055693008989646: 1, 8.02865838709607: 1, 8.027574091274357: 1, 8.018603413492098: 1, 8.016584238895957: 1, 8.00912801733771: 1, 8.002058972220842: 1, 7.999375076259049: 1, 7.981410561372674: 1, 7.979939049190409: 1, 7.970897576323985: 1, 7.9684959877882635: 1, 7.96772688079394 1: 1, 7.96168098402094: 1, 7.95418728311357: 1, 7.944166760294547: 1, 7.944120232242154: 1, 7.940644389624788: 1, 7.91870517566366: 1, 7.913979156957382: 1, 7.906095555976621: 1, 7.878558890118844: 1, 7.878072545811198: 1, 7.874154350566561: 1, 7.873208085754772: 1, 7.860152756734408: 1, 7.838639424092214: 1, 7.833821012246962: 1, 7.833637204214701: 1, 7.826572012111276: 1, 7.821875291060208: 1, 7.81035788448518: 1, 7.809496455269263: 1, 7.787594437286115: 1, 7.780177883027917: 1, 7.753440116873204: 1, 7.724892183537357: 1, 7.719333023497055: 1, 7.692827012204247: 1, 7.686336310541944: 1, 7.660397730569181: 1, 7.657498667238145: 1, 7.6380129999061515: 1, 7.6355825098773265: 1, 7.617748311539303: 1, 7.589675245131953: 1, 7.564829278593581:

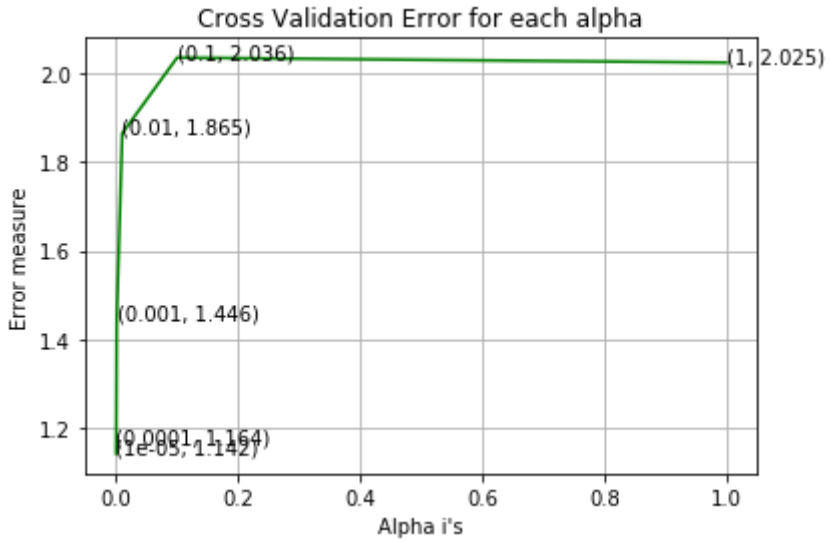
```
1, 7.556309981289133: 1, 7.549387164394647: 1, 7.522057533464738: 1, 7.5166
36473927644: 1, 7.510721736497497: 1, 7.499428699368295: 1, 7.4893011459390
7: 1, 7.484337341500697: 1, 7.476753786368227: 1, 7.449205845152545: 1, 7.4
31785571412563: 1, 7.42722854007254: 1, 7.423777855042161: 1, 7.40679745423
9366: 1, 7.375567161650679: 1, 7.353570689716014: 1, 7.288312576910132: 1,
7.264785154220272: 1, 7.2442955774297975: 1, 7.232861136521713: 1, 7.162686
19041033: 1, 7.157466571032554: 1, 7.146806334795322: 1, 7.133371549795108:
1, 7.132089331316596: 1, 7.129877742198015: 1, 7.086379212627761: 1, 7.0439
10550713147: 1, 7.032134664115205: 1, 6.994317316007247: 1, 6.9887681087201
82: 1, 6.987840247152619: 1, 6.938006363384724: 1, 6.933632826161288: 1, 6.
9285150385996275: 1, 6.905307210656533: 1, 6.89454993193391: 1, 6.835305317
594998: 1, 6.770716241626127: 1, 6.757120636584677: 1, 6.730247512639627:
1, 6.714838498140941: 1, 6.605849153146865: 1, 6.594773115300686: 1, 6.5513
61394881983: 1, 6.481999862202481: 1, 6.424517697638972: 1, 6.3236394356705
405: 1})
```

In [307]:

```
# 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/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, 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 i
s:",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_loss(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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.1417877137308243
For values of alpha =  0.0001 The log loss is: 1.1637328605378794
For values of alpha =  0.001 The log loss is: 1.4457432306630014
For values of alpha =  0.01 The log loss is: 1.8651768087409692
For values of alpha =  0.1 The log loss is: 2.03622483427236
For values of alpha =  1 The log loss is: 2.024942550444193
```




```
For values of best alpha =  1e-05 The train log loss is: 0.8079914187375853
For values of best alpha =  1e-05 The cross validation log loss is: 1.14178
77137308243
For values of best alpha =  1e-05 The test log loss is: 1.1500729891599621
```

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

Ans. Yes, it seems like!

In [308]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    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 [309]:
```
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")
```

```
94.7 % of word of test data appeared in train data
94.0 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [310]:
```
#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 bel
ongs 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 [311]:
```
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 [312]:
```
# 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 = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

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

Feature engineering

Here just taking the Gene & variation and combine that in Gene_var_data corpus.
once fit that by using TFIDF transform into Train[TEXT],test[TEXT],cv[TEXT],By doing that It can have some more informations

In [313]:
```
Gene_Var_data=[]
for i in data["Gene"].values:
    Gene_Var_data.append(i)
```

```
for i in data["Variation"].values:
    Gene_Var_data.append(i)
cnt_vect=TfidfVectorizer()
Fit_vect=cnt_vect.fit(Gene_Var_data)
train_text_vect=cnt_vect.transform(train_df["TEXT"])
cv_text_vect=cnt_vect.transform(cv_df["TEXT"])
test_text_vect=cnt_vect.transform(test_df["TEXT"])
train_text_vect=normalize(train_text_vect,axis=0)
cv_text_vect=normalize(cv_text_vect,axis=0)
test_text_vect=normalize(test_text_vect,axis=0)
```

Stacking the three types of features

And also I stacking that with one hot encoding features

In [314]:

```
# 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_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

train_gene_var_updt=hstack((train_gene_var_onehotCoding,train_text_vect))
train_x_onehotCoding = hstack((train_gene_var_updt, train_text_feature_onehotC
oding)).tocsr()
train_y = np.array(list(train_df['Class']))


test_gene_var_updt=hstack((test_gene_var_onehotCoding,test_text_vect))
```

```
test_x_onehotCoding = hstack((test_gene_var_updt, test_text_feature_onehotCodi
ng)).tocsr()
test_y = np.array(list(test_df['Class']))


cv_gene_var_updt=hstack((cv_gene_var_onehotCoding,cv_text_vect))
cv_x_onehotCoding = hstack((cv_gene_var_updt, cv_text_feature_onehotCoding)).t
ocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,t
rain_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,tes
t_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_vari
ation_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_fea
ture_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_r
esponseCoding))
```

In [315]:
```
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_o
nehotCoding.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, 6427)
(number of data points * number of features) in test data =  (665, 6427)
(number of data points * number of features) in cross validation data = (53
2, 6427)
```

In [316]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_r
esponseCoding.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 = (53
2, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

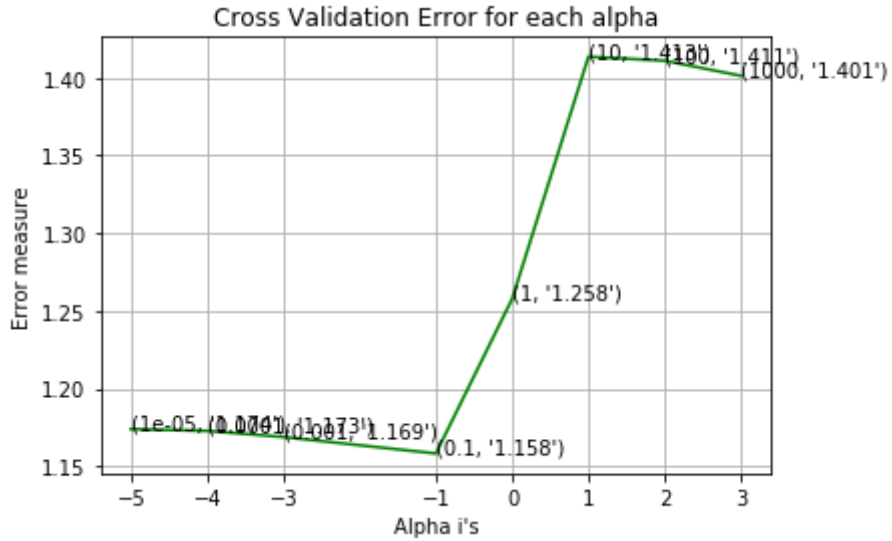
4.1.1.1. Hyper parameter tuning

In [317]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

# 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probab
ility 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 i
```

