Personalized cancer diagnosis

OBJECTIVE :- Try any Feature Engineering Technique to reduce 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 [2]:

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


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 [3]:

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

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

Out[3]:

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

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

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

3.1.2. Reading Text Data

In [4]:

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

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

Out[4]:

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

3.1.3. Preprocessing of text

In [5]:

```
# 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 [6]:

```
#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 : 186.55629841623562 seconds
```

In [7]:

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

Out[7]:

	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 [8]:

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

Out[8]:

	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 [9]:

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

In [10]:

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

Out[10]:

	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 [11]:

```
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 [12]:

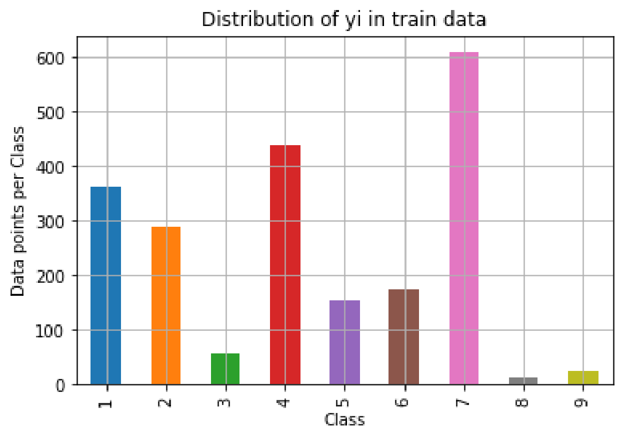
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

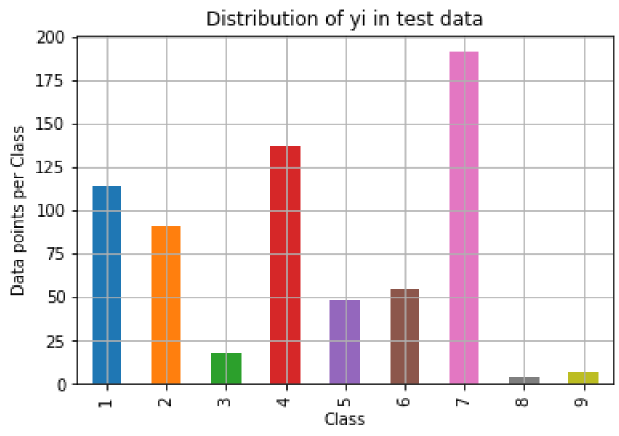
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [13]:

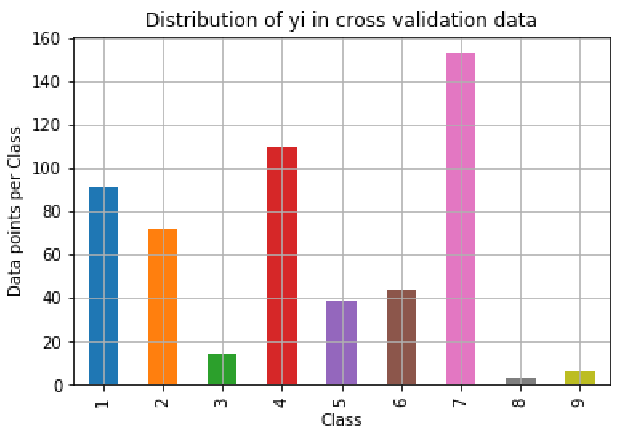
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


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

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

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

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



```
Number of data points in class 7 : 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.

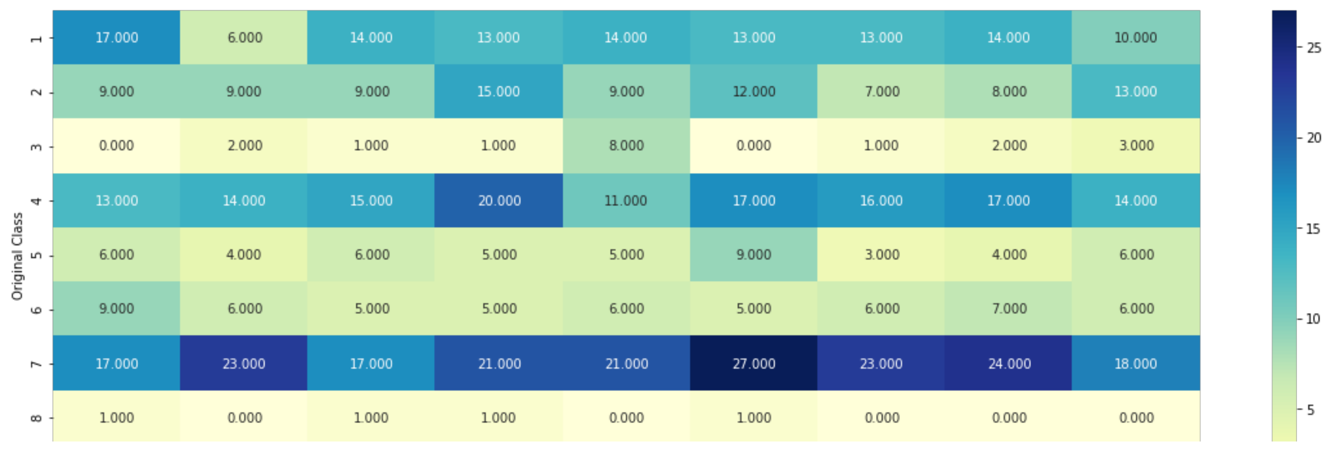
In [14]:

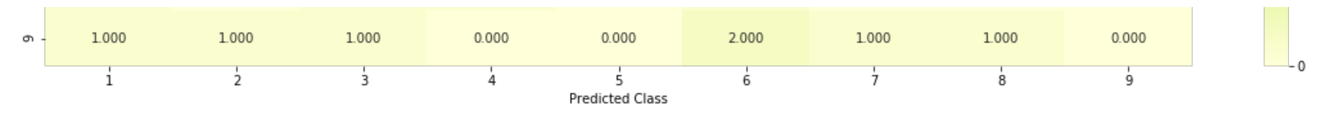
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column
    
    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two 
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1
    
    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two 
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                       [3/4, 4/6]]

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

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

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

In [15]:

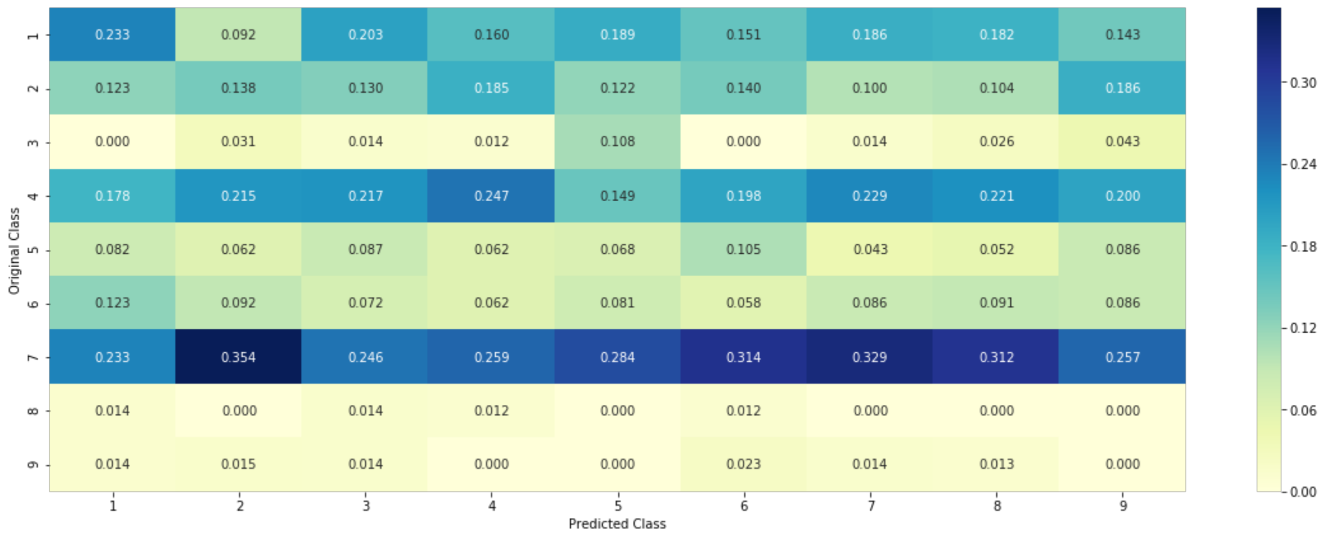
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.476183690719673
Log loss on Test Data using Random Model 2.4659554652747824
-------------------- Confusion matrix --------------------
```

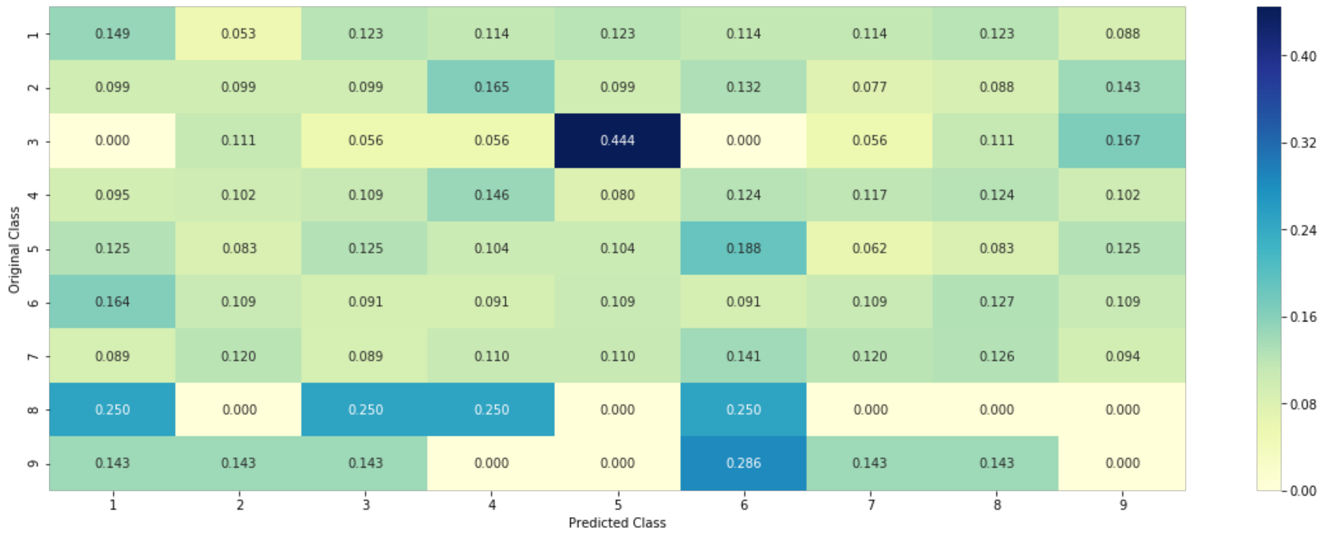




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



3.3 Univariate Analysis

In [16]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / nu
mber of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------
# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.07547169811320754 4, 0.06289308176100628 9, 0.06918238993710691 7, 0.062893081
761006289, 0.06289308176100 6289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the da
ta
    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 [17]:

```
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 : 235
BRCA1     165
TP53      107
EGFR       91
BRCA2      87
PTEN       76
KIT        63
BRAF       60
ERBB2      44
PIK3CA     41
ALK        39
Name: Gene, dtype: int64
```

In [18]:

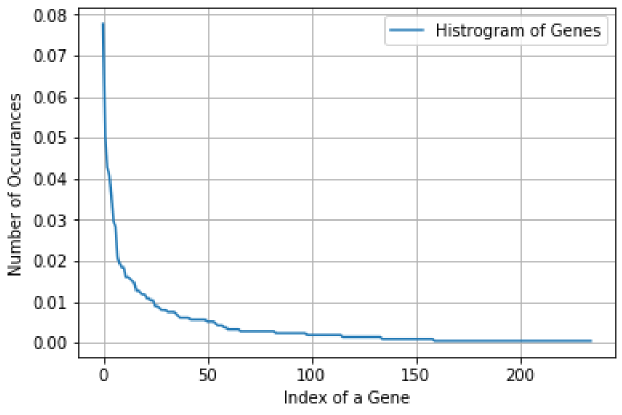
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
d they are distibuted as follows",)
```

```
Ans: There are 235 different categories of genes in the train data, and they are distibuted as fol
lows
```

In [19]:

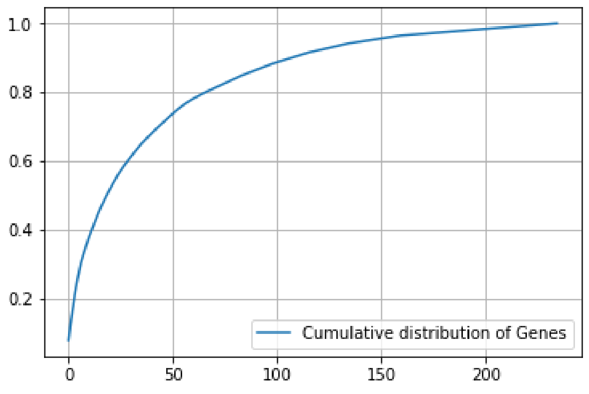
```
s = sum(unique_genes.values);
```

```
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 [20]:

```
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 [21]:

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

In [22]:

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

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

In [23]:

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

In [24]:

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

Out[24]:

```
136      SF3B1
849       ABL1
1233      PIM1
2921    NFE2L2
2423     BRCA1
Name: Gene, dtype: object
```

In [25]:

```
gene_vectorizer.get_feature_names()
```

Out[25]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'aurka',
 'aurkb',
 'axin1',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'elf3',
 'ep300',
 'epas1',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf3',
 'fgf4',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt3',
 'foxa1',
 'foxl2',
 'foxo1',
 'foxp1',
 'fubp1',
 'gata3',
 'gli1',
 'gna11',
 'gnas',
 'h3f3a',
 'hist1h1c',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'il7r',
 'inpp4b',
 'jak1',
 'jak2',
 'jun',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'klf4',
 'kmt2b',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats2',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'ncor1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad51d',
 'rad54l',
 'raf1',
 'rasa1',
 'rb1',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'xpo1',
 'yap1']
```

In [26]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 234)
```

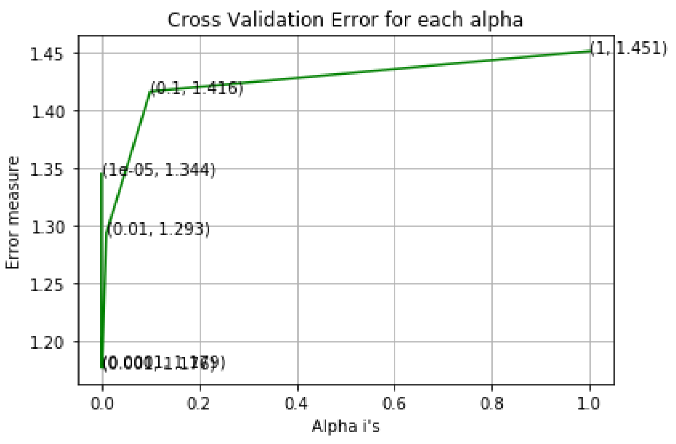
Q4. How good is this gene feature in predicting y_i?

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 [27]:

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

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

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

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


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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3443936520566035
For values of alpha =  0.0001 The log loss is: 1.1786944454113322
For values of alpha =  0.001 The log loss is: 1.1763630327535848
For values of alpha =  0.01 The log loss is: 1.2931758018818347
For values of alpha =  0.1 The log loss is: 1.4157748486302784
For values of alpha =  1 The log loss is: 1.4506328745824781
```



```
For values of best alpha =  0.001 The train log loss is: 1.106659592123214
For values of best alpha =  0.001 The cross validation log loss is: 1.1763630327535848
For values of best alpha =  0.001 The test log loss is: 1.242694652743841
```

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 [28]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  235  genes in train dataset?
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 517 out of  532 : 97.18045112781954
```

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 [29]:

```
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 : 1937
Truncating_Mutations    62
Amplification           46
Deletion                40
Fusions                 23
Overexpression           4
G12V                     3
Q61L                     3
T73I                     2
P34R                     2
C618R                    2
Name: Variation, dtype: int64
```

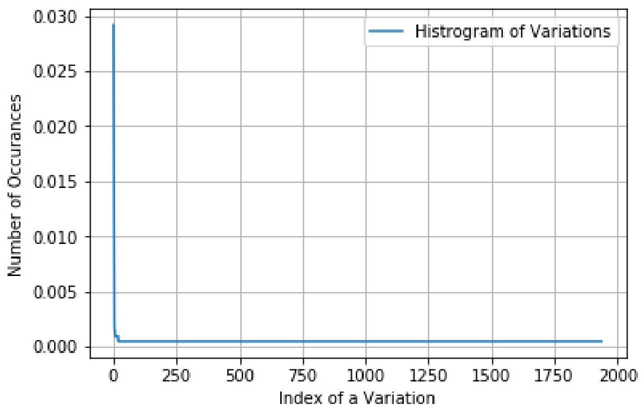
In [30]:

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

```
Ans: There are 1937 different categories of variations in the train data, and they are distibuted
as follows
```

In [31]:

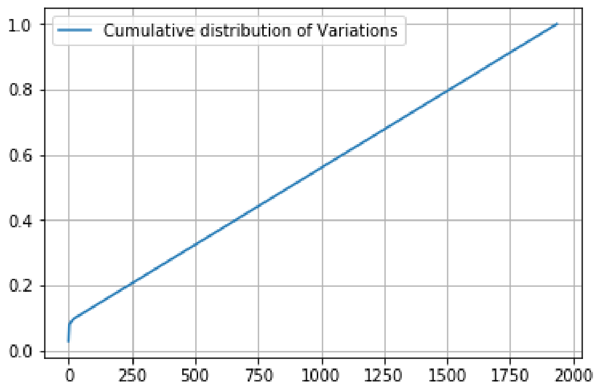
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [32]:

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

