Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25 (https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25)
2. https://www.youtube.com/watch?v=UwbuW7oK8rk (https://www.youtube.com/watch?v=UwbuW7oK8rk)
3. https://www.youtube.com/watch?v=qxXRKVompI8 (https://www.youtube.com/watch?v=qxXRKVompI8)

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data (https://www.kaggle.com/c/msk-redefining-cancer-treatment/data)
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

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

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

Out[3]:

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

3.1.3. Preprocessing of text

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


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

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

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

In [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "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 : 315.89399914893323 seconds
```

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

Out[6]:

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

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

Out[7]:

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

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

In [9]:
```
result[result['ID']==1277]
```

Out[9]:

	ID	Gene	Variation	Class	TEXT
1277	1277	ARID5B	Truncating Mutations	1	ARID5B Truncating Mutations

3.1.4. Test, Train and Cross Validation Split

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

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

# split the data into test and train by maintaining same distribution of output var
aible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_tru
e, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribu
tion 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_trai
n, test_size=0.2)
```

In [11]:
```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variation'].va
lues;cv_variation=cv_df['Variation'].values
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv_df['
Gene'].values
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv_df['
TEXT'].values

from sklearn.feature_extraction.text import CountVectorizer
encode=CountVectorizer()
train_variation=encode.fit_transform(train_variation);test_variation=encode.transfo
rm(test_variation);cv_variation=encode.transform(cv_variation)
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);c
v_gene=encode.transform(cv_gene)
#train_text=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);
cv_gene=encode.transform(cv_gene)

print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_text);c
v_gene=encode.transform(cv_text)
print(train_text.shape)
```

```
(2124, 237)
  (0, 87)       1
(2124, 1956)
  (0, 1342)     1
(2124, 126615)
```