```
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.1739581761584255
for alpha = 0.0001
Log Loss : 1.1728047951914005
for alpha = 0.001
Log Loss : 1.168864539462507
for alpha = 0.1
Log Loss : 1.158250468831457
for alpha = 1
Log Loss : 1.2577399951875554
for alpha = 10
Log Loss : 1.4134772942015252
for alpha = 100
Log Loss : 1.4109445895566923
for alpha = 1000
Log Loss : 1.4012076802171929
```

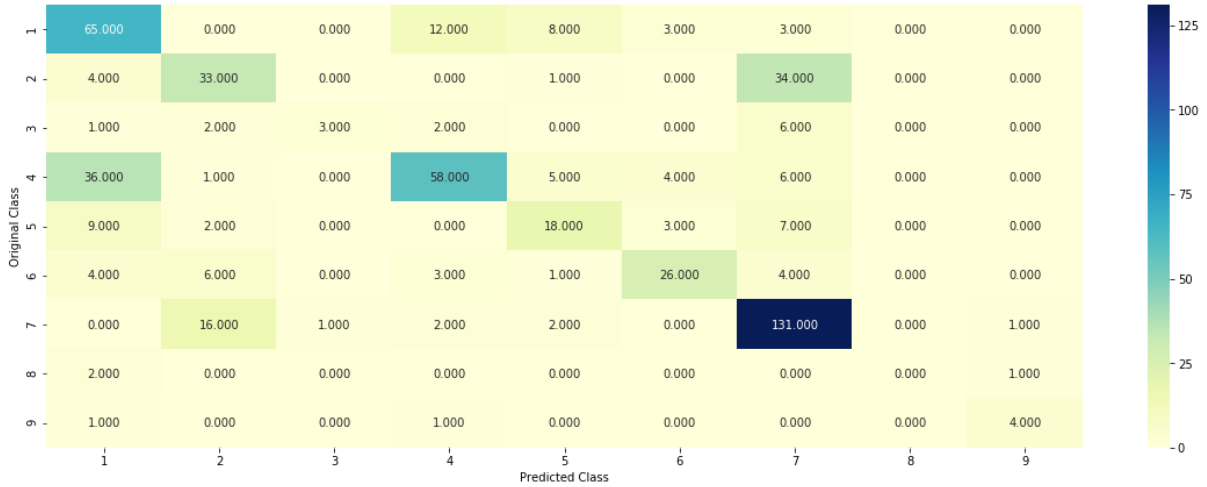



```
For values of best alpha =  0.1 The train log loss is: 0.7606205701319388
For values of best alpha =  0.1 The cross validation log loss is: 1.1582504
68831457
For values of best alpha =  0.1 The test log loss is: 1.1881531434320403
```

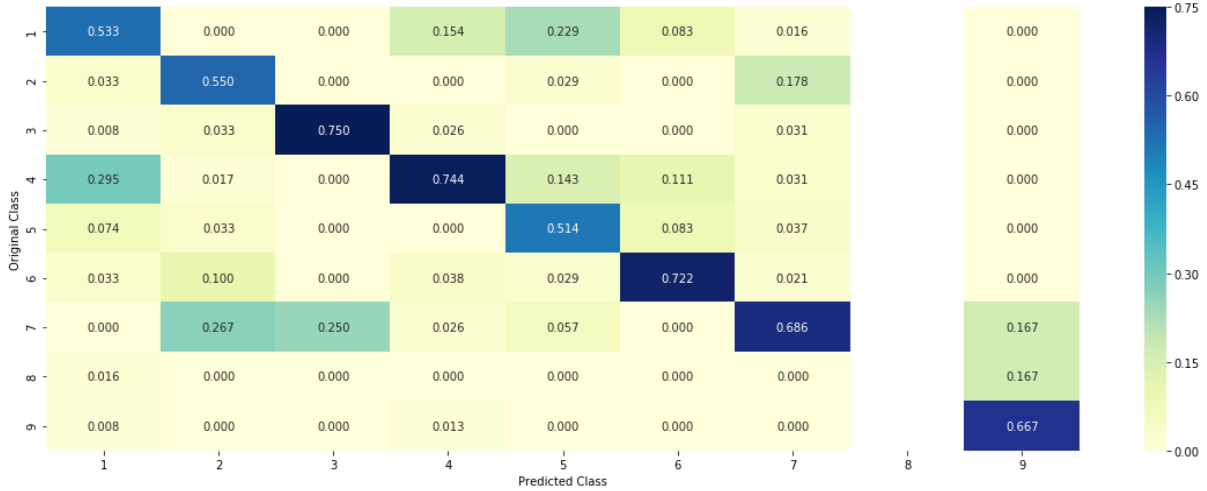
4.1.1.2. Testing the model with best hyper paramters

In [318]:
```
# 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()))
```

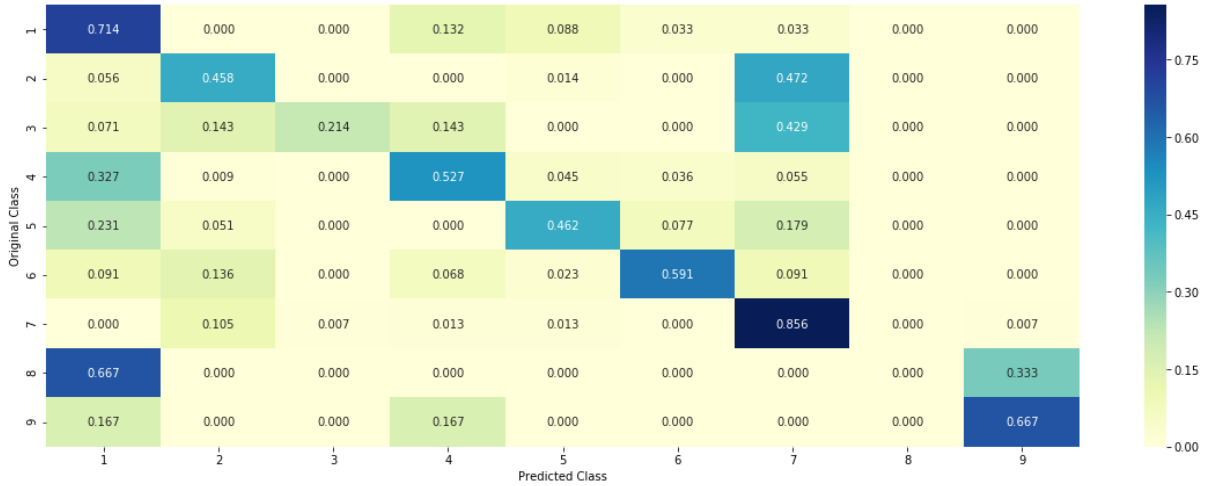
```
Log Loss : 1.158250468831457
Number of missclassified point : 0.36466165413533835
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [319]:
```
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)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.056  0.5863 0.0179 0.0976 0.052  0.0435
0.1346 0.0064 0.0057]]
Actual Class : 2
--------------------------------------------------
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [320]:
```
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)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0472 0.0494 0.0151 0.0824 0.0439 0.0367
0.715  0.0054 0.0048]]
Actual Class : 7
--------------------------------------------------
```

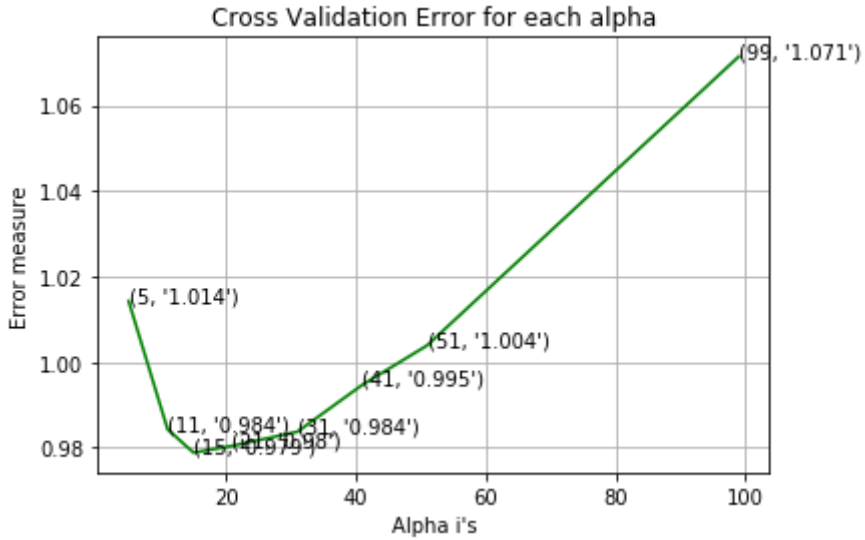
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [321]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_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-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probab
ility 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 i
s:",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_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[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0142427410600647
for alpha = 11
Log Loss : 0.9843009431025068
for alpha = 15
Log Loss : 0.9788393451387649
for alpha = 21
Log Loss : 0.9804274194543124
for alpha = 31
Log Loss : 0.9837139436419055
for alpha = 41
Log Loss : 0.9946640234837321
for alpha = 51
Log Loss : 1.003891538065869
for alpha = 99
Log Loss : 1.071359160401408
```