```
[0.02919021 0.05084746 0.06967985 ... 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 [33]:

```
# 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 [34]:

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

```
train_variation_feature_responseCoding is a converted feature using the response coding method. Th
```

```
e shape of Variation feature: (2124, 9)
```

In [35]:

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

In [36]:

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

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

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

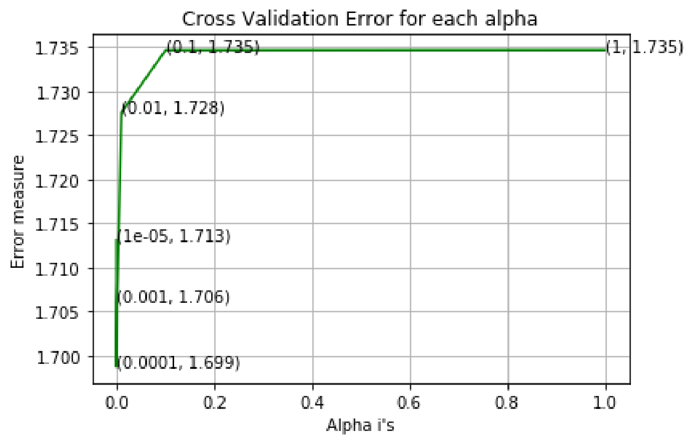
Let's build a model just like the earlier!

In [37]:

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

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

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

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


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

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

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

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


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

```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.7130851810950938
For values of alpha =  0.0001 The log loss is: 1.6986811787610299
For values of alpha =  0.001 The log loss is: 1.7061358416055468
For values of alpha =  0.01 The log loss is: 1.7276007147272798
For values of alpha =  0.1 The log loss is: 1.734651700623611
For values of alpha =  1 The log loss is: 1.734671272266528
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7131956288410888
For values of best alpha =  0.0001 The cross validation log loss is: 1.6986811787610299
For values of best alpha =  0.0001 The test log loss is: 1.6886829465718476
```

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 [38]:

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

```
Q12. How many data points are covered by total  1937  genes in test and cross validation data sets?
Ans
1. In test data 76 out of 665 : 11.428571428571429
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 [39]:

```
# 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 [40]:

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

In [41]:

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

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

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


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

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

FEATURE ENGINEERING

NOTE : - Here I have collected all genes and their variations in a single list and fit this list to the TfidfVectorizer . Then transform the train_df['TEXT'] , test_df['TEXT'] and cv_df['TEXT'] using this text vectorizer so as to find some more relations between them and to gain max information from the literature regarding the gene mutation and their variations

In [42]:

```
# Collecting all the genes and variations in a single list
corpus = []
for word in data['Gene'].values:
    corpus.append(word)
for word in data['Variation'].values:
    corpus.append(word)
```

In [43]:

```
# Training TfidfVectorizer on the 'corpus' list
```

```
text1 = TfidfVectorizer()
text2 = text1.fit_transform(corpus)
text1_features = text1.get_feature_names()

# Transforming the train_df['TEXT']
train_text = text1.transform(train_df['TEXT'])

# Transforming the test_df['TEXT']
test_text = text1.transform(test_df['TEXT'])

# Transforming the cv_df['TEXT']
cv_text = text1.transform(cv_df['TEXT'])

# Normalizing the train_text
train_text = normalize(train_text,axis=0)

# Normalizing the test_text
test_text = normalize(test_text,axis=0)

# Normalizing the cv_text
cv_text = normalize(cv_text,axis=0)
```

NOTE :- I have added this feature in the onehotCoding of all the features

In [44]:

```
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 [45]:

```
#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 [46]:

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

In [47]:

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

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

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

In [48]:

```
#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 [49]:

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

```
Counter({251.57438632342271: 1, 173.05525439010364: 1, 134.31080360170822: 1, 132.09001098211638:
1, 125.89521331068796: 1, 115.12841799806785: 1, 112.31441917389556: 1, 111.67678842170965: 1, 110
.18204674050598: 1, 105.4554511443073: 1, 103.4692283325178: 1, 88.45015009674857: 1, 88.3918679108
086: 1, 86.56668292246655: 1, 80.50028566880445: 1, 80.20459158285179: 1, 77.9029492298702: 1,
77.20529732578296: 1, 76.95649840468259: 1, 75.69615153280213: 1, 75.56659323981016: 1,
73.74409628182941: 1, 70.17222594005604: 1, 68.42493485874678: 1, 67.55081393848629: 1,
67.17536417736741: 1, 66.55100888572241: 1, 66.41026885495125: 1, 64.61817298146134: 1,
63.73352354287781: 1, 63.160317425364894: 1, 62.743875726902104: 1, 61.216733513383744: 1,
59.94765564749114: 1, 58.36613937333939: 1, 55.95051810212409: 1, 55.358491362278336: 1,
53.748618442687615: 1, 52.295651884539566: 1, 51.58447894588376: 1, 49.90961279570698: 1,
49.807348063497585: 1, 48.35367158144103: 1, 48.23496892402996: 1, 46.45226153436802: 1, 45.6378023
7567824: 1, 45.54361662775149: 1, 45.356280641735246: 1, 44.39232275294146: 1, 44.277095397640714:
1, 43.83435007599116: 1, 43.36517164459036: 1, 43.175655201100376: 1, 42.69001594089466: 1,
42.593109247671606: 1, 42.49560366887909: 1, 42.383901233607986: 1, 42.02421092280863: 1,
41.516656155347604: 1, 41.3885857918391: 1, 41.21720179463078: 1, 41.18146718241351: 1,
41.155366129925305: 1, 40.63457808706453: 1, 40.37161875588329: 1, 40.32841632912944: 1, 39.5635553
57916705: 1, 39.19654335367266: 1, 39.15055660515008: 1, 39.062917228226055: 1, 38.87607186277879:
1, 38.2124141952O218: 1, 37.84917895090281: 1, 37.73733826411798: 1, 36.70420853259683: 1,
35.736918999830344: 1, 35.67947922944016: 1, 35.57983080756222: 1, 35.32391773856447: 1, 35.2440642
3549289: 1, 35.1184554187795: 1, 34.887500164679395: 1, 34.86589451621099: 1, 34.81258525581303:
1, 34.81220804140793: 1, 34.717971429230715: 1, 34.54904170374337: 1, 34.24080133614028: 1,
33.78755303890663: 1, 33.404141244957074: 1, 33.10940757718933: 1, 33.05547050728417: 1,
33.028116941702415: 1, 32.69491601069398: 1, 32.68870606320596: 1, 32.62343890284679: 1, 32.3018279
8515043: 1, 32.259568923488814: 1, 32.115885729292586: 1, 32.065183350526624: 1,
31.604803367158922: 1, 31.59422053396663: 1, 31.40924276860093: 1, 31.390296439630795: 1,
31.38938252453254: 1, 31.257278194603558: 1, 31.164548000389345: 1, 31.00179201771293: 1,
30.821646643110004: 1, 30.774585882405844: 1, 30.62762413387965: 1, 30.587523784653662: 1,
30.50988596102781: 1, 30.27303710223893: 1, 30.167759580380416: 1, 30.131811379440453: 1,
30.116791699596114: 1, 30.048920747960626: 1, 29.747787112201777: 1, 29.735842105519602: 1,
29.59598571612562: 1, 29.102427960208995: 1, 29.034026799849105: 1, 28.95073535393567: 1,
28.572976861331004: 1, 28.50244423700852: 1, 28.49811973502315: 1, 28.318033162678166: 1,
28.277075534638076: 1, 28.13503168661557: 1, 27.937144749902494: 1, 27.730100579353437: 1,
27.69188577977784: 1, 27.492323220430638: 1, 27.339995941973164: 1, 27.239867319194758: 1,
26.8672557312298: 1, 26.698871023293727: 1, 26.6859932443602: 1, 26.539392184368076: 1,
26.534474787139946: 1, 26.466333746985285: 1, 26.38485418917919: 1, 26.375745573291777: 1,
26.334943009622158: 1, 26.17631599069066: 1, 26.09099584530261: 1, 26.02584023749815: 1, 25.954921O
46422776: 1, 25.915189455223523: 1, 25.47195622944241: 1, 25.391145506424426: 1,
25.108020155866377: 1, 25.104995452573437: 1, 25.081376943223628: 1, 24.94238013414795: 1,
24.811069211532846: 1, 24.803750735755543: 1, 24.758054064406775: 1, 24.7071156980106 7: 1,
24.687764071911428: 1, 24.64510070309211: 1, 24.63820782961759: 1, 24.610710948620344: 1,
24.42330833672692: 1, 24.352179160189966: 1, 24.327422402171223: 1, 24.233852321238906: 1,
24.155774226411676: 1, 24.100360788540474: 1, 24.08859891138661: 1, 24.068386046418905: 1,
24.01757023646352: 1, 24.014959517000026: 1, 23.929535118585648: 1, 23.914432335104138: 1,
23.86321993294706: 1, 23.804498885921323: 1, 23.721509911284727: 1, 23.65331174146589: 1,
23.626874348216695: 1, 23.533729253691376: 1, 23.443619844648996: 1, 23.28900701788364: 1,
23.150629291658493: 1, 23.080015767343465: 1, 22.915806980208796: 1, 22.83889079927148: 1,
22.784949399947475: 1, 22.682316006803738: 1, 22.678613986427433: 1, 22.630257231324688: 1,
22.588171405483767: 1, 22.587139992544397: 1, 22.375691358736912: 1, 22.23630217353597: 1,
22.21182850924773: 1, 22.07559554480959: 1, 21.91854309729908: 1, 21.899684992394775: 1,
21.88218346791068: 1, 21.834265636653008: 1, 21.76237467050268: 1, 21.674314777859905: 1,
21.63509637661717: 1, 21.627397710060862: 1, 21.607243664541535: 1, 21.539948773445577: 1,
21.524075480795002: 1, 21.523650590316894: 1, 21.456051992820363: 1, 21.43931487338075: 1,
21.42194984970197: 1, 21.359189489729744: 1, 21.265865431090532: 1, 21.166057640547773: 1,
21.129016513650967: 1, 21.116080137345413: 1, 21.09718526543054: 1, 21.09375891642317: 1,
21.129016315030907: 1, 21.110001575454115: 1, 21.097163203450 51: 1, 21.09375091042317: 1,
21.039645359077962: 1, 21.020261798512657: 1, 20.97080534278696: 1, 20.924562470265812: 1,
20.892098903854393: 1, 20.831483588874818: 1, 20.764415480635456: 1, 20.700131468019762: 1,
20.680515641230684: 1, 20.503646138987364: 1, 20.476885416099254: 1, 20.464900158827135: 1,
20.452716125171168: 1, 20.426335478158922: 1, 20.41345752358542: 1, 20.35394398827253: 1,
20.33923510040244: 1, 20.09976094826299: 1, 20.049382668788372: 1, 19.98635196524086: 1,
19.828927946967887: 1, 19.812961939625676: 1, 19.807983788311443: 1, 19.787262972060507: 1,
19.783083869442805: 1, 19.751202494418358: 1, 19.691158409293713: 1, 19.631008063258015: 1,
19.62371630748583: 1, 19.552980124681692: 1, 19.517070554241204: 1, 19.46472107431076: 1,
19.422854200293223: 1, 19.364421670069: 1, 19.362965032513948: 1, 19.350223244675654: 1,
19.320973737541337: 1, 19.312471350399083: 1, 19.301752342985136: 1, 19.184455199423383: 1,
19.173353123767882: 1, 19.14905148340308: 1, 19.137350719208044: 1, 19.129300366323463: 1,
19.113235124383312: 1, 19.091701799584605: 1, 19.069137717827406: 1, 19.050326094992524: 1,
19.016487082977765: 1, 18.93522923160663: 1, 18.932361220121457: 1, 18.90922852738625: 1,
18.897905249427097: 1, 18.87534715027217: 1, 18.87063554716018: 1, 18.858290633503863: 1,
18.844350181032087: 1, 18.79121795381877: 1, 18.745720295475266: 1, 18.661308793595097: 1,
18.64506852428096: 1, 18.58379269098583: 1, 18.57945585692426: 1, 18.553373732442434: 1,
18.522597856649625: 1, 18.427933492932873: 1, 18.413964898937344: 1, 18.385403472009408: 1,
18.345860008169605: 1, 18.333881585801514: 1, 18.24445477702514: 1, 18.231786056438764: 1,
18.19370487373536: 1, 18.175644857061258: 1, 18.139569192462005: 1, 18.038728223156205: 1,
18.027471592593407: 1, 18.010133978081956: 1, 17.845632311118067: 1, 17.82024634860162: 1,
17.809402893207103: 1, 17.719907397468: 1, 17.71163712380207: 1, 17.695542552268847: 1,
17.673538855492282: 1, 17.612509523819295: 1, 17.576145923297258: 1, 17.55686623521779: 1,
17.514832112016993: 1, 17.51388746164991: 1, 17.513796058569564: 1, 17.485808849331207: 1,
17.421775142680417: 1, 17.402866205974647: 1, 17.39555723428535: 1, 17.37829997107862: 1,
17.322319408555586: 1, 17.313563314705245: 1, 17.306418240009304: 1, 17.293013327116967: 1,
17.126056632015924: 1, 17.123750862886467: 1, 17.08901248067968: 1, 17.07377354165954: 1,
17.03837168168493: 1, 17.020263661818337: 1, 17.01002066039137: 1, 16.94893117280987: 1,
16.91128995053334: 1, 16.89249074306168: 1, 16.886539206177485: 1, 16.873078282957803: 1,
16.817126487322145: 1, 16.806956516582545: 1, 16.77921241021119: 1, 16.77103027739043: 1,
16.726666497762587: 1, 16.688000146439585: 1, 16.685407806189044: 1, 16.662981858741638: 1,
16.56413719908667: 1, 16.54070471691353: 1, 16.52804068854835: 1, 16.507241722995555: 1,
16.49556621008926: 1, 16.486299746086946: 1, 16.430011331896214: 1, 16.40895025463731: 1,
16.37814859491864: 1, 16.37034289440376: 1, 16.344348424733315: 1, 16.337891643028918: 1,
16.319492421166913: 1, 16.306190788577336: 1, 16.290179597881316: 1, 16.242195309770622: 1,
16.23901657572557: 1, 16.22200666642402: 1, 16.21838563142543: 1, 16.13708857618499: 1,
16.124663147621416: 1, 16.07820431507468: 1, 16.076516368026276: 1, 16.07072401334151: 1,
16.05091934435291: 1, 16.00566369947747: 1, 15.999710049801577: 1, 15.94152940042169: 1,
15.90919449289692: 1, 15.900821658429924: 1, 15.897791171208898: 1, 15.896014311967011: 1,
15.896010794201308: 1, 15.862735608234768: 1, 15.855749582791093: 1, 15.82506816989564: 1,
15.789986960601649: 1, 15.78916969461788: 1, 15.775841790148824: 1, 15.744319933666901: 1,
15.696742421874003: 1, 15.68879945347092: 1, 15.677520644103799: 1, 15.670218609150213: 1,
15.664935598882447: 1, 15.626960881277363: 1, 15.591562181989062: 1, 15.575772406595709: 1,
15.569795326663117: 1, 15.552997525046345: 1, 15.517078734410664: 1, 15.492951871844895: 1,
15.491994051018382: 1, 15.475391059480176: 1, 15.422121479641378: 1, 15.41745969623155: 1,
15.411711535195893: 1, 15.276321084051547: 1, 15.261391401453224: 1, 15.226581899666964: 1,
15.222956626791955: 1, 15.154113602583799: 1, 15.124651890607717: 1, 15.096312887605311: 1,
15.018425597306775: 1, 15.008891568178889: 1, 14.958423078137397: 1, 14.956405671288321: 1,
14.936159094191765: 1, 14.917111866966275: 1, 14.889245602699319: 1, 14.869327965411241: 1,
14.857841868942462: 1, 14.857740182270044: 1, 14.852039448052345: 1, 14.822728166440028: 1,
14.81000711106031: 1, 14.800876681623922: 1, 14.800161086289188: 1, 14.791315748294073: 1,
14.775113537027064: 1, 14.745557473608045: 1, 14.741894595786839: 1, 14.709217065968188: 1,
14.708127621049268: 1, 14.692545002674404: 1, 14.652144845792876: 1, 14.61563874108985: 1,
14.612268988197714: 1, 14.610532768968138: 1, 14.604659277990542: 1, 14.602948049894986: 1,
14.59526948517293: 1, 14.585130077572497: 1, 14.581702733227175: 1, 14.571709961016797: 1,
14.564666905194866: 1, 14.542343603420816: 1, 14.523059270555342: 1, 14.510739809196378: 1,
14.47085943903769: 1, 14.469886487794836: 1, 14.459896784817051: 1, 14.452456912274513: 1,
14.411622166824145: 1, 14.365874237872983: 1, 14.330870007211686: 1, 14.318176812966255: 1,
14.316844667231846: 1, 14.269697374983348: 1, 14.21168363800057: 1, 14.168791229520236: 1,
14.145755870305324: 1, 14.141620941839333: 1, 14.110060719227876: 1, 14.062619161147953: 1,
14.057358613969907: 1, 14.027566663102327: 1, 14.026065353429296: 1, 14.006365927765145: 1,
13.992791334357499: 1, 13.99107813250987: 1, 13.956238408624586: 1, 13.920272996039888: 1,
13.909895618807228: 1, 13.908005419761256: 1, 13.886159730055445: 1, 13.824583340147846: 1,
13.813708661144364: 1, 13.790628837139453: 1, 13.710560652100495: 1, 13.708333495185315: 1,
13.699991707221558: 1, 13.643427834633277: 1, 13.62064524793457: 1, 13.61161741271675: 1,
13.610006398673026: 1, 13.604456403005418: 1, 13.55741122121105: 1, 13.527316841692056: 1,
13.494912278364493: 1, 13.46797657967231: 1, 13.465714342273222: 1, 13.426284599780352: 1,
13.405960151209694: 1, 13.397514403662399: 1, 13.396492834315646: 1, 13.336565581283304: 1,
13.32516081542953: 1, 13.305127318550223: 1, 13.28071818710283: 1, 13.166601497593515: 1,
13.164321625305613: 1, 13.15257707902021: 1, 13.088656109445893: 1, 13.076576562542073: 1,
13.074178626037261: 1, 13.04965670688989: 1, 13.03153871484421: 1, 13.030644310039895: 1,
13.026694459055024: 1, 13.015277359163269: 1, 13.014614525622274: 1, 13.006873038231891: 1,
12.974897856985569: 1, 12.949249795565391: 1, 12.92658665493298: 1, 12.894457580736226: 1,
12.884338129733226: 1, 12.854258826413814: 1, 12.843965305112437: 1, 12.843018501294994: 1,
12.825620355309809: 1, 12.774853156495524: 1, 12.77059128383463: 1, 12.750584753084164: 1,
12.726589800876182: 1, 12.710708987760269: 1, 12.6934513622867: 1, 12.682224630173128: 1,
12.676266668011515: 1, 12.672085466903821: 1, 12.650371356385493: 1, 12.612942087394545: 1,
12.595228243267162: 1, 12.593892705464084: 1, 12.586439260234032: 1, 12.575568582590064: 1
```