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
```

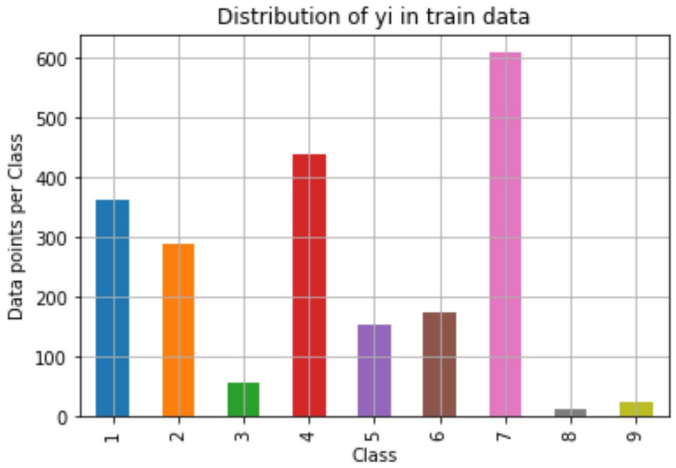
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [13]:
```
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])
```

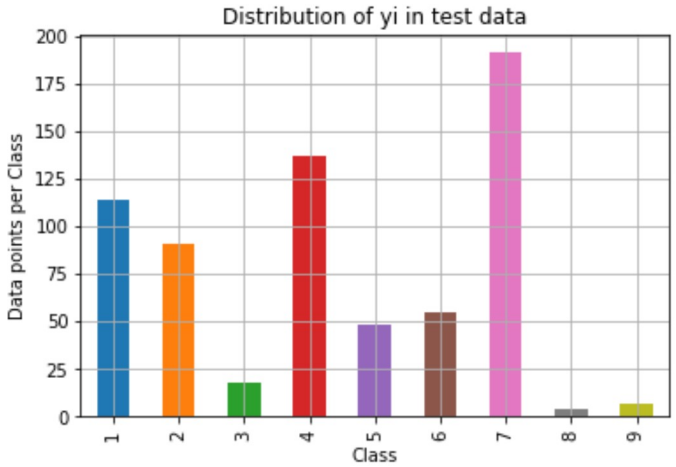
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

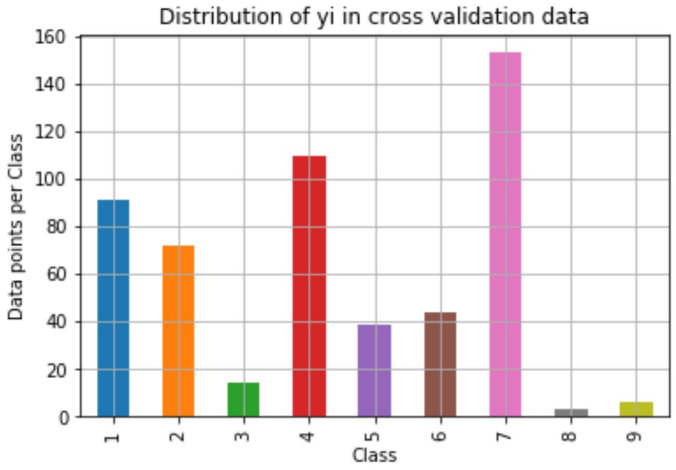
In [14]:
```
# it returns a dict, keys as class labels and values as the number of data points i
n that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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.h
tml
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
er
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.value
s[i], '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3),
'%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.h
tml
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
er
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.round((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.h
tml
# -(train_class_distribution.values): the minus sign will give us in decreasing ord
er
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 %)
```

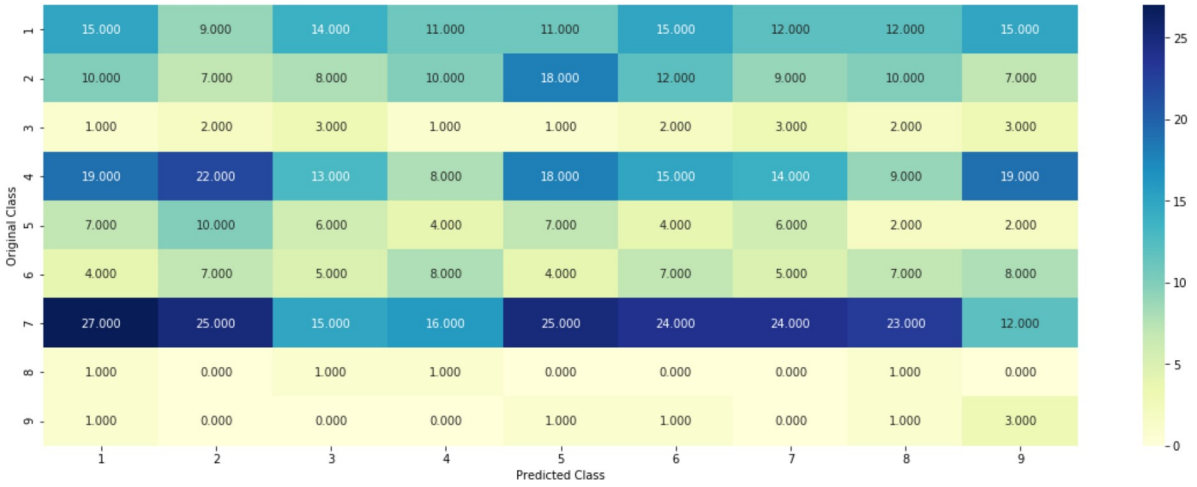
3.2 Prediction using a 'Random' Model

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

In [15]:
```
# 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 pr
edicted class j

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

    # 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 ro
w
    # 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, ytickl
abels=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, ytickl
abels=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, ytickl
abels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