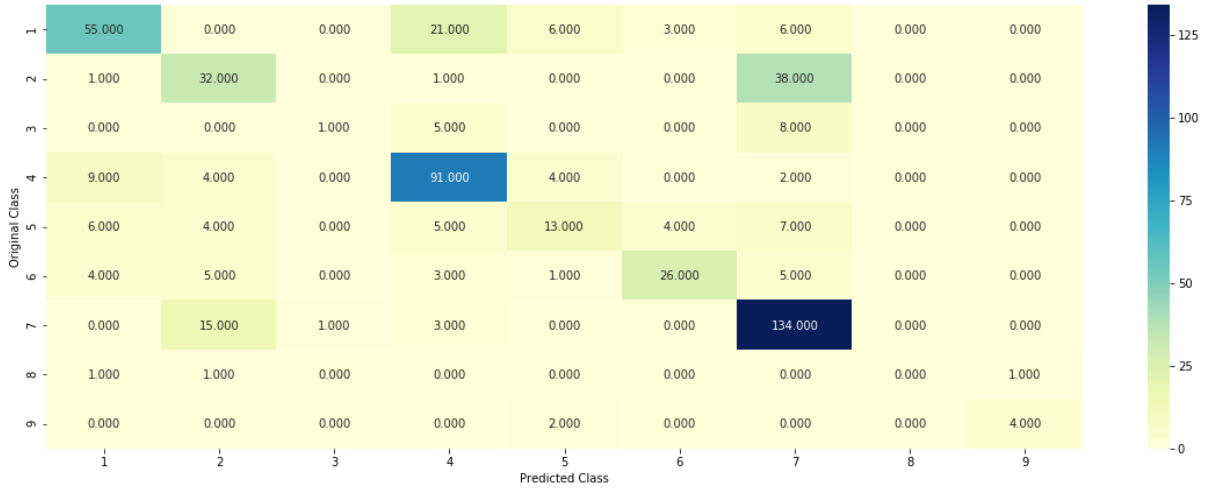

```
For values of best alpha =  15 The train log loss is: 0.7189457900171621
For values of best alpha =  15 The cross validation log loss is: 0.97883934
51387649
For values of best alpha =  15 The test log loss is: 1.055113934016944
```

4.2.2. Testing the model with best hyper paramters

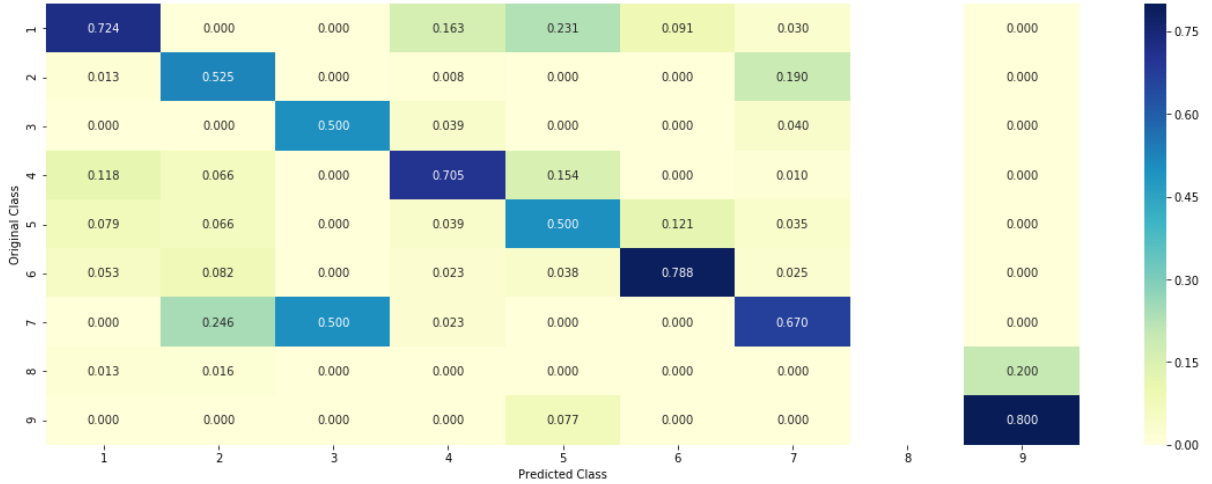
In [322]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
```

```
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_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-neighbors-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_respon
seCoding, cv_y, clf)
```

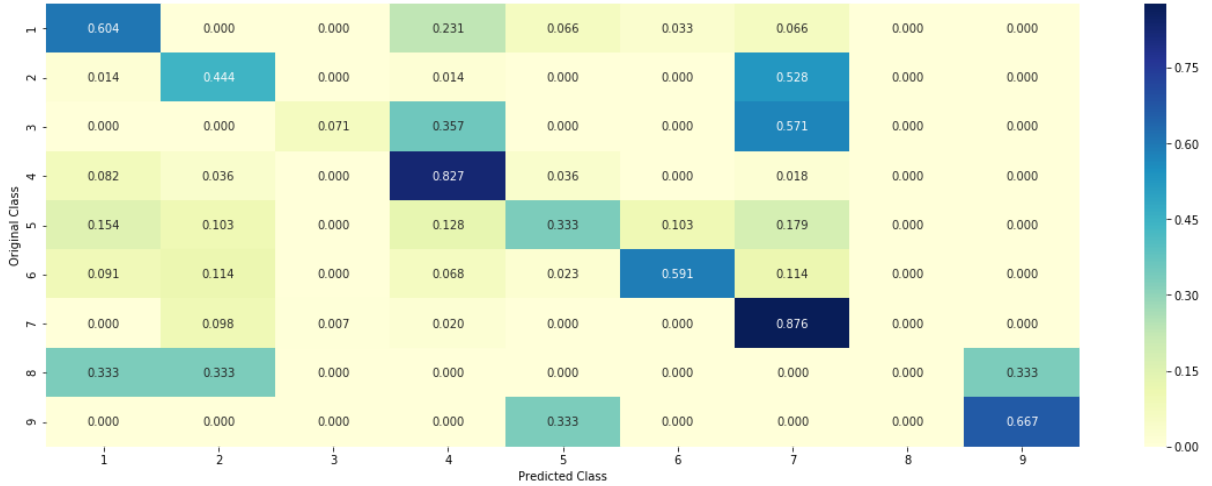
```
Log loss : 0.9788393451387649
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [323]:

```
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 : 2
The  15  nearest neighbours of the test points belongs to classes [2 2 2 2
7 2 2 2 2 2 2 2 2 2 2]
Fequency of nearest points : Counter({2: 14, 7: 1})
```

4.2.4. Sample Query Point-2

In [324]:
```
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].reshap
e(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 o
f the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points bel
ongs to classes [7 7 7 7 7 3 7 7 7 7 7 7 7 7 7]
Fequency of nearest points : Counter({7: 14, 3: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

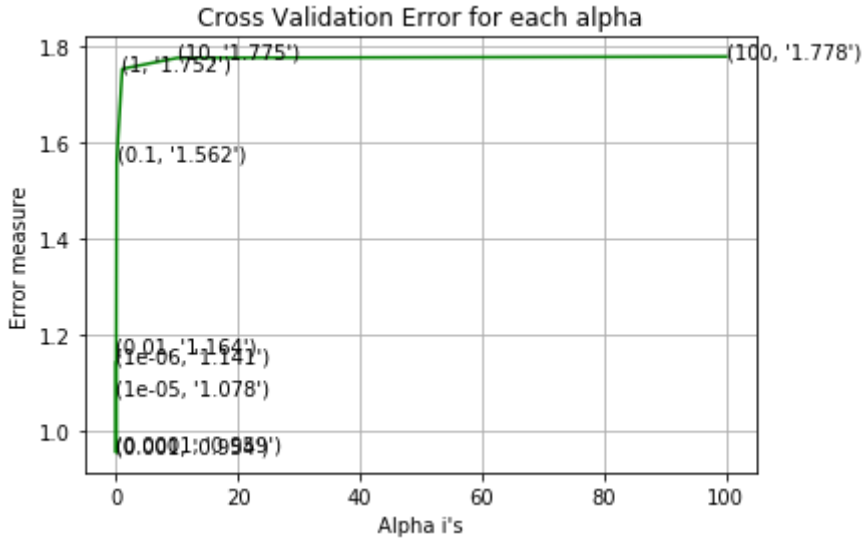
In [325]:

```
# 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/
#-------------------------------


# 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-probab
ility 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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1413576067484803
for alpha = 1e-05
Log Loss : 1.0777443009202097
for alpha = 0.0001
Log Loss : 0.9593629043270478
for alpha = 0.001
Log Loss : 0.9541478702481345
for alpha = 0.01
```

```
Log Loss : 1.1642179922517195
for alpha = 0.1
Log Loss : 1.5618342052704404
for alpha = 1
Log Loss : 1.752104678422237
for alpha = 10
Log Loss : 1.7749643040101175
for alpha = 100
Log Loss : 1.7777065191600168
```



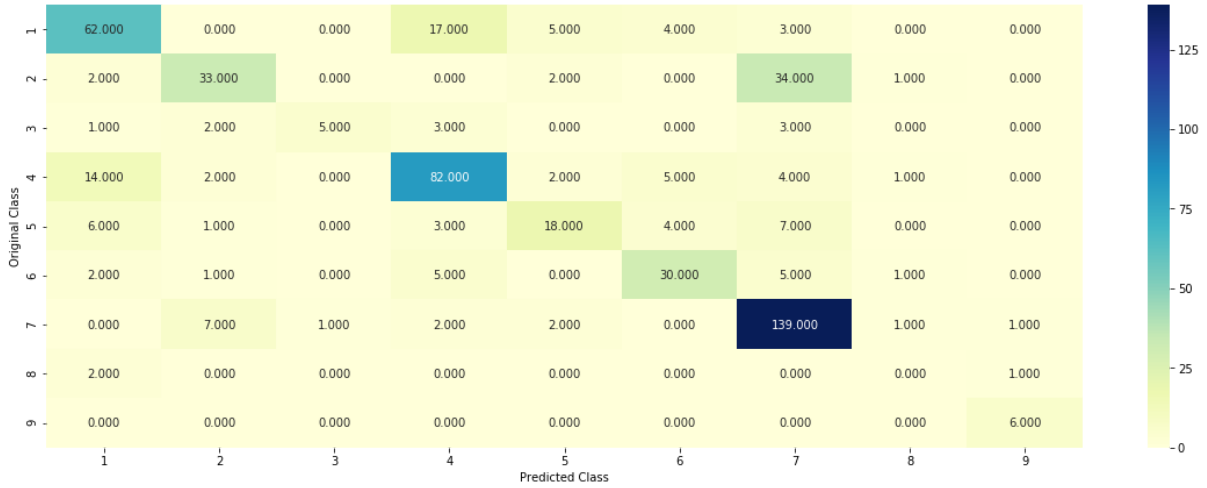