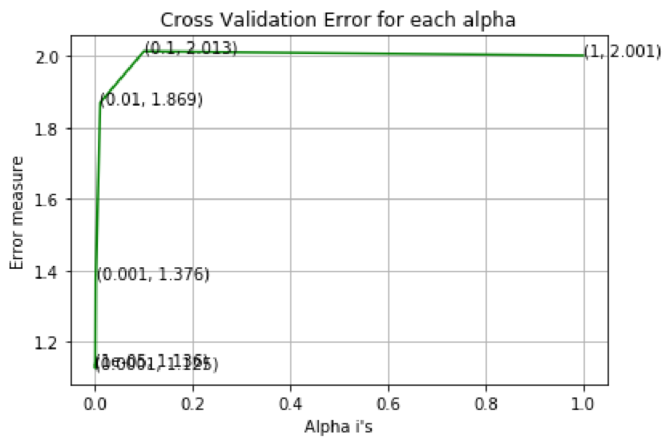
```
12.595226245267162: 1, 12.595892705464084: 1, 12.586459260254052: 1, 12.575506562590064: 1,
12.567508559091694: 1, 12.566772803179301: 1, 12.45761349326816: 1, 12.445598401593674: 1,
12.386356264192722: 1, 12.373621018540785: 1, 12.293264791943644: 1, 12.283633364735014: 1,
12.256274158180817: 1, 12.243414244858153: 1, 12.229205982715252: 1, 12.171624991286434: 1,
12.166917750628661: 1, 12.15089358340639: 1, 12.136749787099857: 1, 12.097752740444097: 1,
12.094970101029165: 1, 12.09408333349567: 1, 12.083209559092376: 1, 12.0717060805148: 1,
12.049170077421161: 1, 12.003033772470582: 1, 11.983548969134766: 1, 11.970154658859206: 1,
11.952450051277136: 1, 11.94792575806352: 1, 11.937146918939105: 1, 11.920011588854258: 1,
11.89449651514875: 1, 11.883698505929289: 1, 11.848257519860695: 1, 11.834080536653769: 1,
11.82127904562613: 1, 11.814670617712265: 1, 11.810913183883255: 1, 11.8067046324479: 1,
11.8027429413541: 1, 11.787917217086065: 1, 11.785793863661322: 1, 11.77982619051454: 1,
11.747468930926397: 1, 11.745052517815079: 1, 11.743018962398175: 1, 11.732887338421392: 1,
11.694211415483048: 1, 11.693810045458955: 1, 11.689021557740308: 1, 11.687995010179462: 1,
11.638598493709184: 1, 11.63179228752341: 1, 11.62103301870392: 1, 11.609498213711479: 1,
11.60542787128881: 1, 11.596835650639964: 1, 11.596048589322097: 1, 11.566877582352651: 1,
11.566332855965612: 1, 11.535973293160463: 1, 11.53594455528053: 1, 11.499259783445606: 1,
11.496853672951556: 1, 11.46998642527753: 1, 11.464800678618895: 1, 11.464055376397292: 1,
11.461154300187356: 1, 11.455142187074125: 1, 11.42192616971927: 1, 11.373016784119224: 1,
11.35492054223831: 1, 11.34262051020579: 1, 11.337414670055445: 1, 11.330915494228348: 1,
11.303353070686208: 1, 11.293595268415176: 1, 11.266846531439372: 1, 11.262339886079392: 1,
11.25498752271489: 1, 11.235275114605585: 1, 11.218081931006834: 1, 11.170270417185767: 1,
11.135978358952563: 1, 11.10478267926173: 1, 11.084707880158808: 1, 11.077053297515118: 1,
11.075494031923382: 1, 11.070705382981588: 1, 11.065628675813105: 1, 11.05005565795698: 1,
11.030975116040832: 1, 11.024980698026297: 1, 11.01301952742587: 1, 11.008296406377786: 1,
11.004028693513394: 1, 10.995057308732916: 1, 10.994750620544664: 1, 10.983914200506847: 1,
10.967007804417287: 1, 10.96183518417962: 1, 10.939521922209893: 1, 10.925356348388876: 1,
10.920916399873445: 1, 10.912538120875567: 1, 10.89750442999233: 1, 10.869876651873867: 1,
10.868216212379757: 1, 10.855798459571039: 1, 10.855406068448453: 1, 10.854054057107069: 1,
10.82139289240446: 1, 10.816409999462481: 1, 10.810030309131575: 1, 10.793860114430261: 1,
10.779193161462238: 1, 10.760458284847543: 1, 10.73516059811743: 1, 10.727994191906424: 1,
10.710708330005476: 1, 10.699238884679867: 1, 10.684744608924728: 1, 10.670743073884736: 1,
10.636577545084897: 1, 10.62759638335446: 1, 10.626090891143214: 1, 10.60895323874099: 1,
10.590075407993476: 1, 10.585928429816537: 1, 10.579803569622081: 1, 10.576995609328245: 1,
10.575582343554377: 1, 10.574472884975592: 1, 10.574448640507768: 1, 10.570510237031895: 1,
10.565419830404657: 1, 10.552648348380224: 1, 10.527960372051991: 1, 10.52521961197401: 1,
10.522387842619572: 1, 10.517386273517824: 1, 10.510898651711232: 1, 10.505872182244309: 1,
10.48893156214172: 1, 10.47718019316744: 1, 10.470377122318746: 1, 10.46794043863143: 1,
10.465544942287881: 1, 10.398838931036353: 1, 10.397782588426999: 1, 10.390713383314337: 1,
10.383619956043681: 1, 10.341849139452426: 1, 10.34028090908996: 1, 10.325165994321251: 1,
10.308170586746353: 1, 10.279537187700972: 1, 10.235767252595553: 1, 10.229078765464958: 1,
10.217952104438076: 1, 10.214114886823314: 1, 10.213055800446853: 1, 10.206345224894195: 1,
10.200663259145243: 1, 10.181506039453906: 1, 10.161870555448889: 1, 10.161625968223085: 1,
10.132823530930299: 1, 10.129473369560166: 1, 10.12605519507778: 1, 10.124287984803043: 1,
10.105325278703909: 1, 10.082619530737054: 1, 10.069154985336844: 1, 10.066854562632608: 1,
10.031928747876384: 1, 10.020302509731044: 1, 10.000677628913975: 1, 9.961427589928148: 1,
9.953270608825614: 1, 9.950474996238517: 1, 9.93554869628605: 1, 9.915192830888756: 1,
9.894740580293844: 1, 9.88154758276383: 1, 9.860795246712124: 1, 9.859183595952153: 1,
9.858166907540175: 1, 9.856779940667286: 1, 9.854126820578978: 1, 9.84524893852659: 1,
9.814226056106316: 1, 9.810158689816891: 1, 9.79119053411544: 1, 9.783598364399543: 1,
9.780757964143708: 1, 9.779877701542041: 1, 9.764557411862754: 1, 9.763372830553582: 1,
9.76040324827935: 1, 9.742030345283725: 1, 9.735940814699902: 1, 9.7103796043376: 1,
9.702959459003136: 1, 9.698266941597804: 1, 9.683752480267078: 1, 9.677187478664191: 1,
9.659130793273784: 1, 9.656317667747262: 1, 9.654288036309834: 1, 9.64877850884777: 1,
9.643700362612673: 1, 9.63909059668844: 1, 9.632586052259605: 1, 9.622808528141697: 1,
9.616486374699948: 1, 9.590981941828693: 1, 9.582163148179966: 1, 9.57977822475654: 1,
9.574280257257133: 1, 9.562721757327889: 1, 9.556673044548868: 1, 9.537258541707699: 1,
9.534990111716894: 1, 9.52908968824945: 1, 9.51485433789357: 1, 9.511294075490097: 1,
9.502411437895589: 1, 9.488922688484536: 1, 9.438217752522116: 1, 9.427596785347369: 1,
9.42650642722151: 1, 9.419117326883338: 1, 9.411568817380706: 1, 9.397786367075298: 1,
9.396641168852769: 1, 9.392116443606797: 1, 9.384202272374738: 1, 9.378455664985598: 1,
9.36918572640066: 1, 9.364568316531052: 1, 9.358854724639883: 1, 9.344569568958894: 1,
9.324441617056683: 1, 9.31654590135104: 1, 9.313425425499709: 1, 9.313415481349058: 1,
9.309092761425426: 1, 9.305068265655324: 1, 9.297313040864577: 1, 9.289616857931987: 1,
9.272338642274647: 1, 9.271956716487155: 1, 9.267649243421591: 1, 9.265158175414937: 1,
9.257751764746857: 1, 9.253311831223526: 1, 9.240780792101472: 1, 9.236778187265042: 1,
9.233058623932914: 1, 9.213999928926894: 1, 9.193599537582408: 1, 9.186019331198802: 1,
9.185108100109915: 1, 9.182804041497294: 1, 9.179008193766071: 1, 9.173089408420857: 1,
9.167209245068571: 1, 9.159409976389004: 1, 9.157212293130396: 1, 9.140107166638376: 1,
9.128153226941267: 1, 9.124051221726955: 1, 9.122515950008506: 1, 9.114514510813798: 1,
9.109301687373911: 1, 9.097007040035626: 1, 9.096056582700344: 1, 9.096005331170293: 1,
9.08789787194169: 1, 9.086719836325724: 1, 9.074806551516515: 1, 9.056702179083736: 1,
9.054530994470051: 1, 9.053840435674138: 1, 9.039560646289676: 1, 9.026870577928914: 1,
9.013373208253729: 1, 9.013112115761986: 1, 9.003349688331461: 1, 8.96056536340601: 1,
8.958208980474433: 1, 8.957206145844994: 1, 8.953607820417401: 1, 8.93937827900686: 1,
8.923306078932315: 1, 8.911316296009266: 1, 8.908867392120392: 1, 8.878878111157354: 1,
8.873703234731773: 1, 8.868102394562863: 1, 8.862110050860519: 1, 8.845575611616407: 1,
8.832409789838263: 1, 8.82598581198974: 1, 8.822570284312258: 1, 8.80541211938735: 1,
8.804206362336597: 1, 8.763072014439077: 1, 8.758094115140157: 1, 8.753981528624154: 1
```

```
8.804206562536597: 1, 8.765072014439077: 1, 8.758094115140157: 1, 8.755981528624154: 1,
8.723965292781045: 1, 8.710688250754995: 1, 8.70882649984189: 1, 8.708017089759721: 1,
8.698525970167314: 1, 8.685523001721153: 1, 8.678863525293782: 1, 8.667585216605683: 1,
8.661976801750885: 1, 8.660883054421085: 1, 8.660399333921996: 1, 8.659105210737863: 1,
8.61554661579925: 1, 8.61011196938408: 1, 8.607182492428194: 1, 8.605212791223183: 1,
8.591799120735944: 1, 8.58655231087693: 1, 8.568283374296353: 1, 8.5643918387877: 1,
8.553925622705398: 1, 8.542770517452817: 1, 8.49754742812819: 1, 8.489011259892822: 1,
8.468503135274782: 1, 8.46435887463019: 1, 8.424833146222715: 1, 8.420111579525003: 1,
8.403033902605806: 1, 8.392436883379395: 1, 8.369166285023317: 1, 8.354029566592304: 1,
8.34762591929544: 1, 8.343859267391307: 1, 8.342334448277676: 1, 8.341736974107173: 1,
8.334486231252274: 1, 8.326933706693197: 1, 8.320692194381207: 1, 8.314222678433968: 1,
8.31217326478885: 1, 8.310206169773423: 1, 8.30505619637743: 1, 8.294804584517076: 1,
8.29408436744205: 1, 8.281737013887666: 1, 8.27961014807162: 1, 8.269774757122514: 1,
8.259644576345917: 1, 8.257537166358885: 1, 8.23554701983026: 1, 8.224842443216474: 1,
8.208118549778673: 1, 8.203783068511681: 1, 8.202615722360813: 1, 8.199652275050509: 1,
8.165081999547494: 1, 8.161476502673436: 1, 8.159600309041965: 1, 8.142969374961218: 1,
8.1095390220497: 1, 8.09515417420084: 1, 8.08845193537861: 1, 8.087716328423433: 1,
8.068167043680262: 1, 8.041730171404845: 1, 7.986600119674676: 1, 7.975185916681081: 1,
7.955721345334025: 1, 7.946197062516309: 1, 7.942791771862014: 1, 7.938719864455899: 1,
7.9077296275621425: 1, 7.904327165853381: 1, 7.897582388878529: 1, 7.897207049303406: 1, 7.89195855
2193687: 1, 7.89119752493764: 1, 7.875666795821693: 1, 7.871108594006099: 1, 7.863005573306056: 1,
7.855141330500171: 1, 7.842815618188009: 1, 7.837081863175913: 1, 7.834456863848323: 1,
7.827211649057149: 1, 7.799294343153897: 1, 7.7907140705797655: 1, 7.7621337633385945: 1,
7.751589843554131: 1, 7.748887467525433: 1, 7.74405607681685: 1, 7.735694976545994: 1,
7.734245656160616: 1, 7.7328166795776445: 1, 7.692723920433035: 1, 7.689241343350648: 1,
7.684091067374121: 1, 7.6765625096261605: 1, 7.674363848634961: 1, 7.665606946041685: 1,
7.661971873548956: 1, 7.6543724154486075: 1, 7.637223004225649: 1, 7.624801391437398: 1,
7.622316113457308: 1, 7.619982308526707: 1, 7.619313790642734: 1, 7.61538204872617: 1,
7.614001027426081: 1, 7.604123796926362: 1, 7.596191505264293: 1, 7.590638694969556: 1,
7.582898905355604: 1, 7.57445774334645: 1, 7.560247032068987: 1, 7.559681179623869: 1,
7.557523860902965: 1, 7.489448525109734: 1, 7.482966887968842: 1, 7.4396060927742065: 1,
7.427814966819331: 1, 7.407021408921912: 1, 7.382191148974232: 1, 7.324814714815561: 1,
7.317274283458695: 1, 7.314379314808395: 1, 7.30414328897001: 1, 7.294750385840536: 1,
7.286490767189511: 1, 7.25104505037087: 1, 7.247179884148973: 1, 7.207187917167306: 1,
7.191959123986742: 1, 7.145637970489036: 1, 7.098645516251767: 1, 7.072165053751391: 1,
7.053386069239939: 1, 7.041461797538944: 1, 7.038423431964305: 1, 7.031679246576867: 1,
7.017106930240964: 1, 7.0029582327719115: 1, 6.986300203261952: 1, 6.97441660185079: 1,
6.9120820462521655: 1, 6.893892940734768: 1, 6.889877613349821: 1, 6.8858036479831695: 1,
6.847016115876277: 1, 6.839384449012169: 1, 6.8386909868285395: 1, 6.777495102036823: 1,
6.752736932497842: 1, 6.744156412226813: 1, 6.674516476388444: 1, 6.6711227081673155: 1,
6.625205831059321: 1, 6.610592173619241: 1, 6.552411234968304: 1, 6.511728469611274: 1,
6.4640720195831145: 1, 6.391127254044778: 1, 6.390902615067136: 1})
```

In [50]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1360866028181156
For values of alpha =  0.0001 The log loss is: 1.1249219404944952
For values of alpha =  0.001 The log loss is: 1.3763769627604154
For values of alpha =  0.01 The log loss is: 1.8691626366703595
For values of alpha =  0.1 The log loss is: 2.0134291536988025
For values of alpha =  1 The log loss is: 2.0013874617404492
```



```
For values of best alpha =  0.0001 The train log loss is: 0.8502630864207521
For values of best alpha =  0.0001 The cross validation log loss is: 1.1249219404944952
For values of best alpha =  0.0001 The test log loss is: 1.0892717130310272
```

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

Ans. Yes, it seems like!

In [51]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(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 [52]:

```
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")
```

```
93.8 % of word of test data appeared in train data
93.9 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [54]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [55]:

```
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 [56]:

```
# 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(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-269]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)-269]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_r
o))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)-269]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

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

Stacking the four types of features

In [57]:

```
# merging gene, variance and text features

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

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

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


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

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

In [58]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 6444)
```

```
(number of data points * number of features) in train data =  (2124, 6444)
(number of data points * number of features) in test data =  (665, 6444)
(number of data points * number of features) in cross validation data = (532, 6444)
```

In [59]:

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

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

4.1. Base Line Model

4.1.1. Naive Bayes

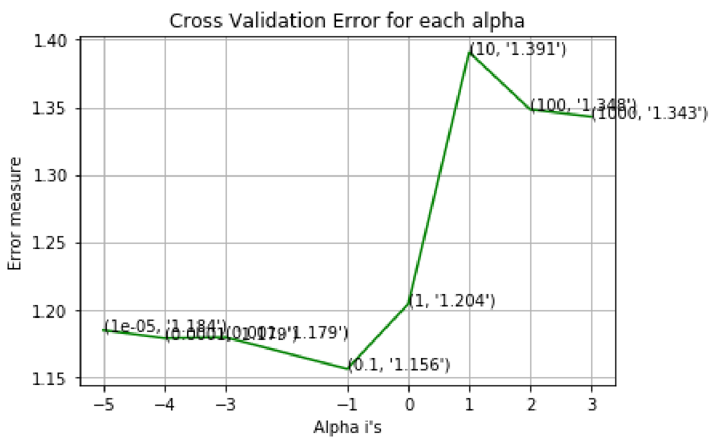
4.1.1.1. Hyper parameter tuning

In [60]:

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

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


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


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

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


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


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

# Variables that will be used in the end to make comparison table of all models
nb_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=
1e-15)
nb_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
nb_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-
15)
```

```
for alpha = 1e-05
Log Loss : 1.1844821834342323
for alpha = 0.0001
Log Loss : 1.1785597339594607
for alpha = 0.001
Log Loss : 1.1792846224069855
for alpha = 0.1
Log Loss : 1.1557199838533636
for alpha = 1
Log Loss : 1.203926358930577
for alpha = 10
Log Loss : 1.3907871580302045
for alpha = 100
Log Loss : 1.3483232993543104
for alpha = 1000
Log Loss : 1.3429617762614663
```



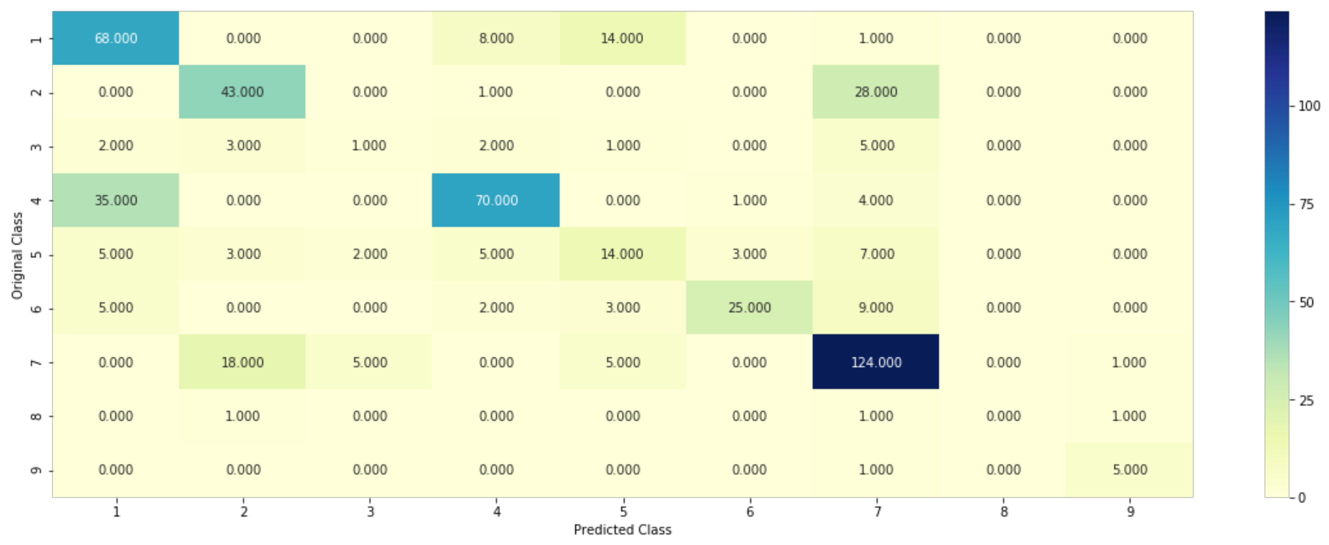
```
For values of best alpha =  0.1 The train log loss is: 0.7553171685507521
For values of best alpha =  0.1 The cross validation log loss is: 1.1557199838533636
For values of best alpha =  0.1 The test log loss is: 1.211123107331458
```

4.1.1.2. Testing the model with best hyper paramters

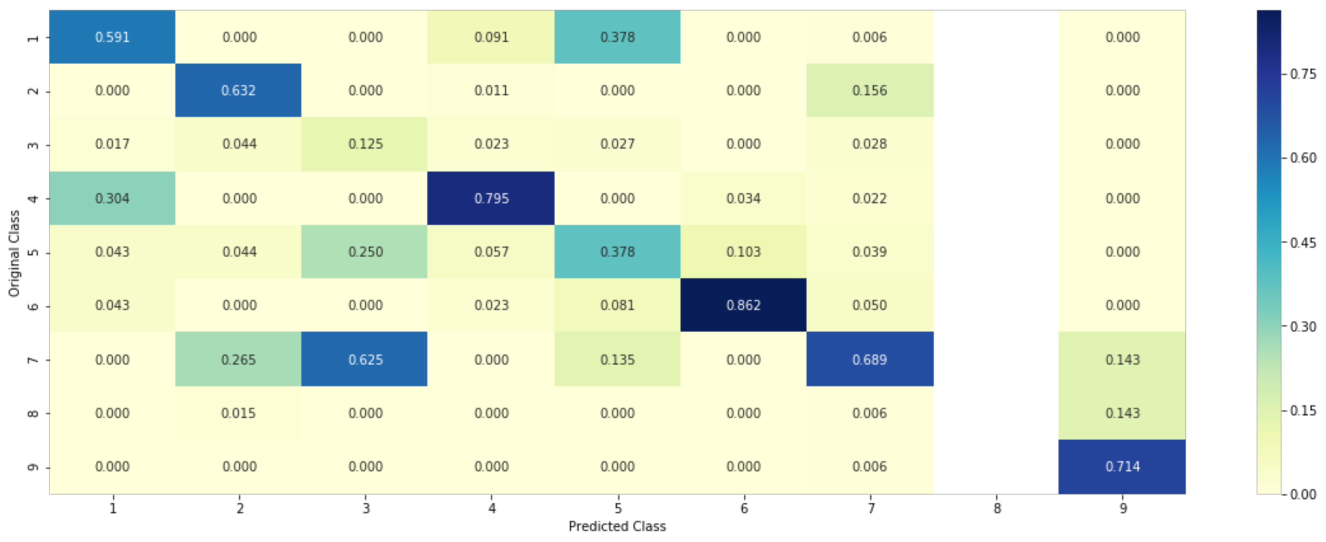
In [61]:

```
# 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()))

# Variables that will be used in the end to make comparison table of models
nb_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

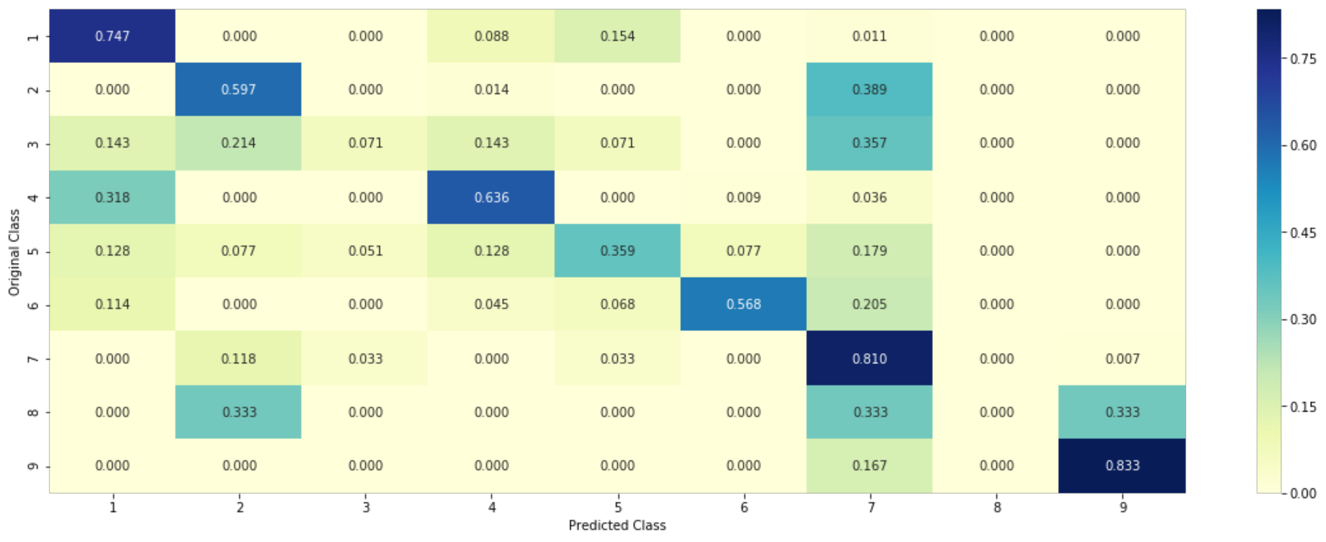
```
Log Loss : 1.1557199838533636
Number of missclassified point : 0.34210526315789475
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [62]:

```
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 : 4
Predicted Class Probabilities: [[0.0452 0.0503 0.0157 0.7427 0.0362 0.0329 0.0673 0.0051 0.0045]]
Actual Class : 4
--------------------------------------------------
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [63]:

```
test_point_index = 100
no_feature = 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: [[0.0553 0.0586 0.0172 0.1348 0.0398 0.036  0.6478 0.0056 0.005 ]]
Actual Class : 7
--------------------------------------------------
```

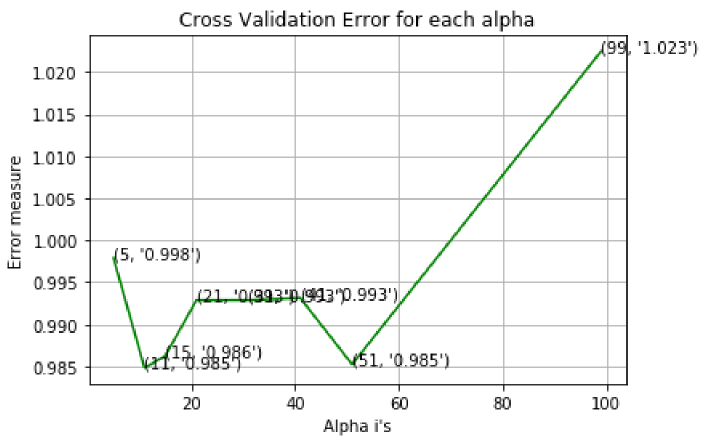
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [64]:

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

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


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


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

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


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

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

# Variables that will be used in the end to make comparison table of all models
knn_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labels=clf.classes_, e
ps=1e-15)
knn_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.classes_, eps=1e-15)
knn_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=clf.classes_, eps=
1e-15)
```

```
for alpha = 5
Log Loss : 0.9979014147881387
for alpha = 11
Log Loss : 0.9848064716006467
for alpha = 15
Log Loss : 0.9862591719994181
for alpha = 21
Log Loss : 0.9928786384103943
for alpha = 31
Log Loss : 0.9928780952661017
for alpha = 41
Log Loss : 0.9931327968328574
for alpha = 51
Log Loss : 0.9852313848005757
for alpha = 99
Log Loss : 1.0225071668153707
```



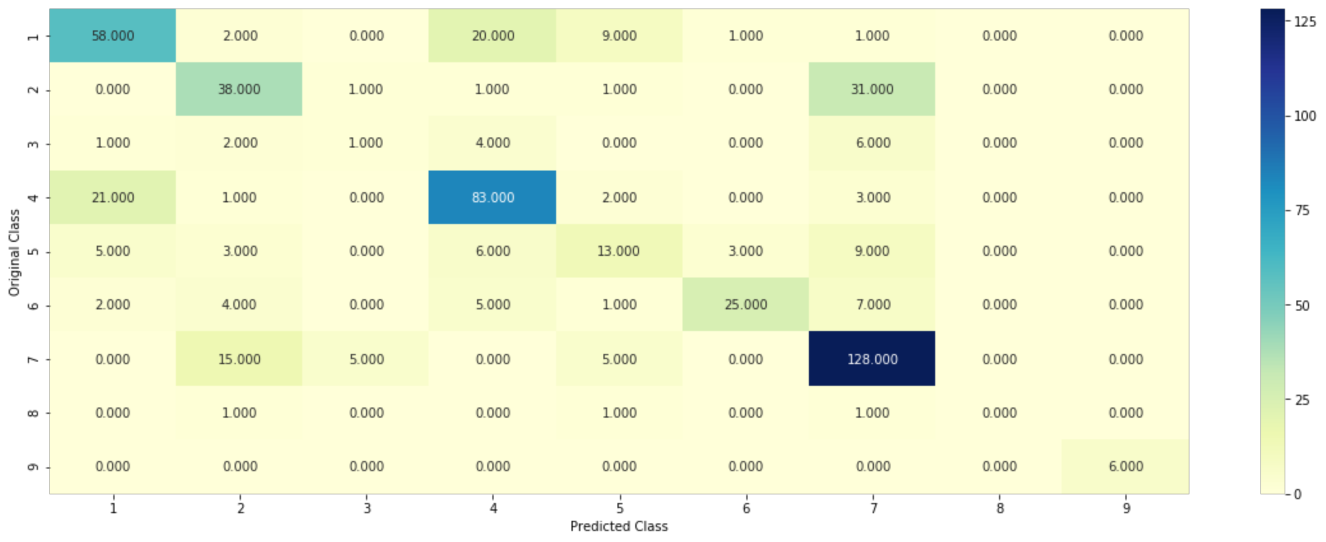
```
For values of best alpha =  11 The train log loss is: 0.6390189098519682
For values of best alpha =  11 The cross validation log loss is: 0.9848064716006467
For values of best alpha =  11 The test log loss is: 1.072404233689416б
```

4.2.2. Testing the model with best hyper paramters

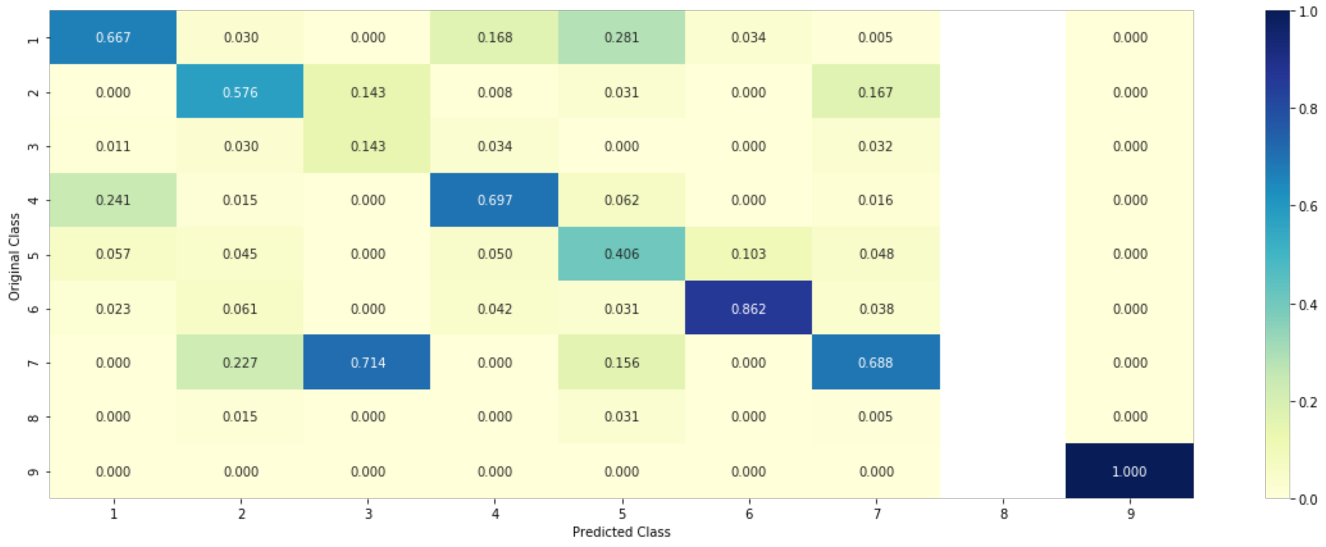
In [65]:

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

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

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

# Variables that will be used in the end to make comparison table of models
knn_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y))/cv_y.shape[0])
*100
```