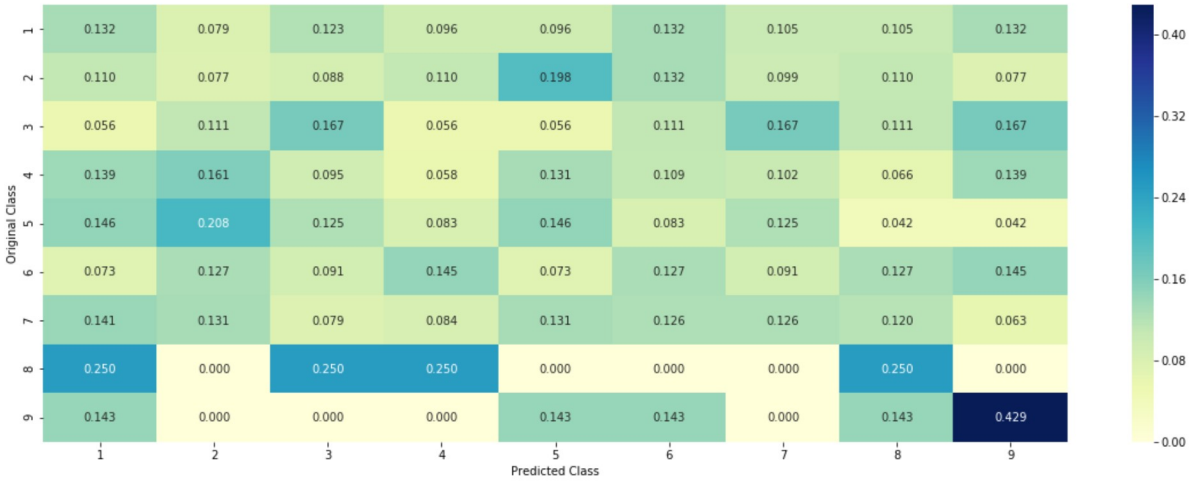
In [16]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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_predi
cted_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 2.4960806201036077
Log loss on Test Data using Random Model 2.4950554750964873
-------------------- 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 / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation o
f 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 i
n whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pe
rticular 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
```

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?

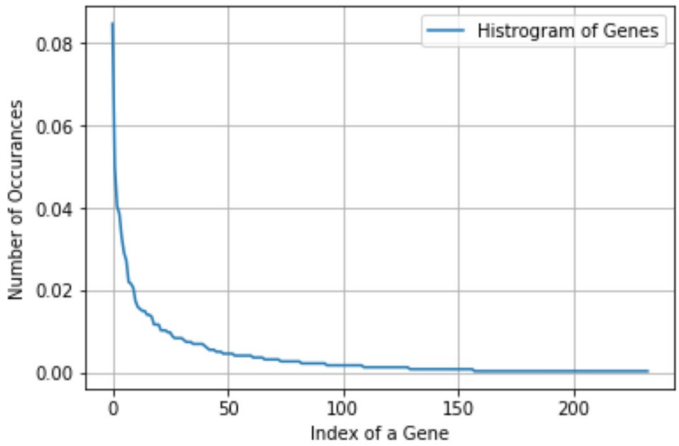
In [17]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 237
BRCA1    174
TP53      99
EGFR      93
PTEN      83
BRCA2     74
KIT       62
BRAF      61
ERBB2     50
ALK       43
FGFR2     39
Name: Gene, dtype: int64
```