```
For values of best alpha =  0.001 The train log loss is: 0.6483218400933497
For values of best alpha =  0.001 The cross validation log loss is: 0.95414
78702481345
For values of best alpha =  0.001 The test log loss is: 0.9771034733314872
```

4.3.1.2. Testing the model with best hyper paramters

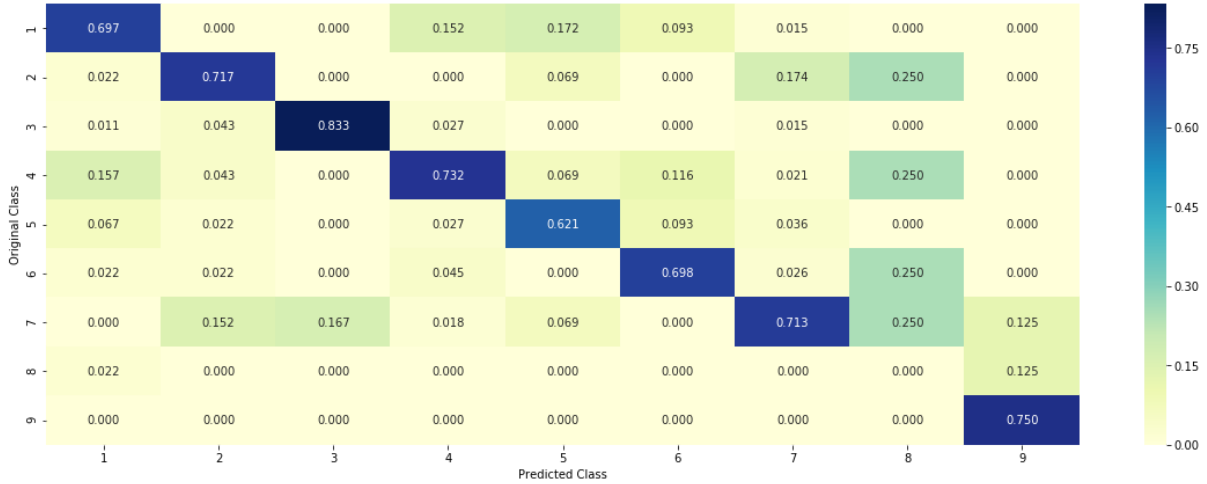
In [326]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, 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 Stochast
ic 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_onehotCo
ding, cv_y, clf)
```

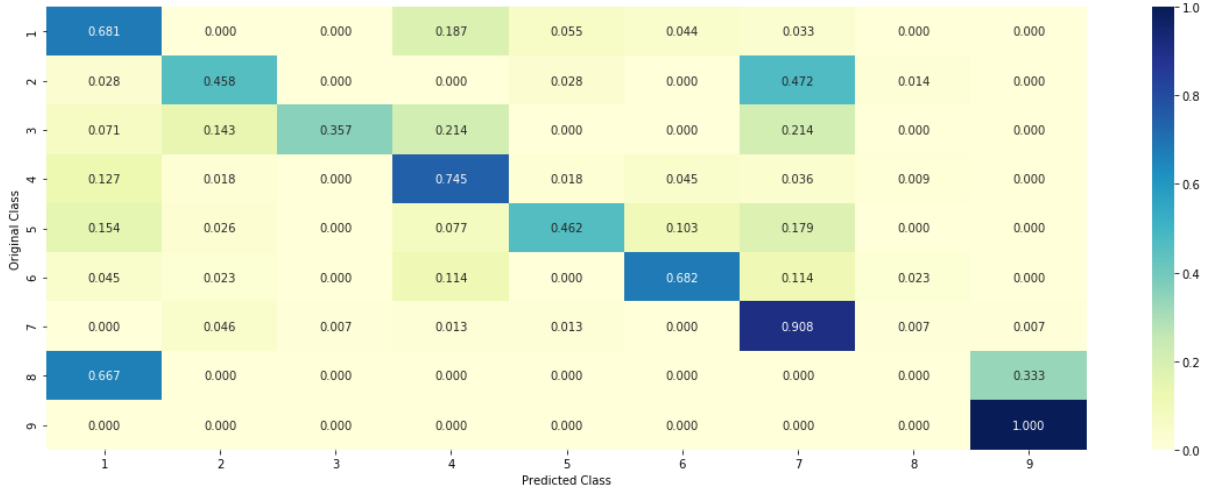
```
Log loss : 0.9541478702481345
Number of mis-classified points : 0.2951127819548872
-------------------- Confusion matrix --------------------
```




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



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



4.3.1.3. Feature Importance

In [327]:
```
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], ye
s_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query poin
t")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," cl
ass:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or
 Not']))
```

4.3.1.3.1. Correctly Classified point

In [328]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', 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 : 2
Predicted Class Probabilities: [[0.0063 0.5497 0.0033 0.0078 0.0302 0.0092
0.3848 0.0081 0.0006]]
Actual Class : 2
--------------------------------------------------
```

4.3.1.3.2. Incorrectly Classified point

In [329]:
```
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 : 7
Predicted Class Probabilities: [[0.0404 0.0621 0.0108 0.0391 0.0281 0.0169
0.7939 0.0048 0.004 ]]
Actual Class : 7
--------------------------------------------------
```

4.3.2. Without Class balancing

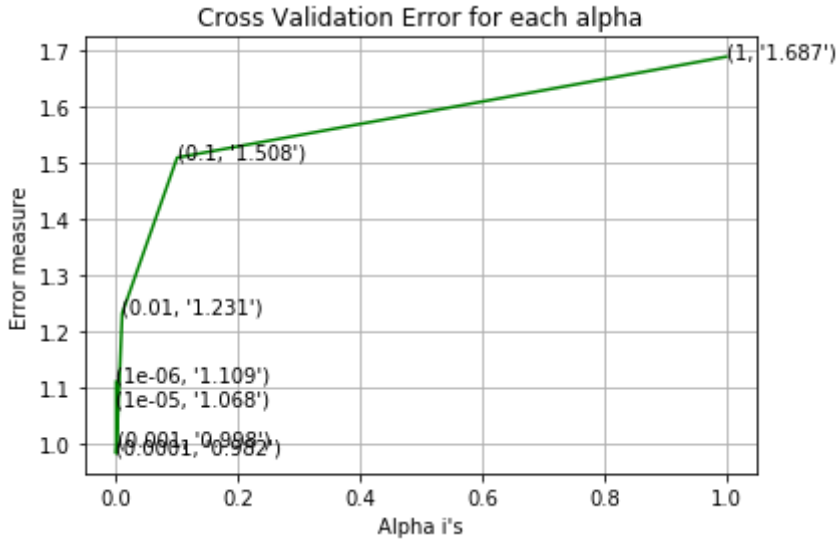
4.3.2.1. Hyper paramter tuning

In [330]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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/
#--------------------------------



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

```
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.109469275804417
for alpha = 1e-05
Log Loss : 1.0683437405676588
for alpha = 0.0001
Log Loss : 0.981756287843249
for alpha = 0.001
Log Loss : 0.9976035090112372
for alpha = 0.01
Log Loss : 1.231375738984243
for alpha = 0.1
Log Loss : 1.5078919931015102
for alpha = 1
Log Loss : 1.6872906162760626
```

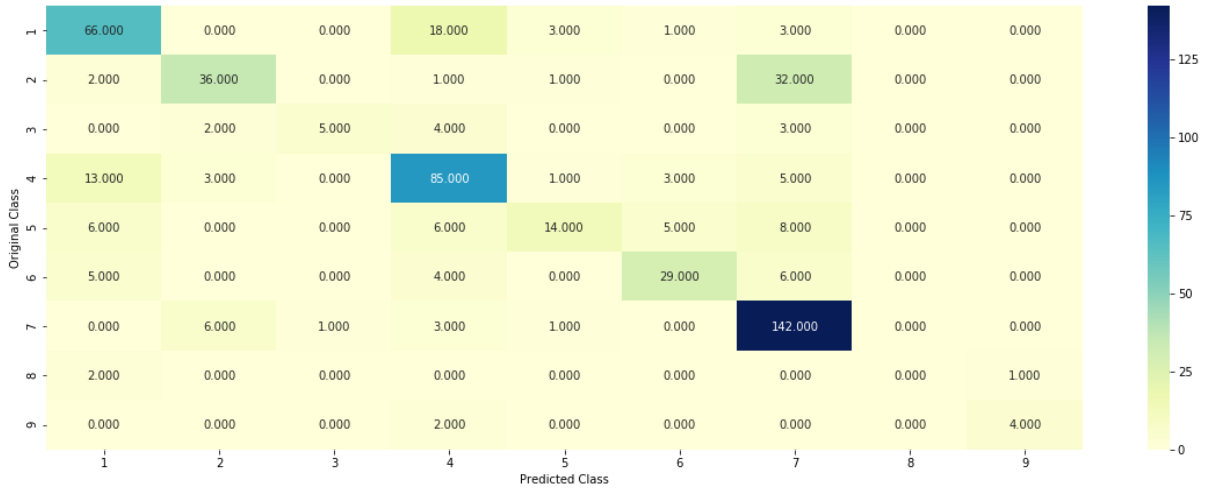