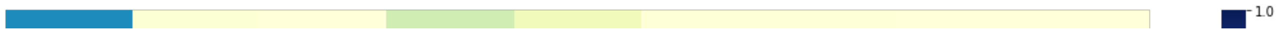
```
Log loss : 0.9848064716006467
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```

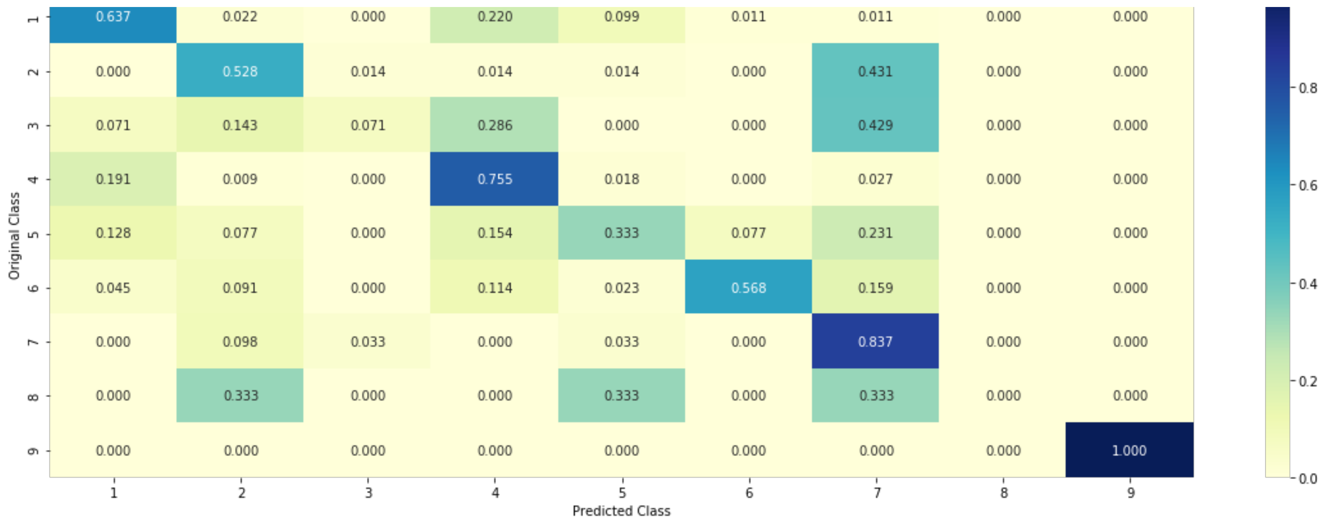


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.2.3.Sample Query point -1

In [66]:

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_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 : 2
Actual Class : 4
The  11  nearest neighbours of the test points belongs to classes [4 4 4 4 4 4 4 3 4 4 4]
Fequency of nearest points : Counter({4: 10, 3: 1})
```

4.2.4. Sample Query Point-2

In [67]:

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

test_point_index = 100

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

```
Predicted Class : 4
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [7 4 4
4 5 7 4 3 4 4 1]
Fequency of nearest points : Counter({4: 6, 7: 2, 5: 1, 3: 1, 1: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

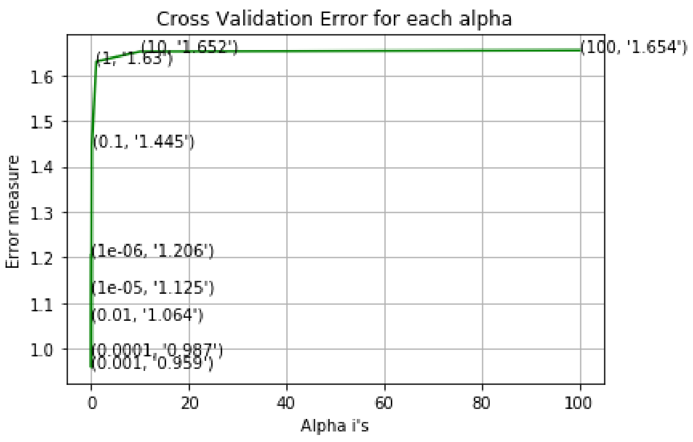
4.3.1.1. Hyper paramter tuning

In [68]:

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

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

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



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

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

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


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

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

# Variables that will be used in the end to make comparison table of all models
lr_balance_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=1e-15)
lr_balance_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
lr_balance_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-15)
```

```
for alpha = 1e-06
Log Loss : 1.2055676669059574
for alpha = 1e-05
Log Loss : 1.1245366139065447
for alpha = 0.0001
Log Loss : 0.9874522582420813
for alpha = 0.001
Log Loss : 0.9589334993636797
for alpha = 0.01
Log Loss : 1.063838279821865
for alpha = 0.1
Log Loss : 1.4453368852040585
for alpha = 1
Log Loss : 1.6296984956708118
for alpha = 10
Log Loss : 1.651760156831469
for alpha = 100
Log Loss : 1.6543871569870159
```



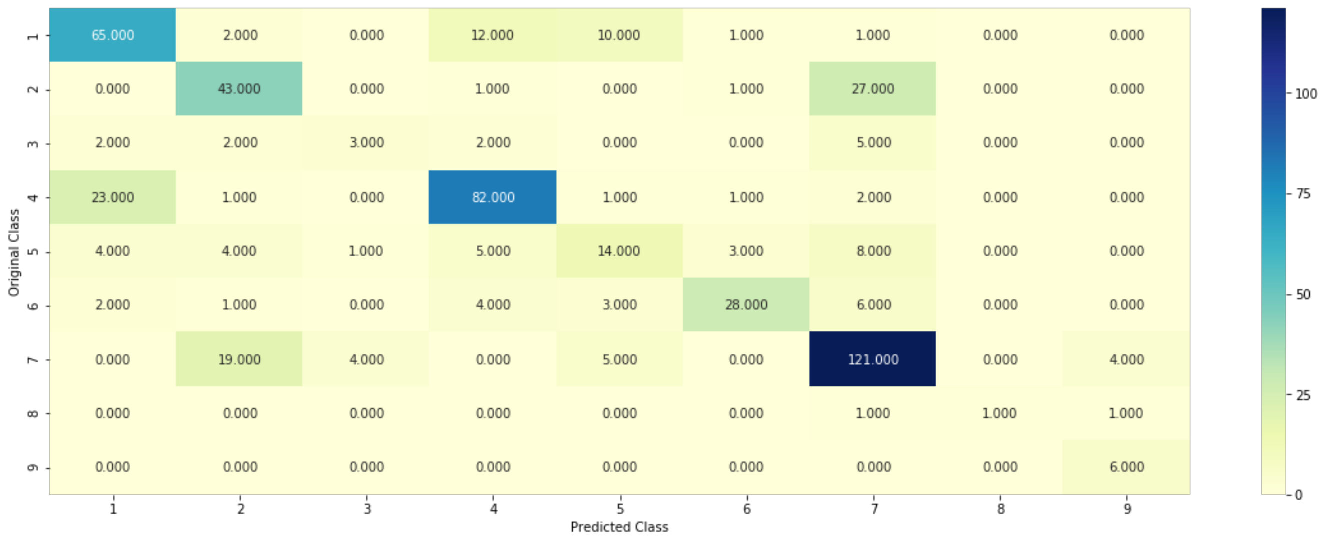
```
For values of best alpha =  0.001 The train log loss is: 0.6148709086017381
For values of best alpha =  0.001 The cross validation log loss is: 0.9589334993636797
For values of best alpha =  0.001 The test log loss is: 0.9221464647747004
```

4.3.1.2. Testing the model with best hyper paramters

In [69]:

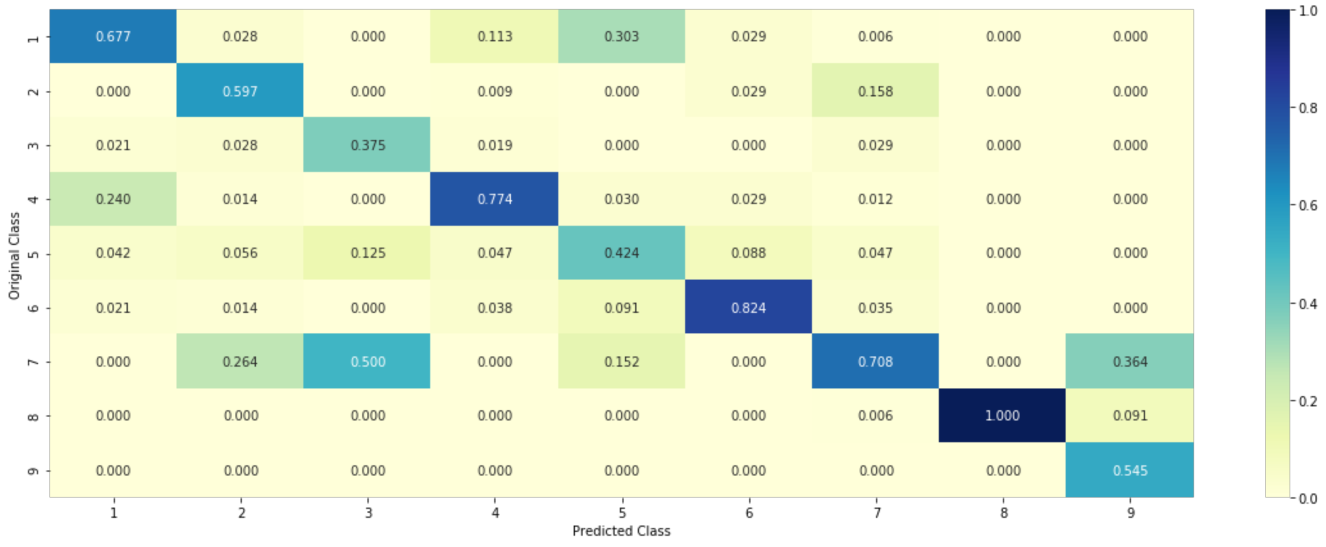
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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-in
tuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

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

# Variables that will be used in the end to make comparison table of models
lr_balance_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

```
Log loss : 0.9589334993636797
Number of mis-classified points : 0.3176691729323308
-------------------- Confusion matrix --------------------
```



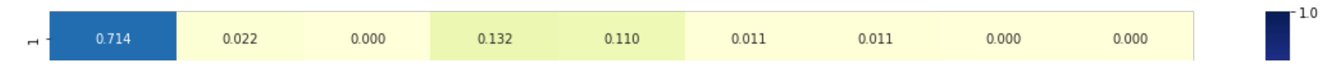
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	65.000	2.000	0.000	12.000	10.000	1.000	1.000	0.000	0.000
2	0.000	43.000	0.000	1.000	0.000	1.000	27.000	0.000	0.000
3	2.000	2.000	3.000	2.000	0.000	0.000	5.000	0.000	0.000
4	23.000	1.000	0.000	82.000	1.000	1.000	2.000	0.000	0.000
5	4.000	4.000	1.000	5.000	14.000	3.000	8.000	0.000	0.000
6	2.000	1.000	0.000	4.000	3.000	28.000	6.000	0.000	0.000
7	0.000	19.000	4.000	0.000	5.000	0.000	121.000	0.000	4.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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

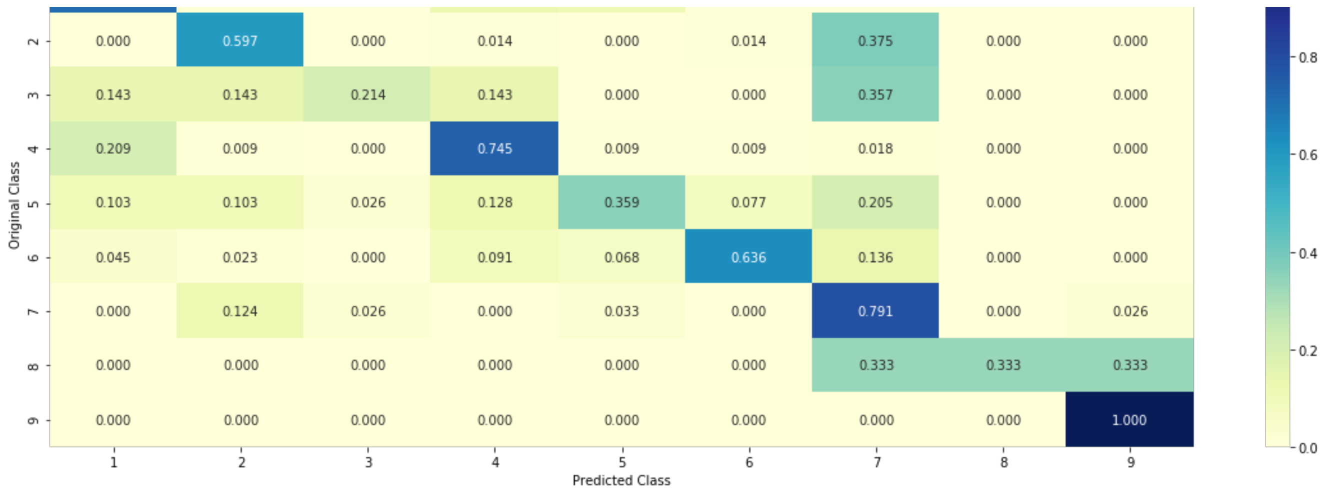


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.677	0.028	0.000	0.113	0.303	0.029	0.006	0.000	0.000
2	0.000	0.597	0.000	0.009	0.000	0.029	0.158	0.000	0.000
3	0.021	0.028	0.375	0.019	0.000	0.000	0.029	0.000	0.000
4	0.240	0.014	0.000	0.774	0.030	0.029	0.012	0.000	0.000
5	0.042	0.056	0.125	0.047	0.424	0.088	0.047	0.000	0.000
6	0.021	0.014	0.000	0.038	0.091	0.824	0.035	0.000	0.000
7	0.000	0.264	0.500	0.000	0.152	0.000	0.708	0.000	0.364
8	0.000	0.000	0.000	0.000	0.000	0.000	0.006	1.000	0.091
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.545

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



	1	2	3	4	5	6	7	8	9
1	0.714	0.022	0.000	0.132	0.110	0.011	0.011	0.000	0.000



4.3.1.3. Feature Importance

In [70]:

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

4.3.1.3.1. Correctly Classified point

In [71]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[2.800e-03 9.000e-04 6.810e-02 9.211e-01 3.000e-03 1.200e-03 3.000
e-04
  2.400e-03 2.000e-04]]
Actual Class : 4
--------------------------------------------------
```

4.3.1.3.2. Incorrectly Classified point

In [72]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_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: [[0.1488 0.0937 0.0149 0.2783 0.0425 0.0428 0.3664 0.0074 0.0052]]
Actual Class : 7
--------------------------------------------------
```

4.3.2. Without Class balancing

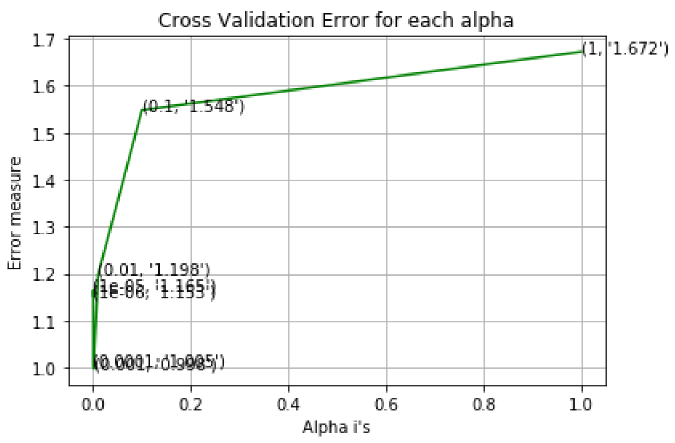
4.3.2.1. Hyper paramter tuning

In [73]:

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

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

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



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

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

fig, ax = plt.subplots()
```

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


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

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

# Variables that will be used in the end to make comparison table of all models
lr_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=
1e-15)
lr_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
lr_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-
15)
```

```
for alpha = 1e-06
Log Loss : 1.153214726979856
for alpha = 1e-05
Log Loss : 1.16516635676725
for alpha = 0.0001
Log Loss : 1.0047722122935654
for alpha = 0.001
Log Loss : 0.9977573229948815
for alpha = 0.01
Log Loss : 1.1983231512010426
for alpha = 0.1
Log Loss : 1.5480337154212096
for alpha = 1
Log Loss : 1.6719760227821325
```



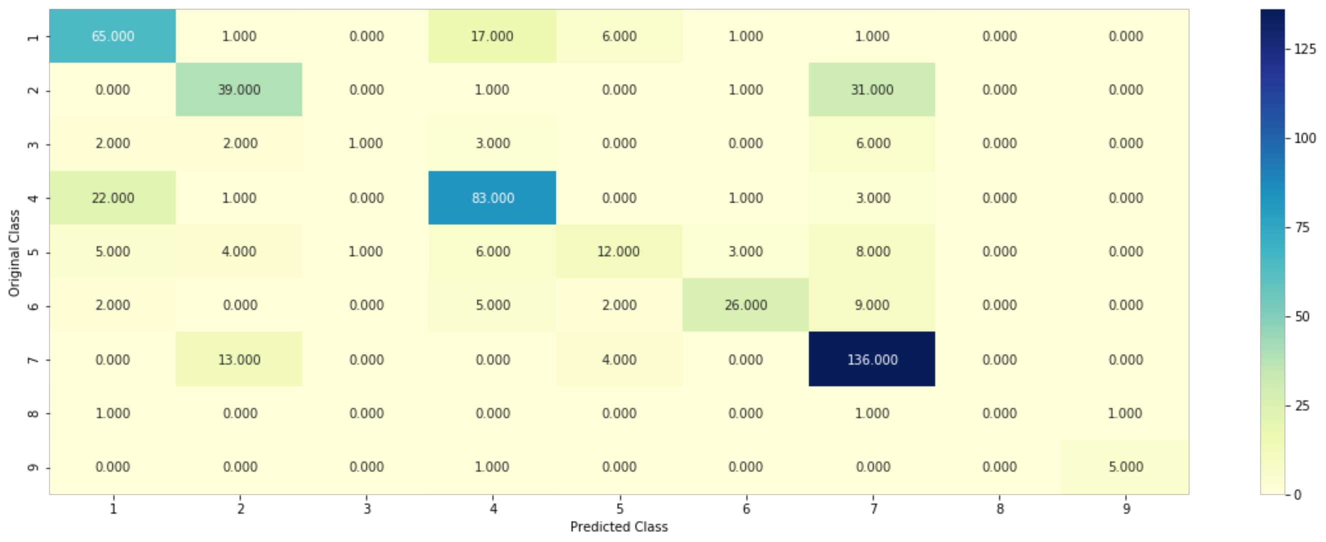
```
For values of best alpha =  0.001 The train log loss is: 0.6137349678129014
For values of best alpha =  0.001 The cross validation log loss is: 0.9977573229948815
For values of best alpha =  0.001 The test log loss is: 0.9563443420319919
```