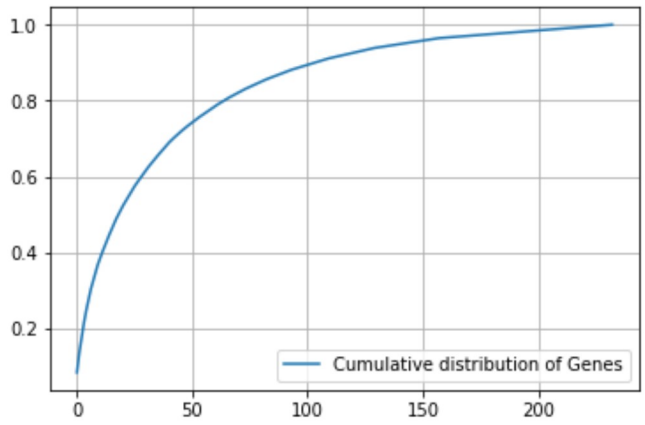
In [18]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in th
e train data, and they are distibuted as follows",)
```

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

In [20]:
```
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 [21]:
```
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 [22]:
```
#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 [23]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding
method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [24]:
```
# 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 [25]:
```
train_df['Gene'].head(1000)
```

Out[25]:
```
450        TP53
2167       PTEN
764       ERBB2
1322       MLH1
1475      FGFR2
2696       BRAF
13          CBL
2707       BRAF
1638      RRAS2
2350       JAK2
941      PDGFRB
1233       PIM1
2046       SOS1
901      PDGFRA
2159       PTEN
2924     NFE2L2
1149        MET
584       SMAD4
2772       BRAF
2319       JAK2
1704       PMS2
642      CDKN2A
1295       HRAS
821       ERCC2
2441      BRCA1
749       ERBB2
1463      FGFR2
2805      BRCA2
1415      FGFR3
403        TP53
          ...
1663       FLT3
3153       KRAS
3264        RET
1836    PPP2R1A
880      PDGFRA
438        TP53
3058        KIT
2525      BRCA1
2392     PTPN11
786       ERBB4
1521        ALK
2584      BRCA1
2240       PTEN
2874      BRCA2
2284        CIC
1635     MAP2K4
2713       BRAF
2975        KIT
818       ERCC2
1681       FLT3
3081     NOTCH1
770       ERBB2
1334       MLH1
2965        KIT
1066      EWSR1
3179       AKT3
1629     MAP2K4
678      CDKN2A
2962        KIT
956       RBM10
Name: Gene, Length: 1000, dtype: object
```

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

Out[26]:
```
['abl1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
```

In [27]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

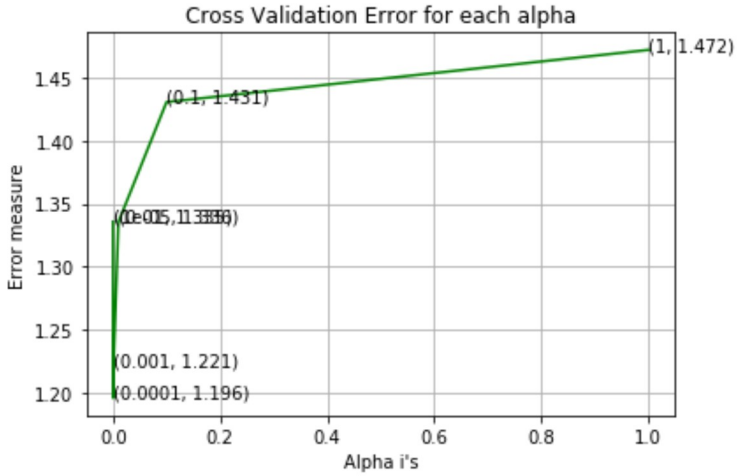
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding meth
od. The shape of gene feature: (2124, 232)
```

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

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

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generat
ed/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_interc
ept=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 Gr
adient 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.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",lo
g_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 l
oss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.3358618375245341
For values of alpha =  0.0001 The log loss is: 1.1958823718375513
For values of alpha =  0.001 The log loss is: 1.221187064501795
For values of alpha =  0.01 The log loss is: 1.3354305350512976
For values of alpha =  0.1 The log loss is: 1.4308001572017828
For values of alpha =  1 The log loss is: 1.472177271738837
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0451881551912146
For values of best alpha =  0.0001 The cross validation log loss is: 1.195882371
8375513
For values of best alpha =  0.0001 The test log loss is: 1.2226660776343181
```

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

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

In [29]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", uniqu
e_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_co
verage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_
coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  233  genes
in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 514 out of  532 : 96.61654135338345
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

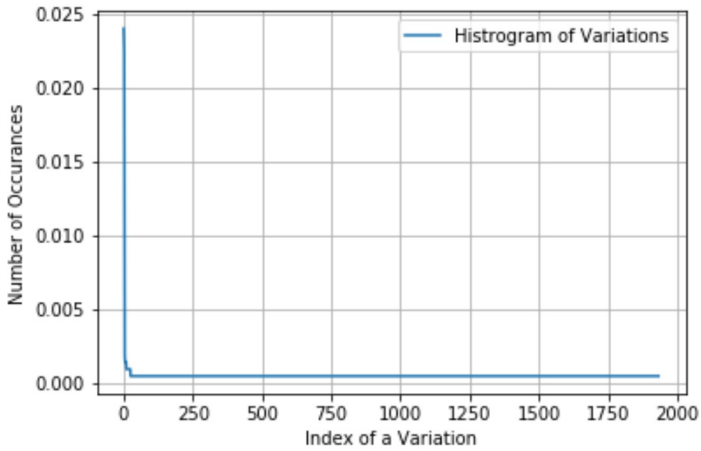
In [30]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1933
Truncating_Mutations    51
Amplification           49
Deletion                48
Fusions                 19
Overexpression           4
T58I                     3
Q61H                     3
E17K                     3
Q61L                     3
G12V                     3
Name: Variation, dtype: int64
```