```
For values of best alpha =  0.0001 The train log loss is: 0.456388293599789
For values of best alpha =  0.0001 The cross validation log loss is: 0.9817
56287843249
For values of best alpha =  0.0001 The test log loss is: 0.9866424801154886
```

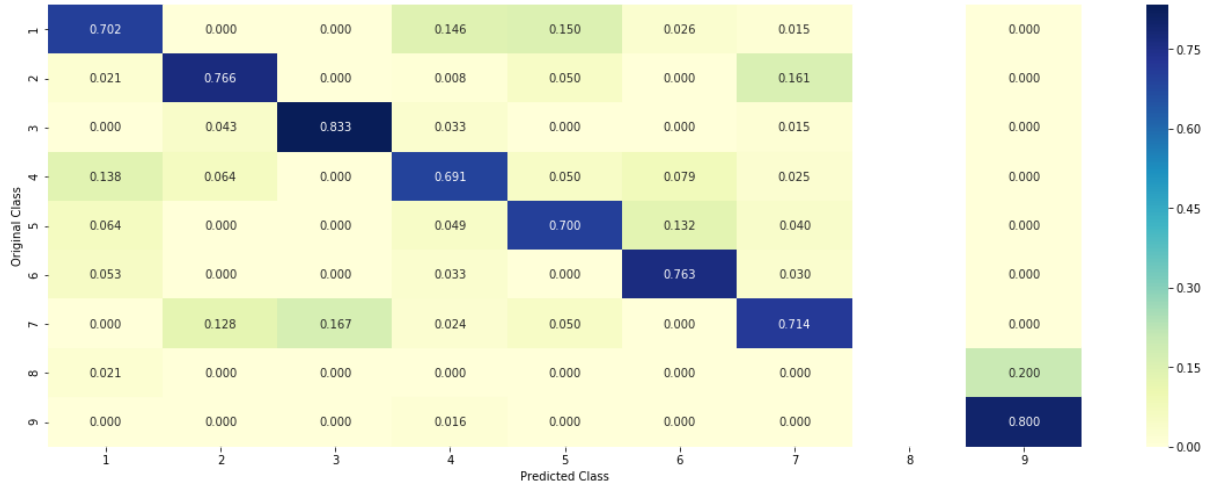
4.3.2.2. Testing model with best hyper parameters

In [331]:
```
# 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)
```

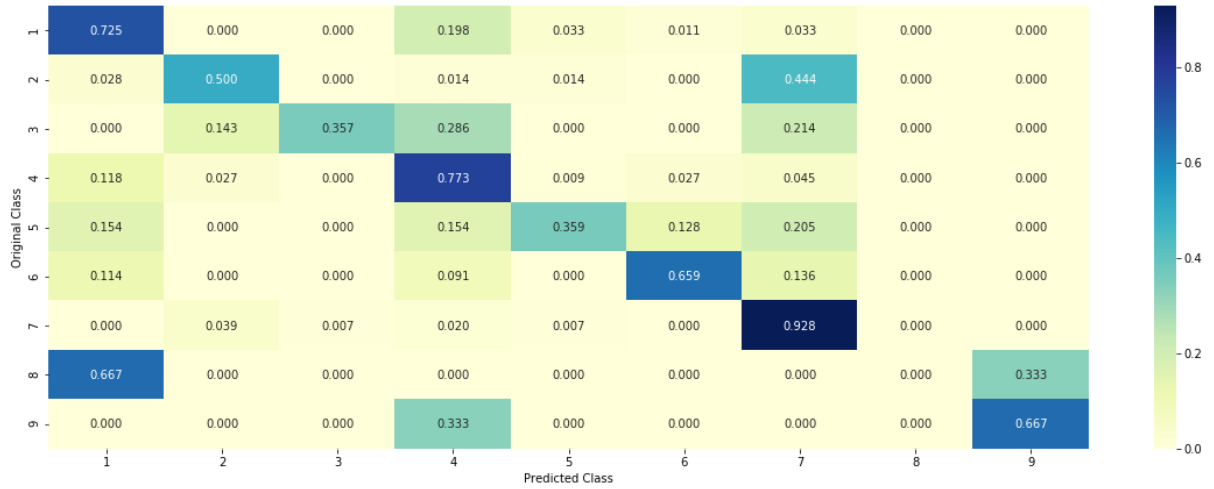
```
Log loss : 0.981756287843249
Number of mis-classified points : 0.28383458646616544
-------------------- Confusion matrix --------------------
```




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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [332]:

```
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 : 2
Predicted Class Probabilities: [[1.270e-02 7.029e-01 2.800e-03 8.600e-03 2.
940e-02 6.400e-03 2.286e-01
  8.300e-03 3.000e-04]]
Actual Class : 2
--------------------------------------------------
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [333]:
```
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 : 7
Predicted Class Probabilities: [[0.0371 0.0424 0.012  0.0197 0.0343 0.0165
0.8293 0.0047 0.0039]]
Actual Class : 7
--------------------------------------------------
```

4.4. Linear Support Vector Machines

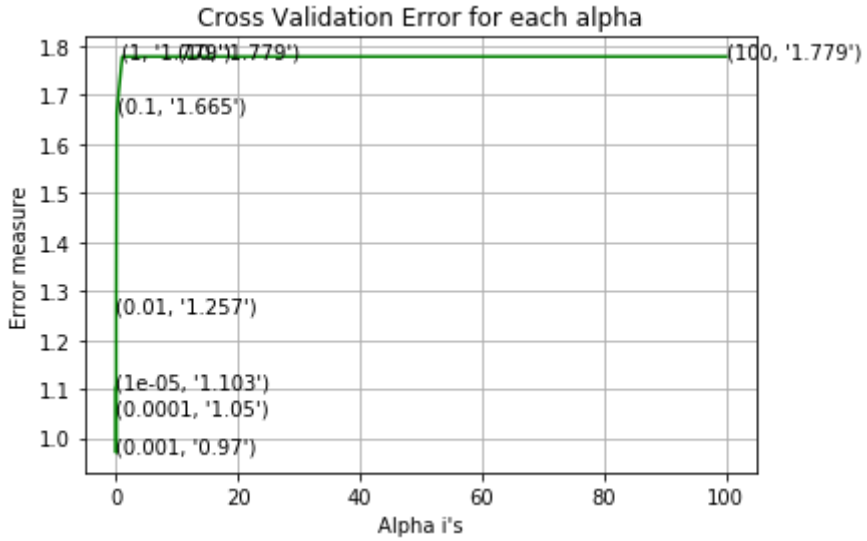
4.4.1. Hyper paramter tuning

In [334]:
```
# read more about support vector machines with linear kernals here http://scik
it-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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_func
tion_shape='ovr', random_state=None)
```

```
# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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', 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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.1030285478134367
for C = 0.0001
Log Loss : 1.0501924182104416
for C = 0.001
Log Loss : 0.9702112724497592
for C = 0.01
Log Loss : 1.2574780041704086
for C = 0.1
Log Loss : 1.664590275567539
for C = 1
Log Loss : 1.7785275815700619
for C = 10
Log Loss : 1.7785275064282509
for C = 100
Log Loss : 1.7785275973570276
```



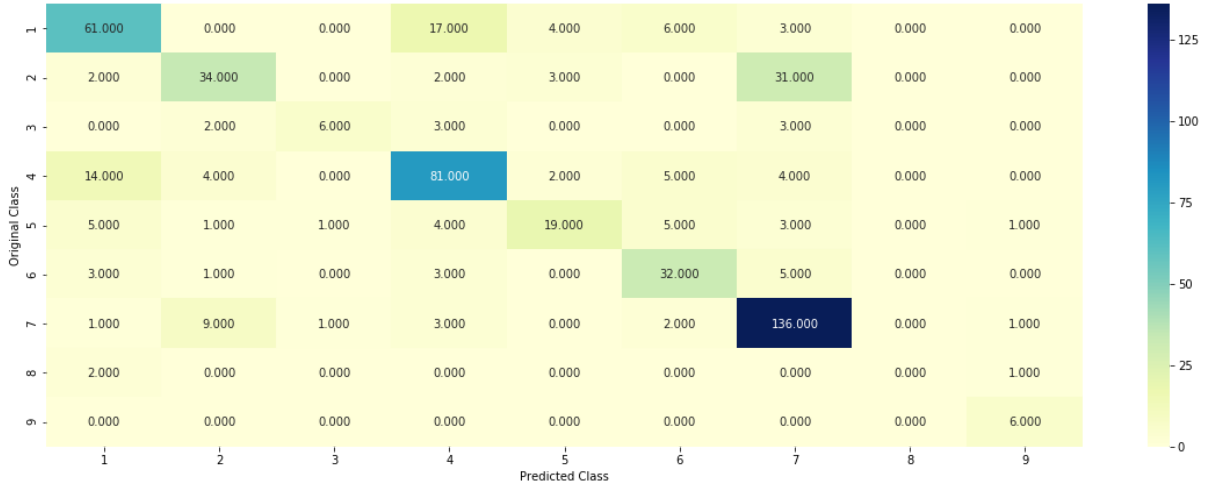

```
For values of best alpha =  0.001 The train log loss is: 0.5562551362875902
For values of best alpha =  0.001 The cross validation log loss is: 0.97021
12724497592
For values of best alpha =  0.001 The test log loss is: 1.0399317714748202
```