4.3.2.2. Testing model with best hyper parameters

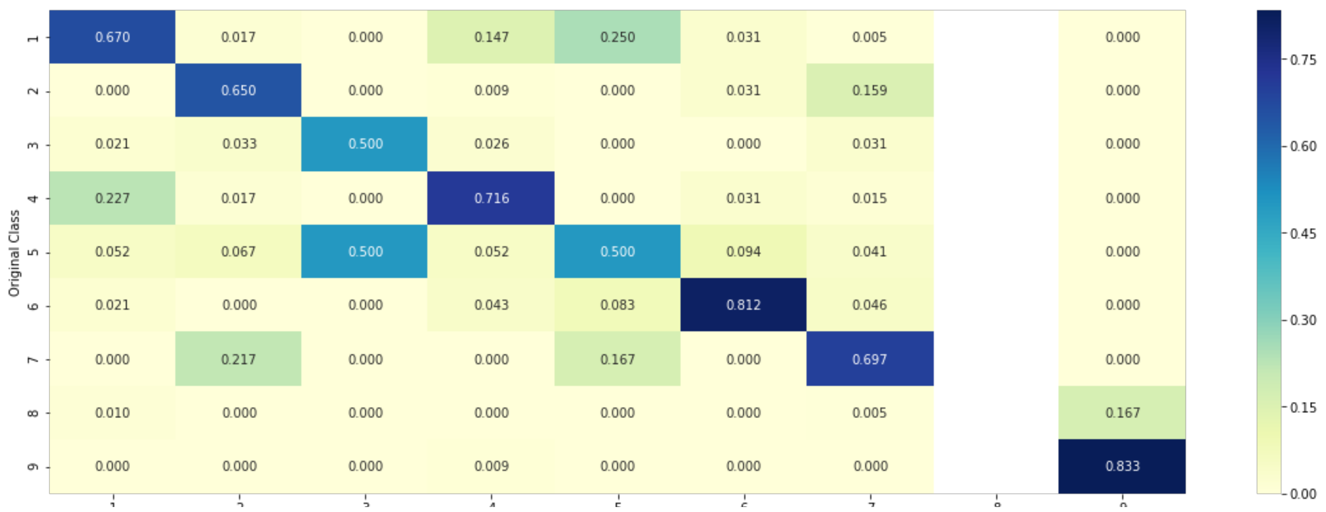
In [74]:

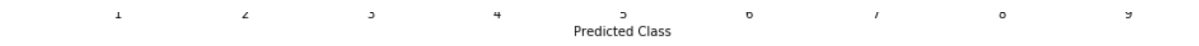
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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

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

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

# Variables that will be used in the end to make comparison table of models
lr_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

```
Log loss : 0.9977573229948815
Number of mis-classified points : 0.3101503759398496
-------------------- Confusion matrix --------------------
```

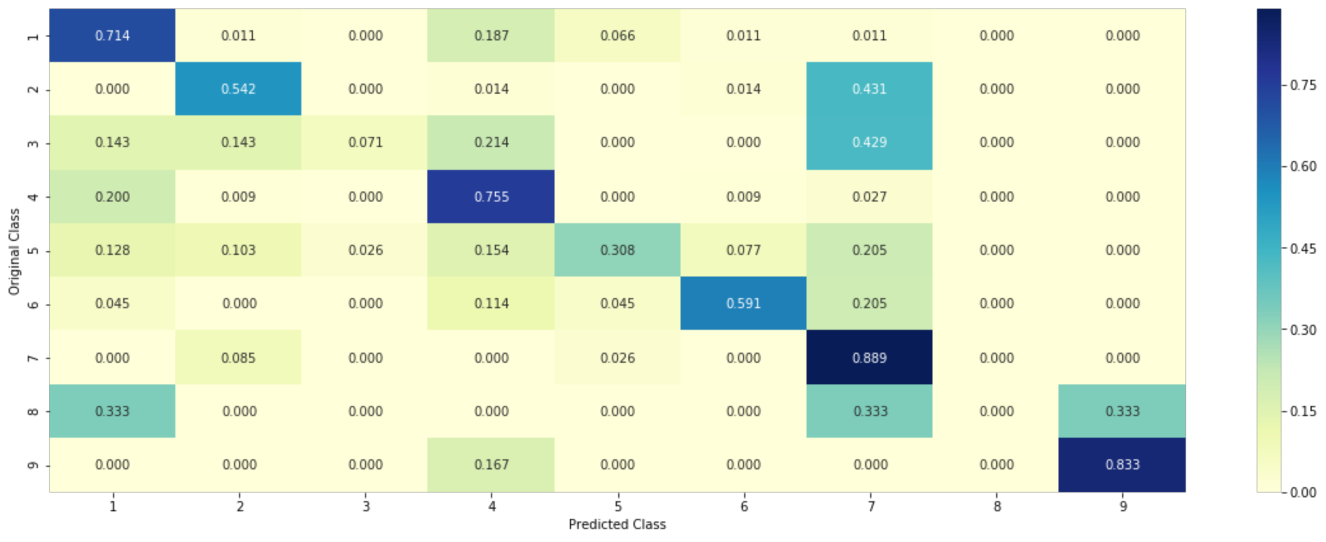


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.2.3. Feature Importance, Correctly Classified point

In [75]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', 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 : 4
Predicted Class Probabilities: [[3.600e-03 1.000e-03 1.770e-02 9.693e-01 2.900e-03 1.200e-03 4.000
e-04
  3.900e-03 1.000e-04]]
Actual Class : 4
--------------------------------------------------
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [76]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_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: [[0.1437 0.0937 0.0164 0.2446 0.0433 0.0429 0.402  0.0085 0.0049]]
Actual Class : 7
--------------------------------------------------
```

4.4. Linear Support Vector Machines

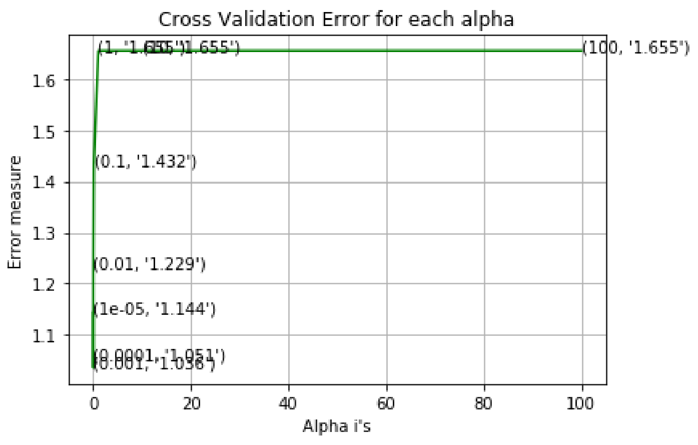
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [77]:

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

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

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



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

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

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


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

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

# Variables that will be used in the end to make comparison table of all models
svm_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps
=1e-15)
svm_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
svm_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e
-15)
```

```
for C = 1e-05
Log Loss : 1.1437526030765588
for C = 0.0001
Log Loss : 1.0506227032145152
for C = 0.001
Log Loss : 1.0356022297353413
for C = 0.01
Log Loss : 1.2293260789367166
for C = 0.1
Log Loss : 1.4321497566890853
for C = 1
Log Loss : 1.6551438944177297
for C = 10
Log Loss : 1.6551440325956068
for C = 100
Log Loss : 1.6551440743843642
```



```
For values of best alpha =  0.001 The train log loss is: 0.5406525457948057
For values of best alpha =  0.001 The cross validation log loss is: 1.0356022297353413
For values of best alpha =  0.001 The test log loss is: 0.9781375569874082
```

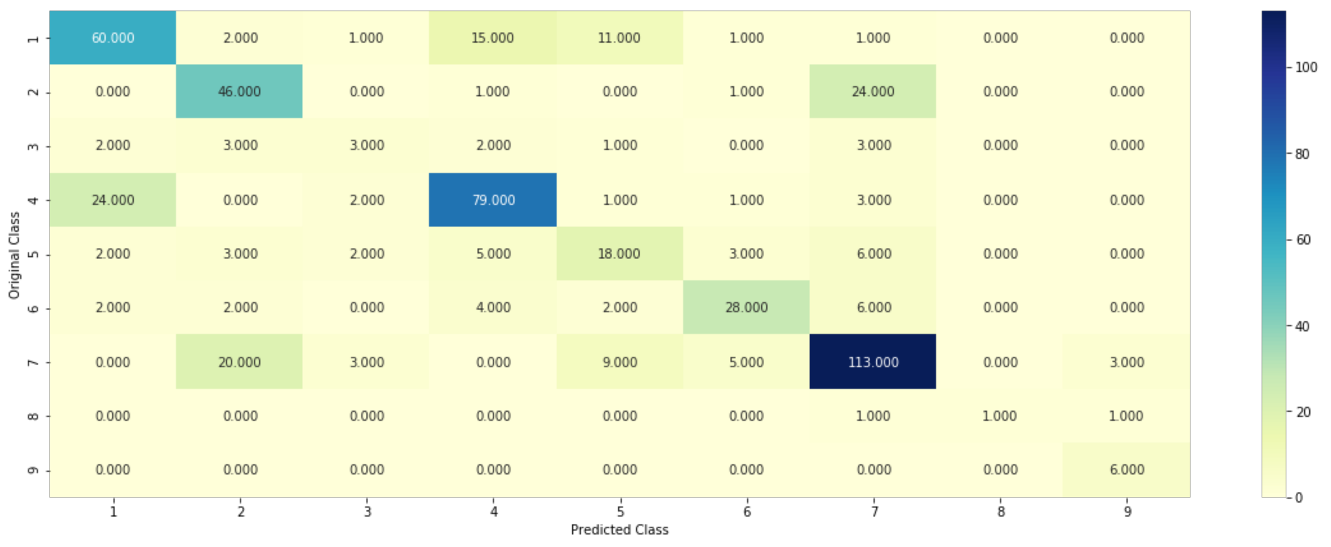
4.4.2. Testing model with best hyper parameters

In [78]:

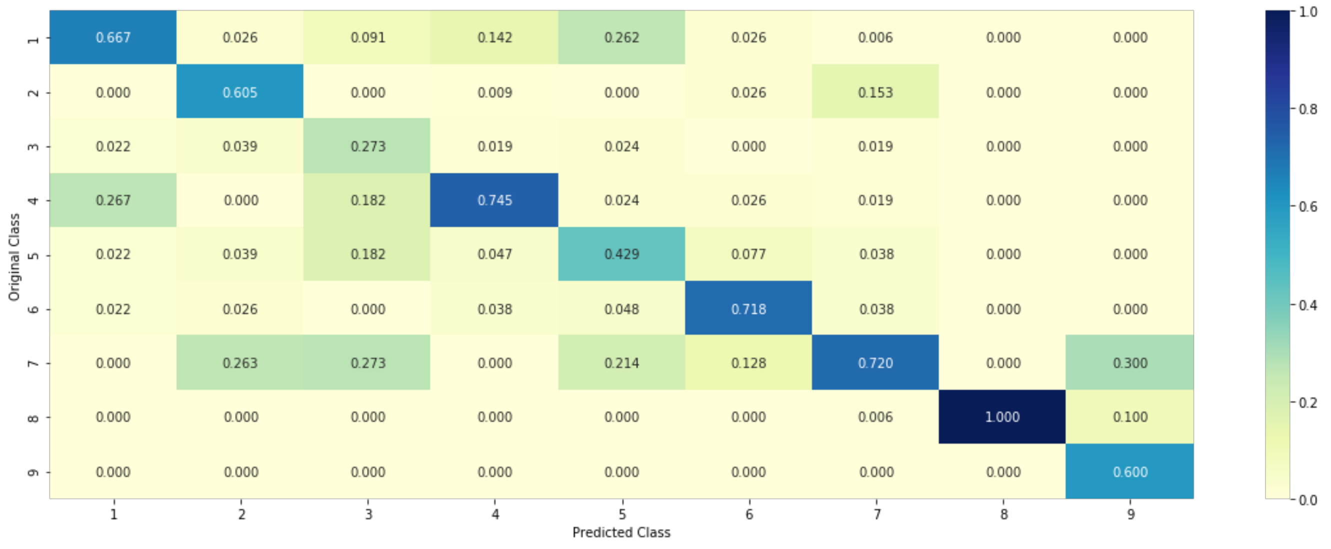
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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

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


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

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

# Variables that will be used in the end to make comparison table of models
svm_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])*1
00
```

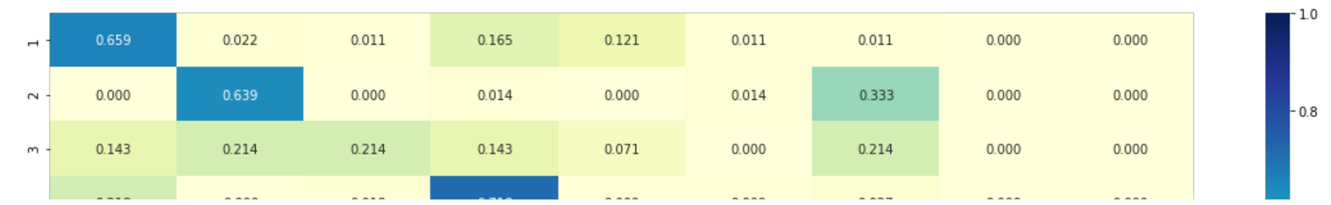
```
Log loss : 1.0356022297353413
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

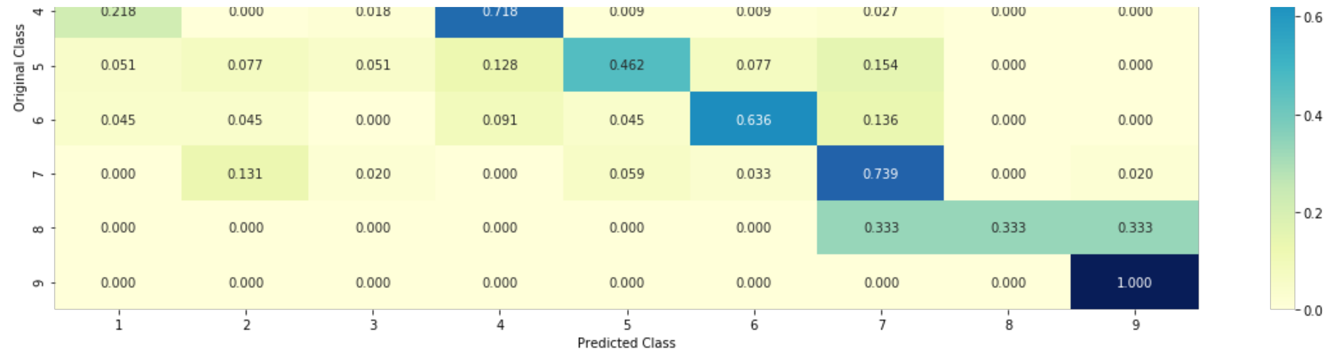


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [79]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0217 0.0196 0.0665 0.8317 0.0156 0.0058 0.0339 0.0032 0.002 ]]
Actual Class : 4
--------------------------------------------------
```

4.3.3.2. For Incorrectly classified point

In [80]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_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: [[0.1077 0.0874 0.018  0.1735 0.0459 0.026  0.5305 0.0054 0.0056]]
Actual Class : 7
--------------------------------------------------
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [81]:

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

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

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

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


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

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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

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

# Variables that will be used in the end to make comparison table of all models
rf_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=
1e-15)
rf_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
rf_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-
15)
```