In [31]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variat
ions in the train data, and they are distibuted as follows",)
```

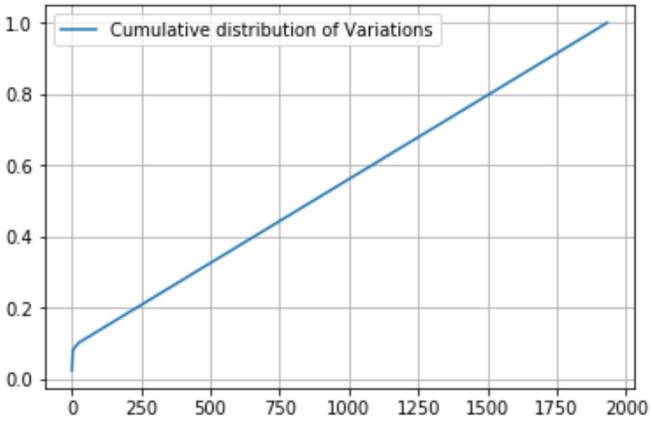
```
Ans: There are 1933 different categories of variations in the train data, and th
ey are distibuted as follows
```

In [32]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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

```
[0.0240113  0.04708098 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 [34]:
```
# 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", c
v_df))
```

In [35]:
```
print("train_variation_feature_responseCoding is a converted feature using the resp
onse coding method. The shape of Variation feature:", train_variation_feature_respo
nseCoding.shape)
```

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

In [36]:
```
# 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['Varia
tion'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation
'])
```

In [37]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-ho
t encoding method. The shape of Variation feature:", train_variation_feature_onehot
Coding.shape)
```

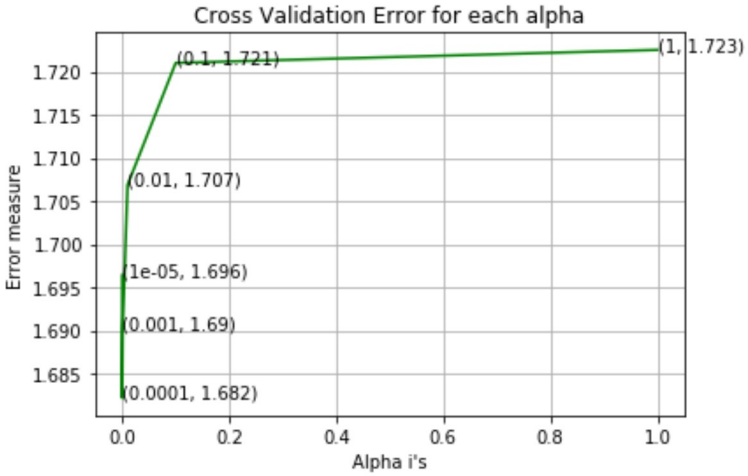
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot en
coding method. The shape of Variation feature: (2124, 1965)
```

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

Let's build a model just like the earlier!