4.4.2. Testing model with best hyper parameters

In [335]:
```
# read more about support vector machines with linear kernals here http://scik
it-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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_func
tion_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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/
# --------------------------------
```

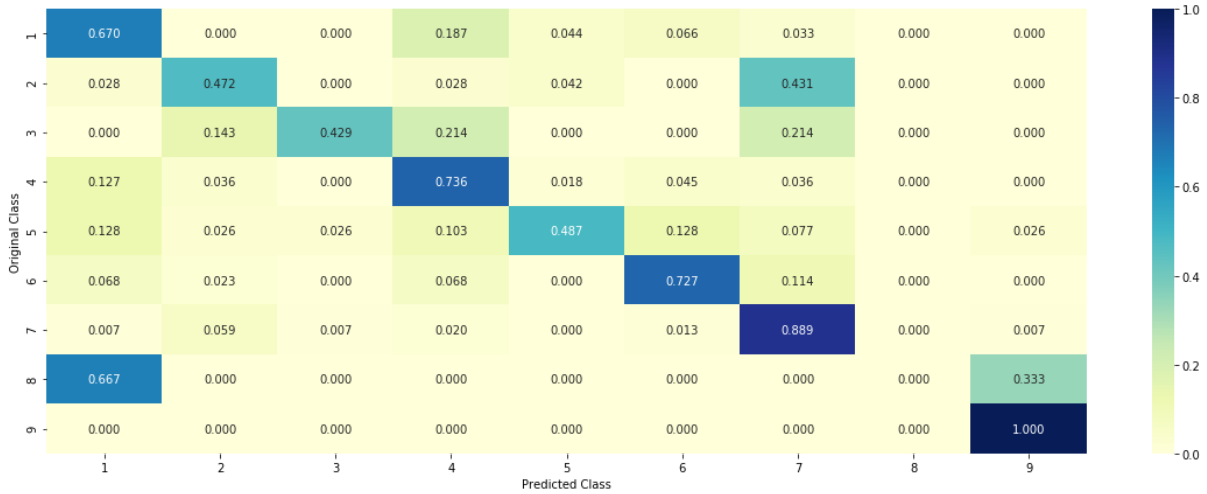
```
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight
='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)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

```
Log loss : 0.9702112724497592
Number of mis-classified points : 0.2951127819548872
-------------------- 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 [336]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_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)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0477 0.7184 0.0098 0.0286 0.0356 0.0136
0.1354 0.0079 0.003 ]]
Actual Class : 2
--------------------------------------------------
```

4.3.3.2. For Incorrectly classified point

In [337]:
```
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 : 7
Predicted Class Probabilities: [[0.066  0.037  0.0123 0.0246 0.0371 0.0196
0.7952 0.0047 0.0035]]
Actual Class : 7
--------------------------------------------------
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [338]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing 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/
```

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, 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],c
v_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()
```

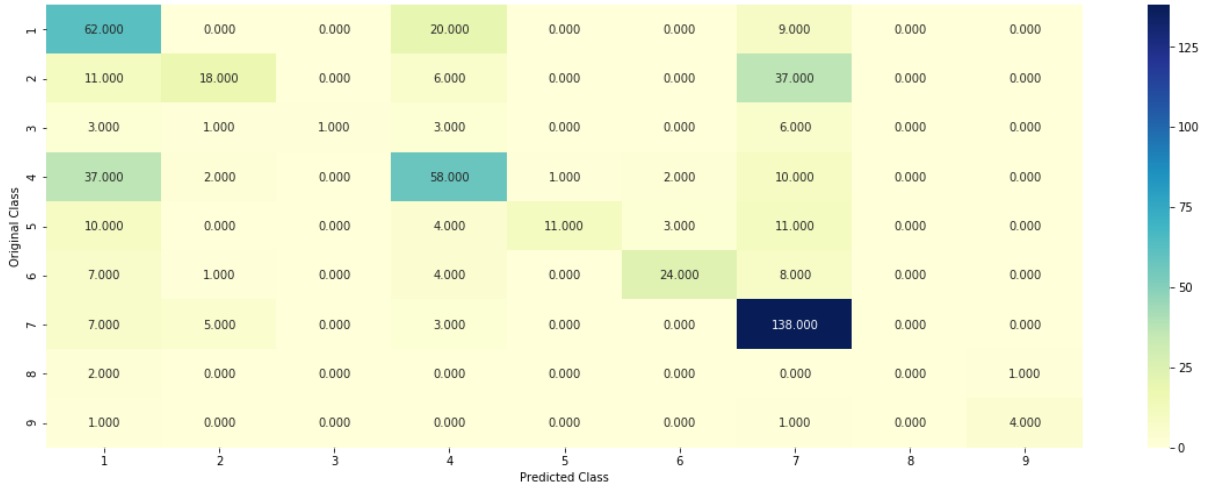
```
'''

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1957565414748372
for n_estimators = 100 and max depth =  10
Log Loss : 1.1922923251796937
for n_estimators = 200 and max depth =  5
Log Loss : 1.1833444907195354
for n_estimators = 200 and max depth =  10
Log Loss : 1.1848133910918428
for n_estimators = 500 and max depth =  5
Log Loss : 1.1751897894382282
for n_estimators = 500 and max depth =  10
Log Loss : 1.1775093298275472
for n_estimators = 1000 and max depth =  5
Log Loss : 1.175324285771641
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1759318447074925
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1746573512695804
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1750976400837914
For values of best estimator =  2000 The train log loss is: 0.9133225437684
408
For values of best estimator =  2000 The cross validation log loss is: 1.17
46573512695804
For values of best estimator =  2000 The test log loss is: 1.18432280742665
27
```

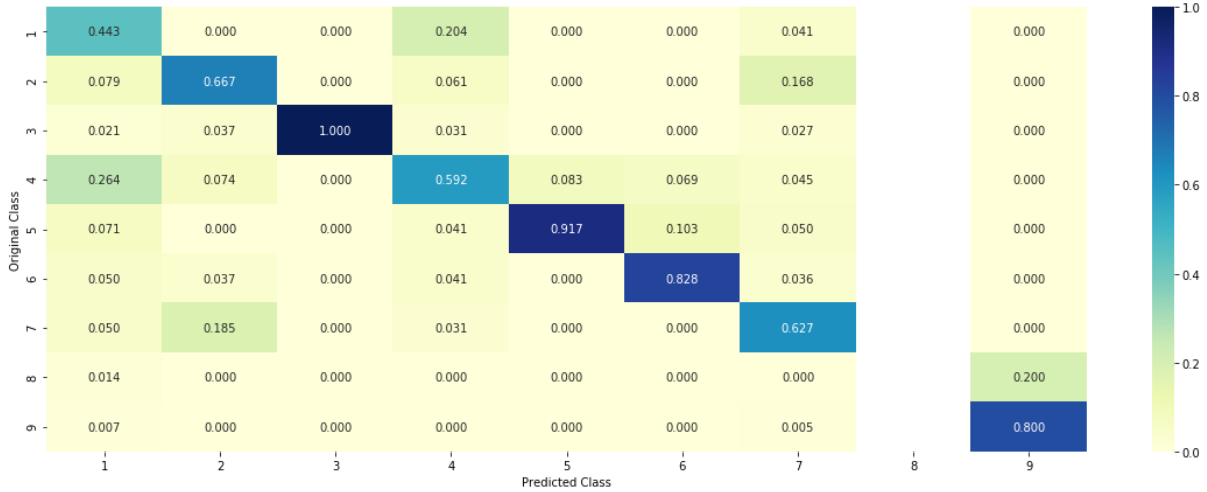
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [339]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_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)
```

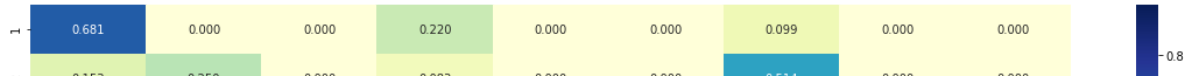
```
Log loss : 1.1746573512695804
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [340]:
```
# 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)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0321 0.3367 0.0178 0.0319 0.0469 0.037
0.4884 0.0068 0.0024]]
Actual Class : 2
--------------------------------------------------
```

4.5.3.2. Inorrectly Classified point

In [341]:
```
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)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0506 0.174  0.0193 0.0494 0.0519 0.0435
0.6019 0.0066 0.0029]]
Actuall Class : 7
--------------------------------------------------
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [342]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing 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/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, 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],c
v_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))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
```

```
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.252395114049456
for n_estimators = 10 and max depth =  3
Log Loss : 1.731497744533845
for n_estimators = 10 and max depth =  5
Log Loss : 1.3226414368397983
for n_estimators = 10 and max depth =  10
Log Loss : 2.0136420928338534
for n_estimators = 50 and max depth =  2
Log Loss : 1.752653605603669
for n_estimators = 50 and max depth =  3
Log Loss : 1.4520872336624988
for n_estimators = 50 and max depth =  5
Log Loss : 1.2873101045833217
for n_estimators = 50 and max depth =  10
Log Loss : 1.7841439204453657
for n_estimators = 100 and max depth =  2
Log Loss : 1.5728644081443437
for n_estimators = 100 and max depth =  3
Log Loss : 1.5258605820806055
for n_estimators = 100 and max depth =  5
Log Loss : 1.2615756094201513
for n_estimators = 100 and max depth =  10
Log Loss : 1.7541425342240604
for n_estimators = 200 and max depth =  2
Log Loss : 1.6622199505325508
for n_estimators = 200 and max depth =  3
Log Loss : 1.5578686388512741
for n_estimators = 200 and max depth =  5
Log Loss : 1.3670922590377885
for n_estimators = 200 and max depth =  10
Log Loss : 1.7883157999609138
for n_estimators = 500 and max depth =  2
Log Loss : 1.7500832413184833
for n_estimators = 500 and max depth =  3
Log Loss : 1.6102893559095166
for n_estimators = 500 and max depth =  5
Log Loss : 1.3780588483750758
for n_estimators = 500 and max depth =  10
Log Loss : 1.8537106360768496
```

```
for n_estimators = 1000 and max depth =  2
Log Loss : 1.711888119815348
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6059204768943431
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3557250699593748
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8467500422905054
For values of best alpha =  100 The train log loss is: 0.062364610913203444
For values of best alpha =  100 The cross validation log loss is: 1.2615756
094201513
For values of best alpha =  100 The test log loss is: 1.2463169929506979
```