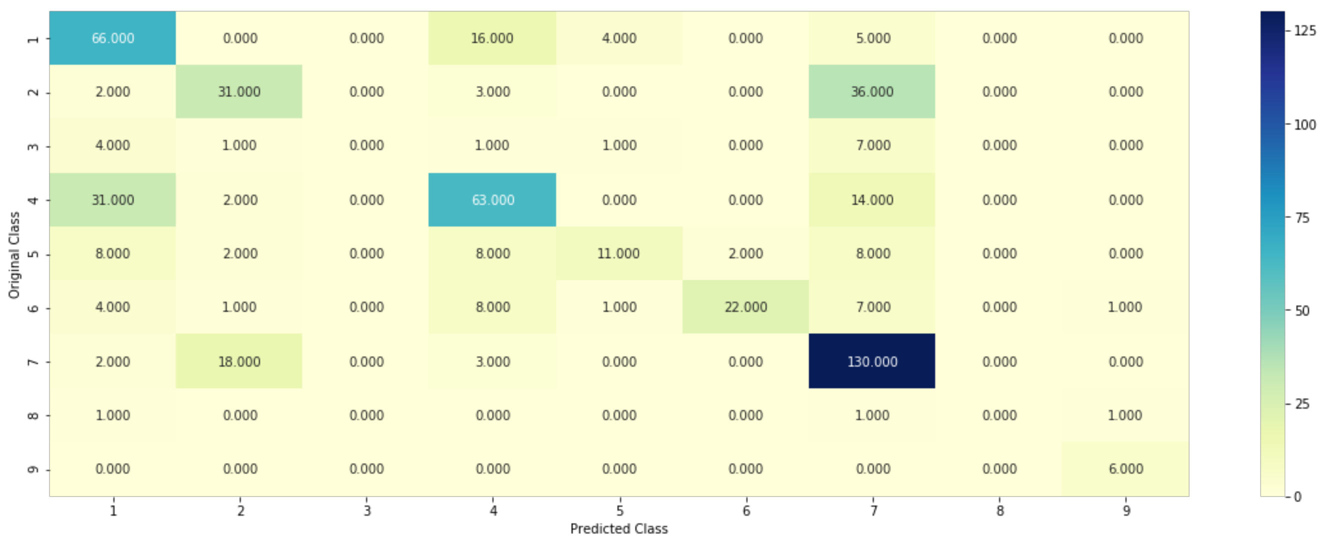
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1730070609715588
for n_estimators = 100 and max depth =  10
Log Loss : 1.2004024288067399
for n_estimators = 200 and max depth =  5
Log Loss : 1.1542467348809176
for n_estimators = 200 and max depth =  10
Log Loss : 1.1865338242586476
for n_estimators = 500 and max depth =  5
Log Loss : 1.1542148294959045
for n_estimators = 500 and max depth =  10
Log Loss : 1.1806470840892447
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1526005916539268
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1791171333276258
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1522167376194856
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1763121080208225
For values of best estimator =  2000 The train log loss is: 0.9033388433412035
For values of best estimator =  2000 The cross validation log loss is: 1.1522167376194856
For values of best estimator =  2000 The test log loss is: 1.1976185026027777
```

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

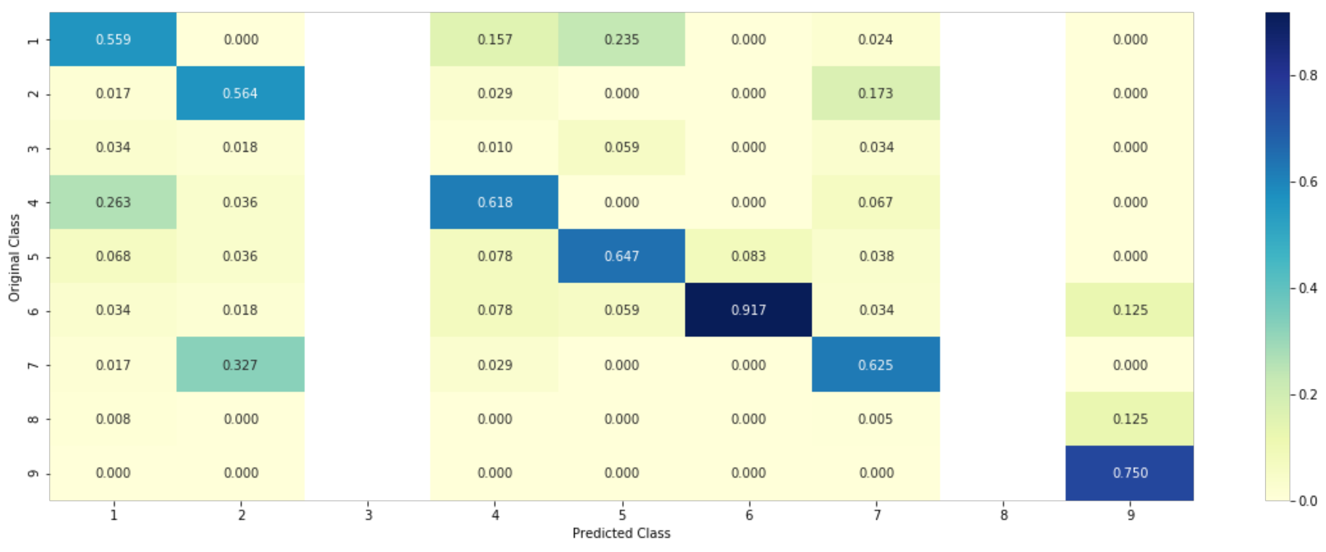
In [82]:

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

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

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

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

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)

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

# Variables that will be used in the end to make comparison table of models
rf_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

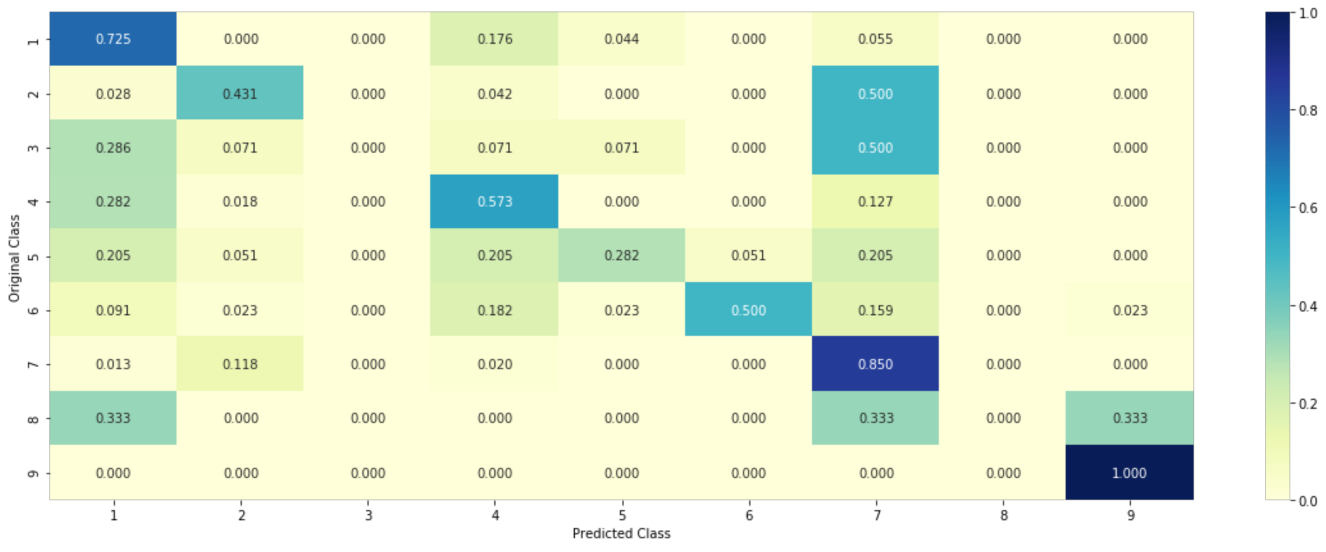
```
Log loss : 1.1522167376194856
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [83]:

```
# 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 : 4
Predicted Class Probabilities: [[3.690e-02 6.000e-03 6.450e-02 8.258e-01 2.850e-02 2.040e-02 1.590
e-02
  1.800e-03 2.000e-04]]
Actual Class : 4
--------------------------------------------------
```

4.5.3.2. Inorrectly Classified point

In [84]:

```
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 : 4
Predicted Class Probabilities: [[0.1959 0.1359 0.0238 0.2751 0.0635 0.0587 0.2305 0.0082 0.0083]]
Actuall Class : 7
--------------------------------------------------
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [85]:

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

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

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

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


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

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

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

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

# Variables that will be used in the end to make comparison table of all models
rf_response_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding),
labels=clf.classes_, eps=1e-15)
rf_response_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.classes_, ep
s=1e-15)
rf_response_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=clf.classe
s_, eps=1e-15)
```

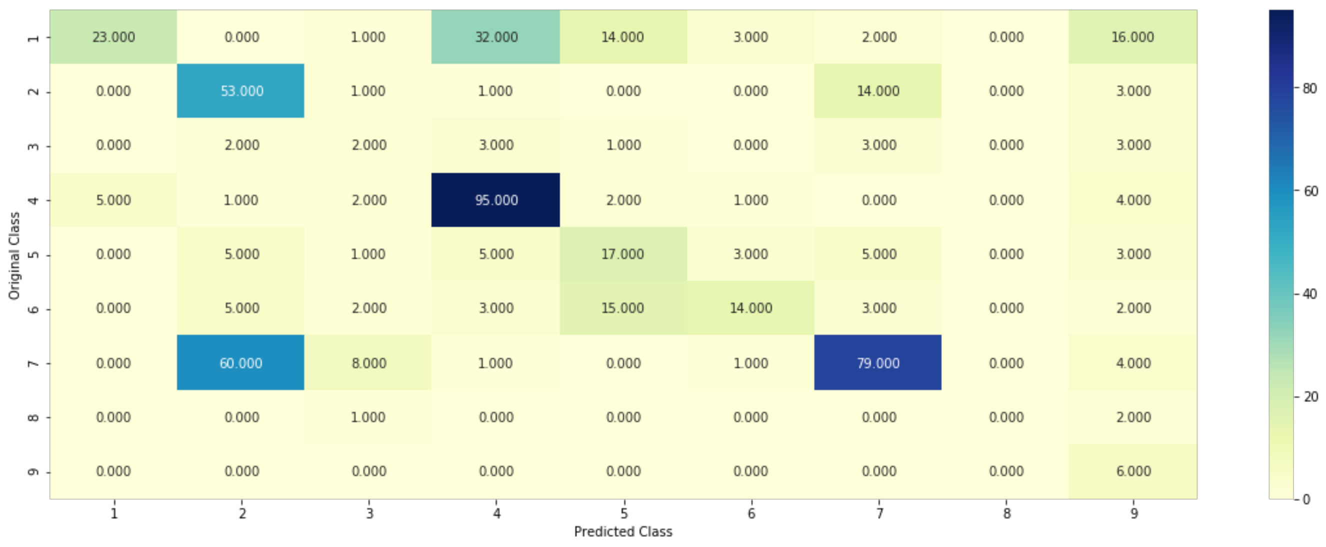
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.142177114133496
for n_estimators = 10 and max depth =  3
Log Loss : 1.6420089009526253
for n_estimators = 10 and max depth =  5
Log Loss : 1.52366056312118 6
for n_estimators = 10 and max depth =  10
Log Loss : 2.144960680066069
for n_estimators = 50 and max depth =  2
Log Loss : 1.6100012647206625
for n_estimators = 50 and max depth =  3
Log Loss : 1.3647180507449557
for n_estimators = 50 and max depth =  5
Log Loss : 1.4370313056414101
for n_estimators = 50 and max depth =  10
Log Loss : 2.000261533630471
for n_estimators = 100 and max depth =  2
Log Loss : 1.4881477913250116
for n_estimators = 100 and max depth =  3
Log Loss : 1.3963652127856123
for n_estimators = 100 and max depth =  5
Log Loss : 1.400511905946684
for n_estimators = 100 and max depth =  10
Log Loss : 1.871373968601297
for n_estimators = 200 and max depth =  2
Log Loss : 1.4797267588799017
for n_estimators = 200 and max depth =  3
Log Loss : 1.4238279827066793
for n_estimators = 200 and max depth =  5
Log Loss : 1.3999384078529051
for n_estimators = 200 and max depth =  10
Log Loss : 1.8287028163667884
for n_estimators = 500 and max depth =  2
Log Loss : 1.6005410755997311
for n_estimators = 500 and max depth =  3
Log Loss : 1.5056813419768533
for n_estimators = 500 and max depth =  5
Log Loss : 1.4222027079134192
for n_estimators = 500 and max depth =  10
Log Loss : 1.8404239817979255
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5699453738596776
for n_estimators = 1000 and max depth =  3
Log Loss : 1.508236246250369
for n_estimators = 1000 and max depth =  5
Log Loss : 1.409801037962935
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8269750556886937
For values of best alpha =  50 The train log loss is: 0.13992511623893156
For values of best alpha =  50 The cross validation log loss is: 1.3647180507449557
For values of best alpha =  50 The test log loss is: 1.5168480576687147
```

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

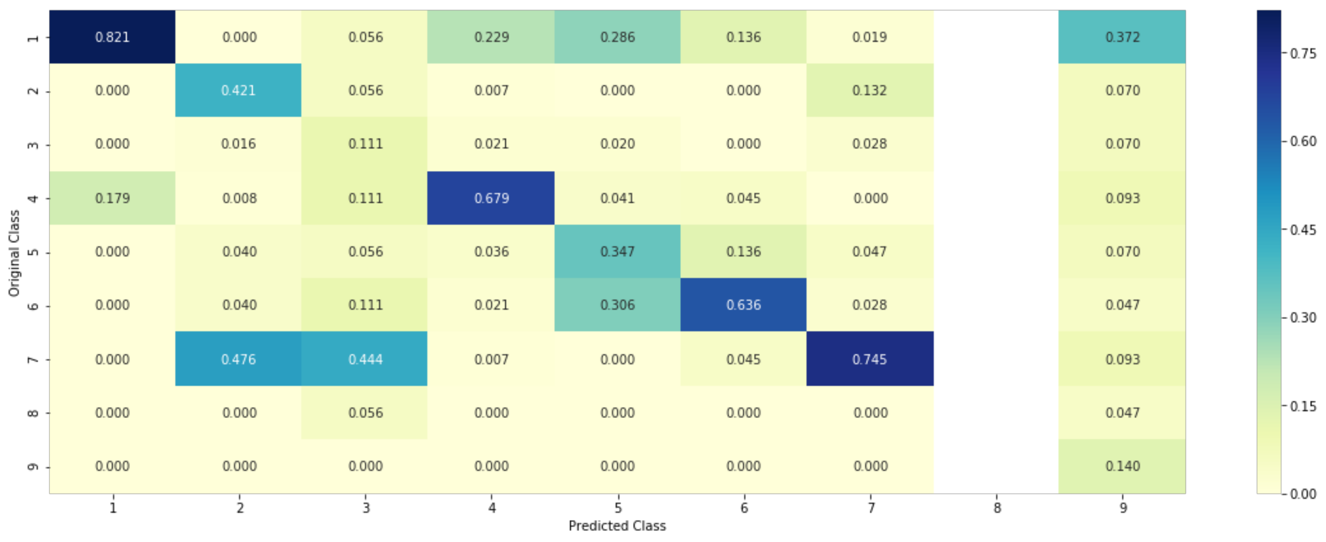
In [86]:

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

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

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

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

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

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

# Variables that will be used in the end to make comparison table of models
rf_response_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y))/cv_y.s
hape[0])*100
```

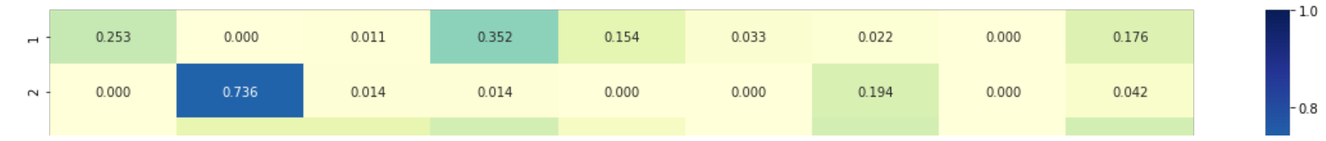
```
Log loss : 1.3647180507449557
Number of mis-classified points : 0.4567669172932331
-------------------- Confusion matrix --------------------
```

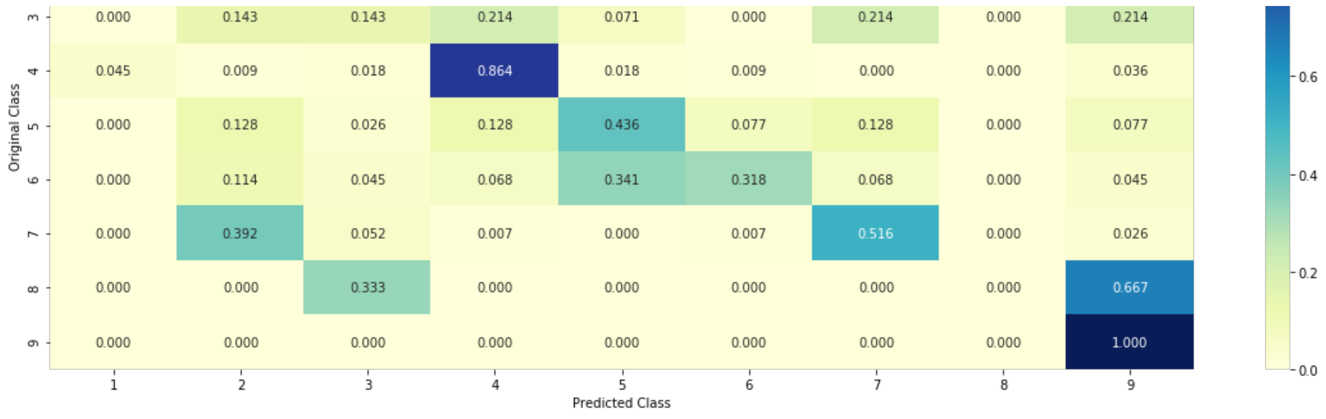


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [87]:

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


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

```
Predicted Class : 4
Predicted Class Probabilities: [[1.680e-02 4.700e-03 4.410e-02 7.772e-01 1.970e-02 1.560e-02 7.000
e-04
  2.110e-02 1.001e-01]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
```

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

4.5.5.2. Incorrectly Classified point

In [88]:

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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0264 0.08   0.2991 0.2734 0.0664 0.056  0.0614 0.0442 0.0931]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [89]:

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

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

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


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

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


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

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

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

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


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', 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.predict_proba(cv_x_onehot
Coding))))
```

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

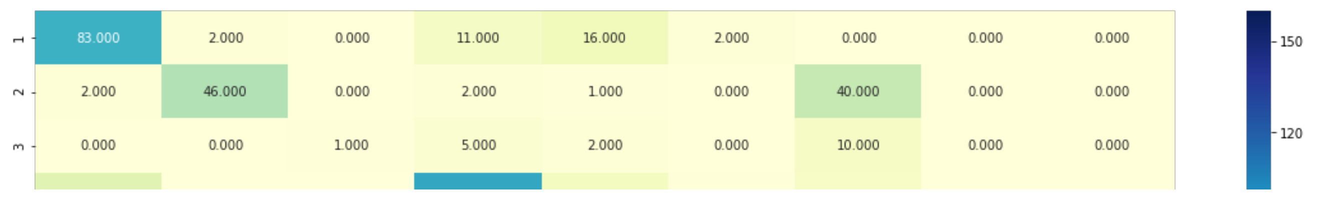
```
Logistic Regression :  Log Loss: 0.96
Support vector machines : Log Loss: 1.66
Naive Bayes : Log Loss: 1.18
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.465
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.060
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.205
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.543
```

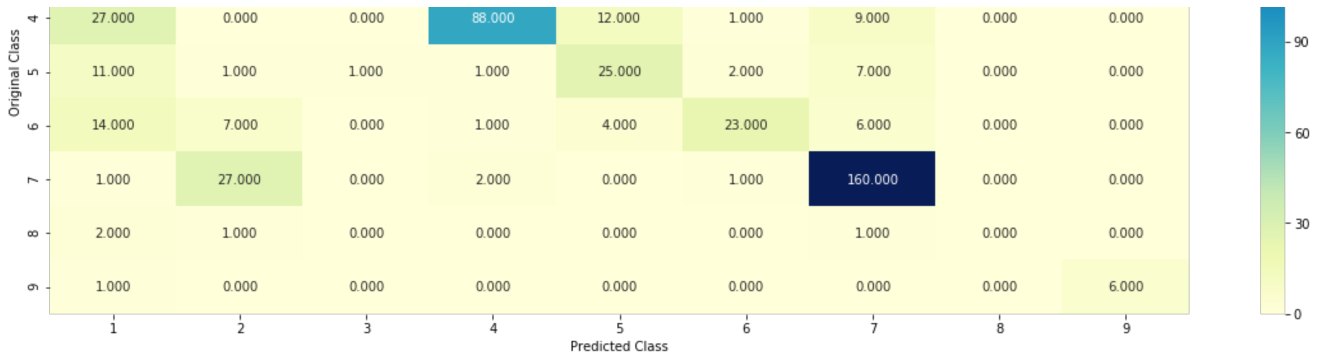
4.7.2 testing the model with the best hyper parameters

In [90]:

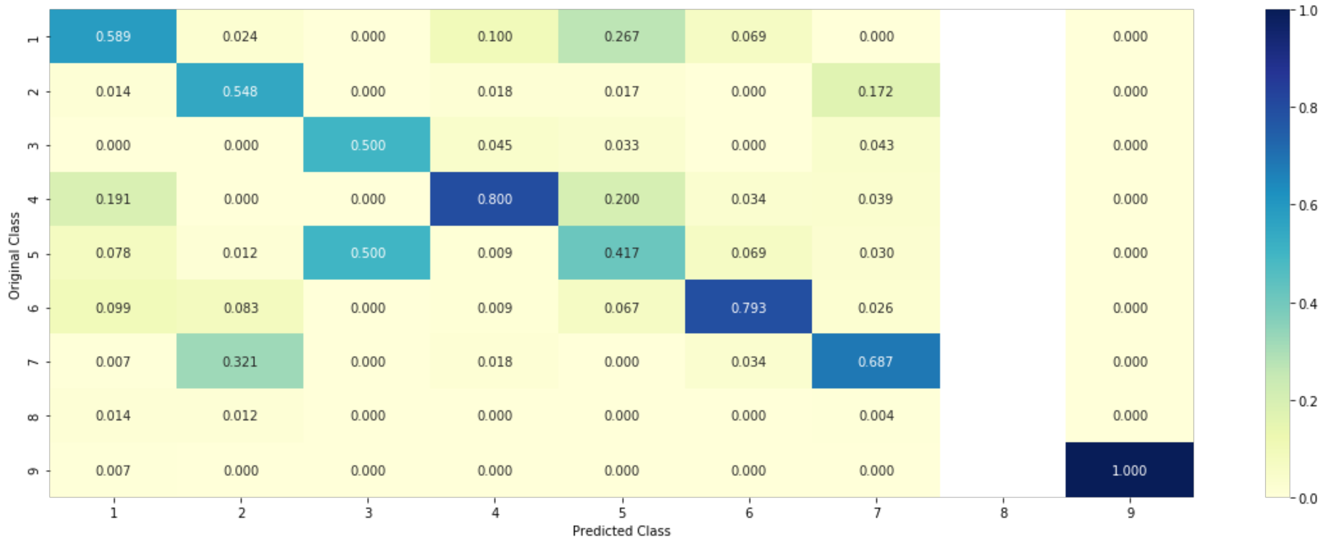
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error1 = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error1)

log_error2 = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error2)

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

# Variables that will be used in the end to make comparison table of all models
stack_train = log_error
stack_cv = log_error1
stack_test = log_error2
stack_misclassified = (np.count_nonzero((sclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0
])*100
```

```
Log loss (train) on the stacking classifier : 0.6184358774240548
Log loss (CV) on the stacking classifier : 1.0602518435042068
Log loss (test) on the stacking classifier : 1.0804392030380716
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```

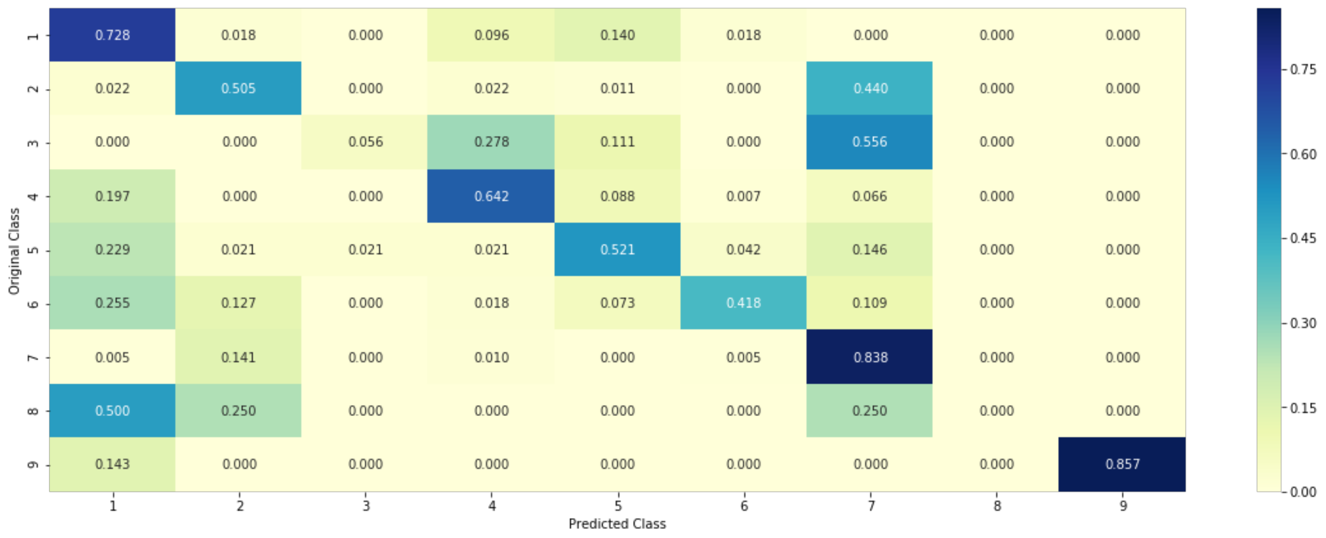




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

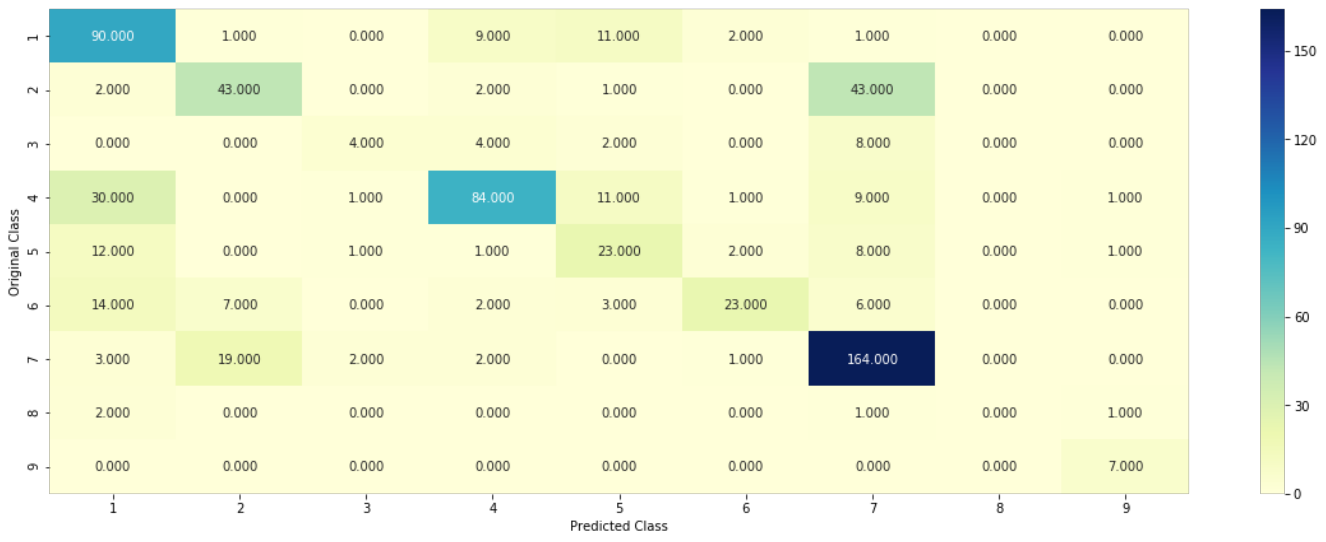


4.7.3 Maximum Voting classifier

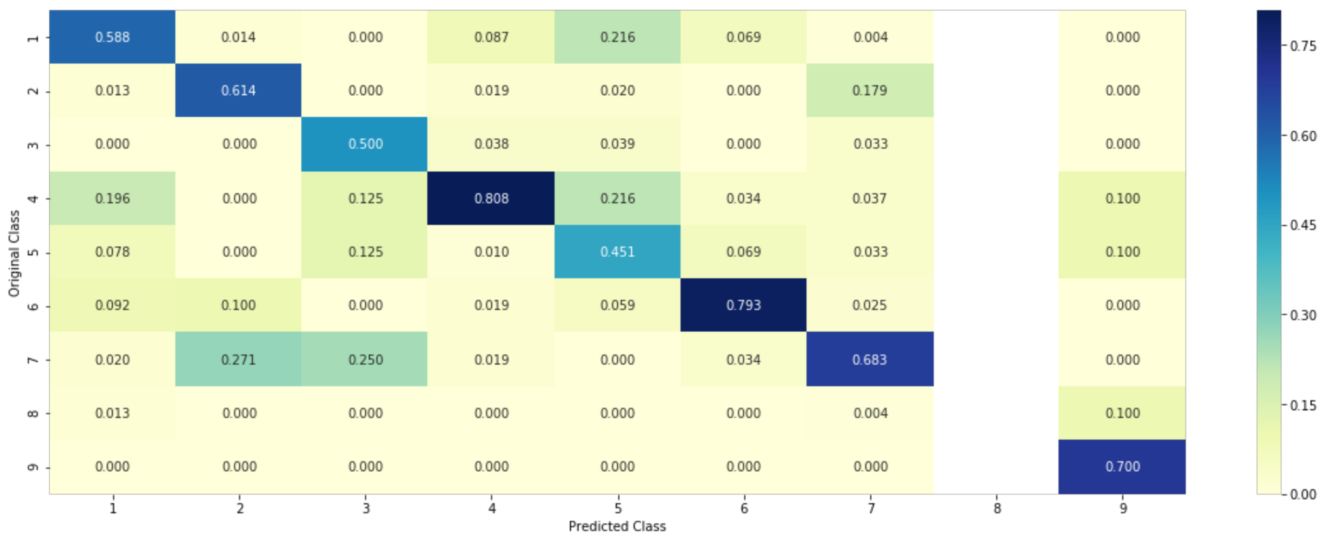
In [91]:

```
#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))

# Variables that will be used in the end to make comparison table of all models
mvc_train = log_loss(train_y, vclf.predict_proba(train_x_onehotCoding))
mvc_cv = log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
mvc_test = log_loss(test_y, vclf.predict_proba(test_x_onehotCoding))
mvc_misclassified = (np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
*100
```

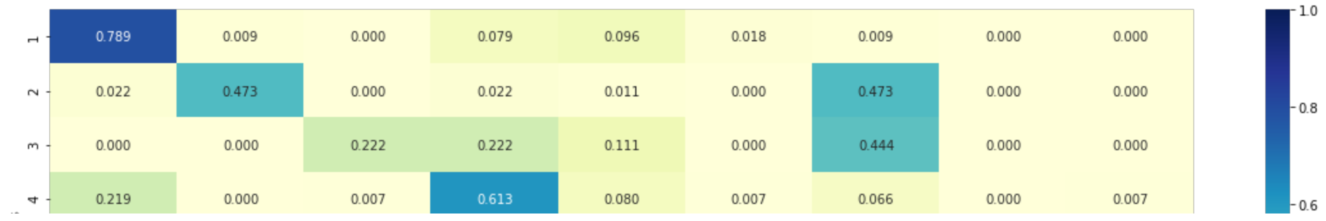
```
Log loss (train) on the VotingClassifier : 0.8485518666906269
Log loss (CV) on the VotingClassifier : 1.1102919878020685
Log loss (test) on the VotingClassifier : 1.110892812805966
Number of missclassified point : 0.34135338345864663
-------------------- Confusion matrix --------------------
```

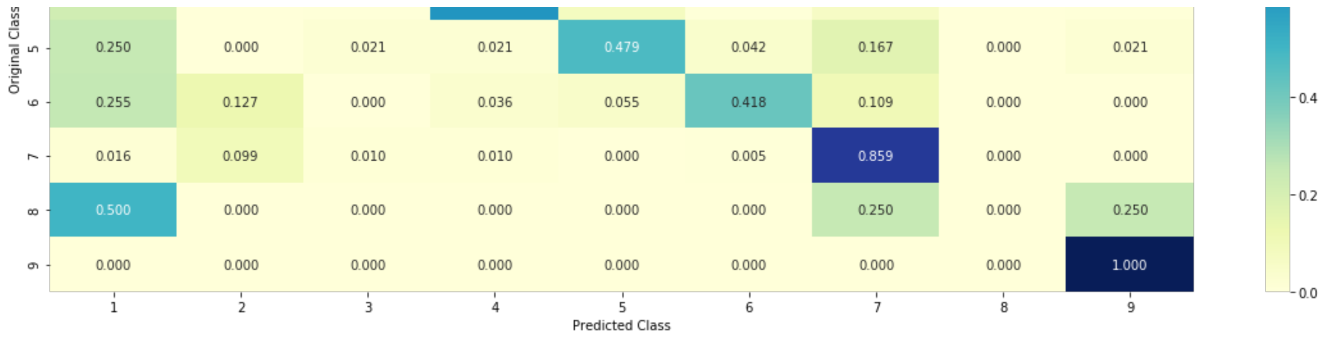


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





CONCLUSION

(a). Procedure Followed :-

STEP 1:- Collect all Genes and variations in a single list and train TfidfVectorizer on this list .

STEP 2:- Then apply same vectorizer on train_df['TEXT'] , test_df['TEXT'] and cv_df['TEXT'] in order to transforming them into vectors which have some more information and relation

STEp 3:- Normalize these vectors and then stack them with all onehotCoded features

As a part of feature engineering I have added these vectors which little more information about genes , variations and some type of connection between gene , variation and the literarture . By doing this I tried to extract some more information from literarture about genes and variations . It really improves the result by lowering 'Test Loss' and 'CV-Loss' below 1 which was the objective of this assignment .

NOTE :- I have performed all the above mentioned steps under the 'FEATURE ENGINEERING' section .

(b).Table (Model Performances):-

In [92]:

```
# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names =['Naive Bayes','K-Nearest Neighbour','LR With Class Balancing',\
        'LR Without Class Balancing','Linear SVM',\
        'RF With One hot Encoding','RF With Response Coding',\
        'Stacking Classifier','Maximum Voting Classifier']

# Training loss
train_loss =
[nb_train,knn_train,lr_balance_train,lr_train,svm_train,rf_train,rf_response_train,stack_train,mvc_
train]

# Cross Validation loss
cv_loss = [nb_cv,knn_cv,lr_balance_cv,lr_cv,svm_cv,rf_cv,rf_response_cv,stack_cv,mvc_cv]

# Test loss
test_loss = [nb_test,knn_test,lr_balance_test,lr_test,svm_test,rf_test,rf_response_test,stack_test
,mvc_test]

# Percentage Misclassified points
misclassified =
[nb_misclassified,knn_misclassified,lr_balance_misclassified,lr_misclassified,svm_misclassified,\
                 rf_misclassified,rf_response_misclassified,stack_misclassified,mvc_misclassified]

numbering = [1,2,3,4,5,6,7,8,9]

# Initializing prettytable
ptable = PrettyTable()
```

```
# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Train_loss",train_loss)
ptable.add_column("CV_loss",cv_loss)
ptable.add_column("Test_loss",test_loss)
ptable.add_column("Misclassified(%)",misclassified)

# Printing the Table
print(ptable)
```

```
+--------+----------------------------+---------------------+---------------------+------------------
---------------------+
| S.NO. |            MODEL            |      Train_loss     |       CV_loss       |      Test_loss
|  Misclassified(%)   |
+--------+----------------------------+---------------------+---------------------+------------------
---------------------+
|   1    |         Naive Bayes         |  0.7553171685507521 | 1.1557199838533636 | 1.211123107331458
| 34.21052631578947   |
|   2    |     K-Nearest Neighbour     |  0.6390189098519682 | 0.9848064716006467 |
1.0724042336894166 | 33.83458646616541   |
|   3    |  LR With Class Balancing    |  0.6148709086017381 | 0.9589334993636797 |
0.9221464647747004 | 31.76691729323308   |
|   4    | LR Without Class Balancing |  0.6137349678129014 | 0.9977573229948815 | 0.956344342031991
9 | 31.015037593984964 |
|   5    |          Linear SVM         |  0.5406525457948057 | 1.0356022297353413 |
0.9781375569874082 | 33.45864661654135   |
|   6    |  RF With One hot Encoding  |  0.9033388433412035 | 1.1522167376194856 |
1.1976185026027777 | 38.15789473684211   |
|   7    |  RF With Response Coding   | 0.13992511623893156 | 1.3647180507449557 |
1.5168480576687147 | 45.67669172932331   |
|   8    |     Stacking Classifier     |  0.6184358774240548 | 1.0602518435042068 |
1.0804392030380716 | 35.037593984962406 |
|   9    | Maximum Voting Classifier  |  0.8485518666906269 | 1.1102919878020685 | 1.110892812805966
| 34.13533834586466   |
+--------+----------------------------+---------------------+---------------------+------------------
---------------------+
```