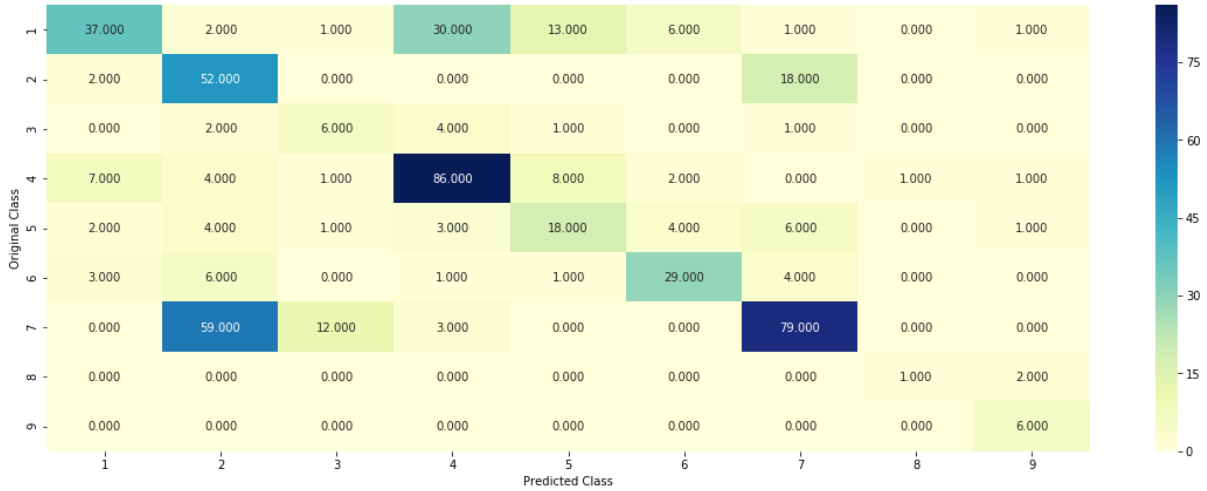
4.5.4. Testing model with best hyper parameters (Response Coding)

In [343]:

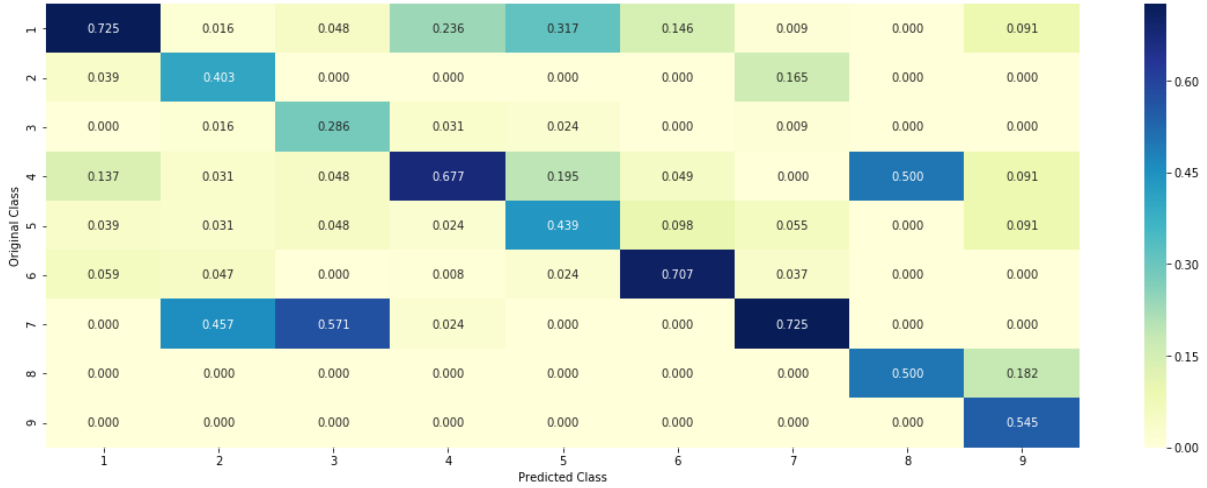
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing 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(max_depth=max_depth[int(best_alpha%4)], n_estimat
ors=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_sta
te=42)
```

```
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_respons
eCoding,cv_y, clf)
```

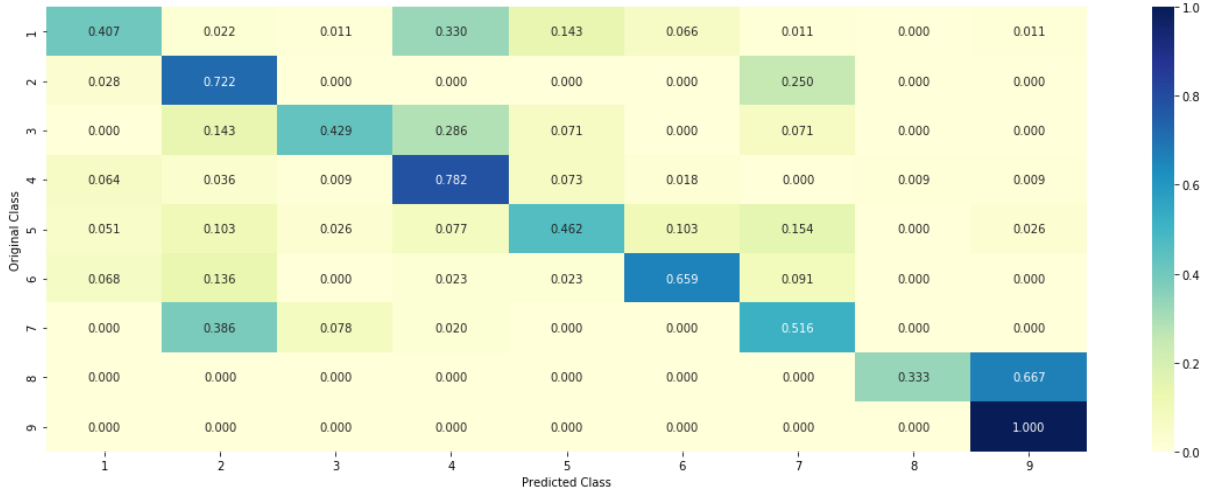
```
Log loss : 1.2615756094201513
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```




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




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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [344]:
```
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].reshap
e(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 : 2
Predicted Class Probabilities: [[0.0106 0.7461 0.0425 0.0136 0.0127 0.0277
0.1146 0.0242 0.0081]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation 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
```

4.5.5.2. Incorrectly Classified point

In [345]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(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: [[0.0237 0.195  0.2041 0.0223 0.0324 0.0451
0.4041 0.0489 0.0243]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation 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
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [346]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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/
#-------------------------------


# read more about support vector machines with linear kernals here http://scik
it-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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_func
tion_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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://scik
it-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.
```

```
html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing 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/
# --------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='bala
nced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanc
ed', random_state=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.pred
ict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
```

```
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.p
redict_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_probas=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, sclf.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: 0.96
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.17
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.485
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.063
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.180
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.521
```

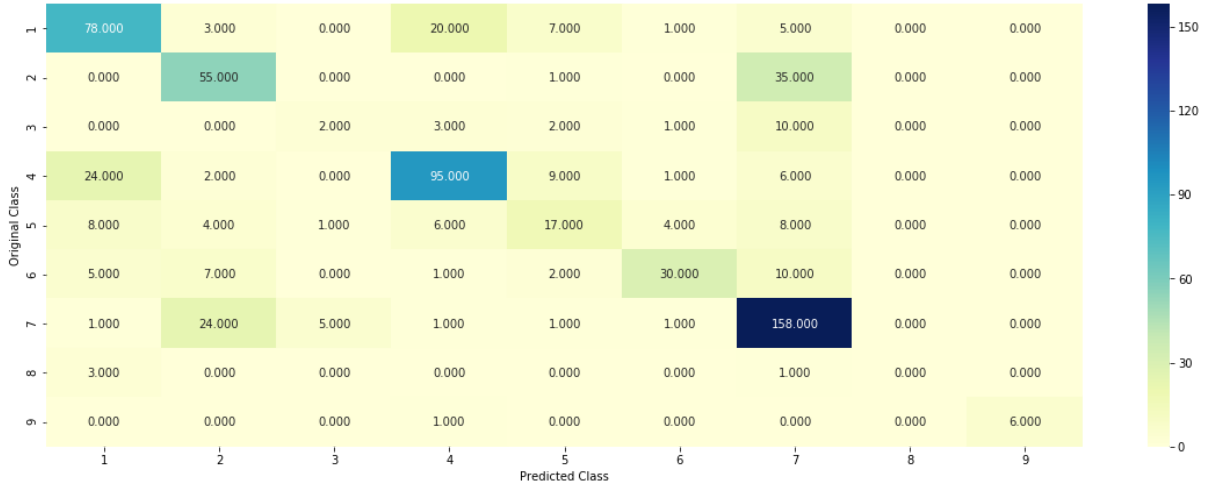
4.7.2 testing the model with the best hyper parameters

In [347]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=lr, use_probas=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))
```

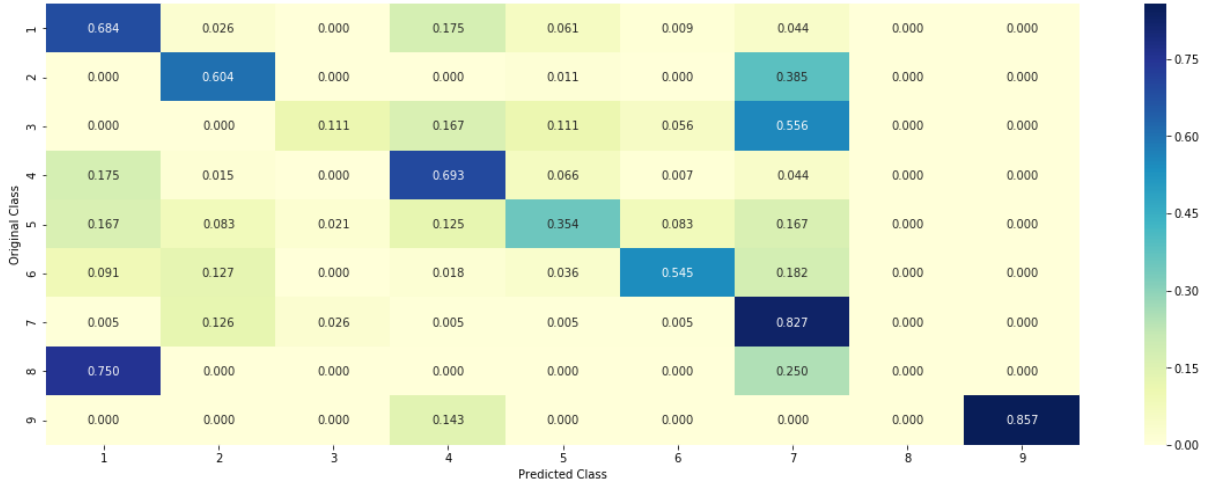
```
print("Log loss (test) on the stacking classifier :",log_error)

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_onehotCodin
g))
```

```
Log loss (train) on the stacking classifier : 0.6105009232885991
Log loss (CV) on the stacking classifier : 1.0626084032071585
Log loss (test) on the stacking classifier : 1.0836944361840508
Number of missclassified point : 0.3368421052631579
-------------------- Confusion matrix --------------------
```




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

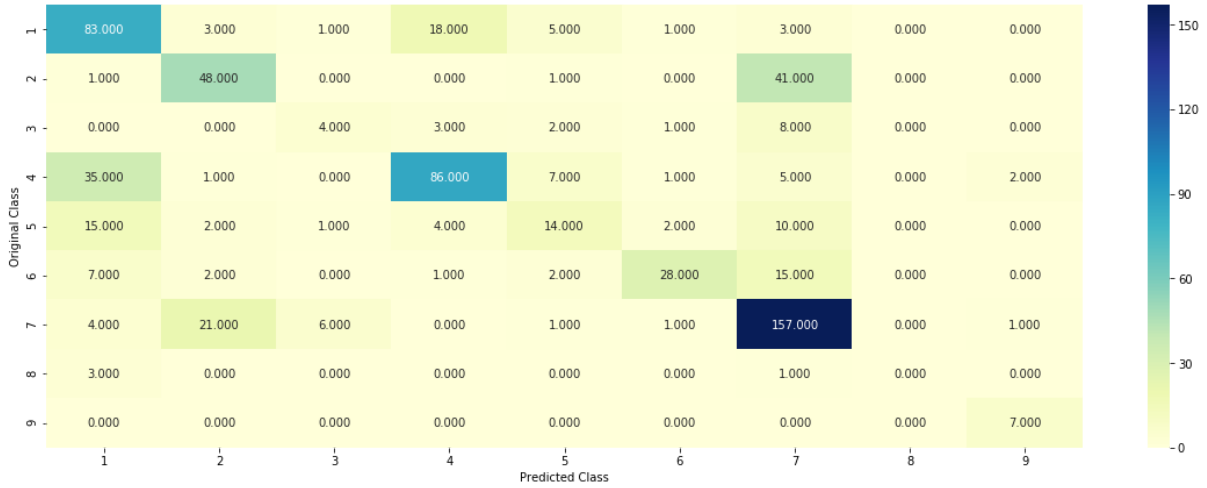

```
-------------------- Recall matrix (Row sum=1) --------------------
```



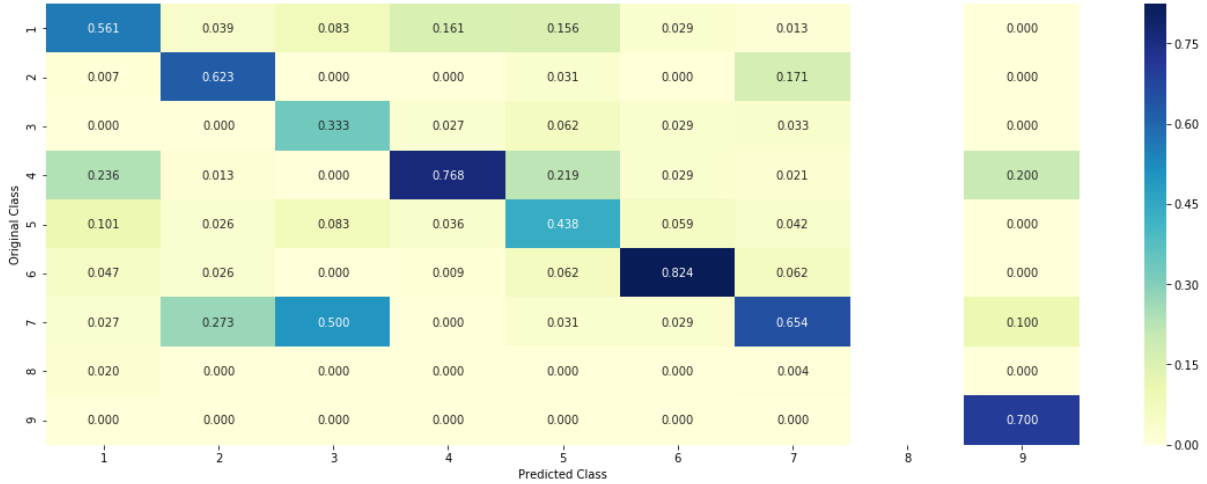
4.7.3 Maximum Voting classifier

In [348]:
```
#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 : 0.8620517996179623
Log loss (CV) on the VotingClassifier : 1.1384625629157292
Log loss (test) on the VotingClassifier : 1.1335406160107757
Number of missclassified point : 0.35789473684210527
-------------------- Confusion matrix --------------------
```



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



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

TfidfVectorization is used in every model with feature engineering

In [350]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No","Model","Train logloss","Cv logloss","Test logloss",
"Misclassified error"]

x.add_row(["1","Naive Bayes","0.760","1.158","1.188","0.36"])
x.add_row(["2","KNN","0.718","0.978","1.055","0.33"])
x.add_row(["3","Logistic regression with class balancing","0.648","0.95","0.97
7","0.30"])
x.add_row(["4","Logistic regression without class balancing","0.456","0.981",
"0.986","0.28"])
x.add_row(["5","Linear svm(with one hot encoding)","0.55","0.97","1.03","0.30"
])
x.add_row(["6","Random Forest(with one hot encoding)","0.97","1.17","1.18","0.
40"])
x.add_row(["7","Random Forest(with response coding)","0.062","1.26","1.24","0.
41"])
x.add_row(["8","Stacking classifier","0.61","1.06","1.08","0.33"])
x.add_row(["9","Maximum voting classifier","0.86","1.138","1.33","0.35"])


print(x)
```

```
+------+-----------------------------------------------+---------------+-----
-------+-------------+---------------------+
| S.No |                     Model                     | Train logloss | Cv l
ogloss | Test logloss | Misclassified error |
+------+-----------------------------------------------+---------------+-----
-------+-------------+---------------------+
|  1   |                  Naive Bayes                  |     0.760     |   1.
158    |    1.188     |         0.36        |
|  2   |                     KNN                       |     0.718     |   0.
978    |    1.055     |         0.33        |
|  3   |   Logistic regression with class balancing    |     0.648     |
0.95    |    0.977     |         0.30        |
|  4   | Logistic regression without class balancing   |     0.456     |   0.
```

```
981    |    0.986     |         0.28        |
|  5   |      Linear svm(with one hot encoding)      |      0.55     |
0.97    |     1.03     |         0.30        |
|  6   |     Random Forest(with one hot encoding)    |      0.97     |
1.17    |     1.18     |         0.40        |
|  7   |     Random Forest(with response coding)     |     0.062     |
1.26    |     1.24     |         0.41        |
|  8   |              Stacking classifier            |      0.61     |
1.06    |     1.08     |         0.33        |
|  9   |          Maximum voting classifier          |      0.86     |   1.
138    |     1.33     |         0.35        |
+------+---------------------------------------------+---------------+-----
-------+--------------+---------------------+
```

Conclusion:

1)After Feature engineering Misclassified errors are quite good when compare to without feature engineering

2)And also Logistic regression model gives good MSE and also cv log loss and test log loss are less than 1.0