In [38]:
```
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_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:
#-------------------------------


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[:,1000], 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.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss 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: 1.6964036859819638
For values of alpha =  0.0001 The log loss is: 1.6821815639982485
For values of alpha =  0.001 The log loss is: 1.6901667102216893
For values of alpha =  0.01 The log loss is: 1.7068565861251481
For values of alpha =  0.1 The log loss is: 1.721004847441335
For values of alpha =  1 The log loss is: 1.7225142779747606
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7544372376792005
For values of best alpha =  0.0001 The cross validation log loss is: 1.682181563
9982485
For values of best alpha =  0.0001 The test log loss is: 1.705529949256027
```

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 [39]:
```
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_co
verage/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  1933  genes in test and cross va
lidation data sets?
Ans
1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 61 out of  532 : 11.466165413533833
```

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 [19]:
```
# 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 [20]:
```
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.ge
t(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 [42]:
```
# building a TfidfVectorizer with all the words that occured minimum 3 times in tra
in data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [43]:
```
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 build 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 [44]:
```
#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 [45]:
```
# 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_fe
ature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_
responseCoding.sum(axis=1)).T
```

In [46]:
```
# 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)
```

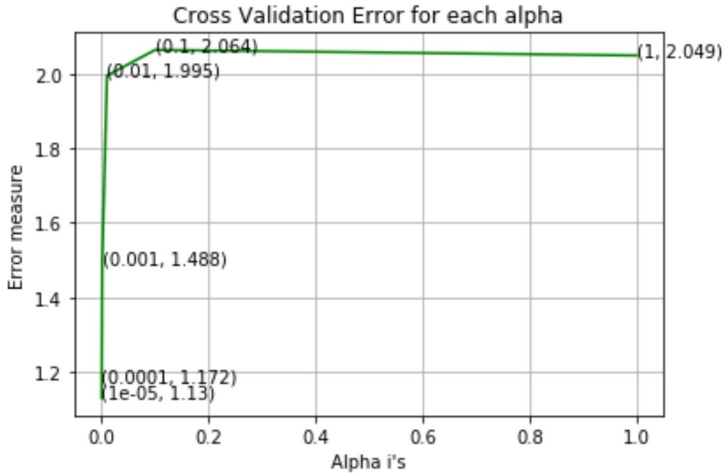
In [47]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reve
rse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [48]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({248.23116335293767: 1, 177.48929467809373: 1, 138.57409682420501: 1, 13
0.13525546312957: 1, 127.86119957029003: 1, 116.64610375412225: 1, 116.185386851
6794: 1, 114.09409293828331: 1, 110.72538320754: 1, 108.11307867153707: 1, 106.0
0690610095037: 1, 88.48748557017836: 1, 88.24659881080846: 1, 87.39282708460624:
1, 82.6805481325271: 1, 78.60774133538901: 1, 78.311307803483: 1, 78.06340732683
785: 1, 77.35224049187141: 1, 76.73276540867985: 1, 73.71223410407015: 1, 72.237
46735012793: 1, 69.76971700680544: 1, 68.13322258624484: 1, 66.6043400989091: 1,
65.96079007495331: 1, 65.36195766515372: 1, 63.64551162345823: 1, 63.63804732869
9244: 1, 62.985543902344226: 1, 62.88852581721374: 1, 61.980275449402114: 1, 60.
29069802470132: 1, 58.93868595345348: 1, 58.607923057113254: 1, 56.1984983536858
14: 1, 55.67566484418234: 1, 54.301261461866304: 1, 52.325778176578744: 1, 51.43
412848069889: 1, 50.10162985181774: 1, 50.02867938944281: 1, 49.7054323134103061
1, 49.49615804680878: 1, 48.5290386481262: 1, 46.62603861380613: 1, 45.857034106
71138: 1, 45.02436987005878: 1, 44.57969742965053: 1, 44.296405332038006: 1, 43.
787049945331155: 1, 43.444321069579495: 1, 43.30804814150154: 1, 42.824098755965
4: 1, 42.6297831762025: 1, 42.37785737787571: 1, 42.215358949251495: 1, 42.00840
100692535: 1, 41.634283063076175: 1, 41.52789817253825: 1, 41.37910521969451: 1,
41.10054795352338: 1, 40.977576015446196: 1, 40.56907826601832: 1, 40.3388489429
0341: 1, 40.26915064219218: 1, 40.037579929937756: 1, 39.517576312002795: 1, 38.
662414361996646: 1, 38.64395681580293: 1, 38.109542956900164: 1, 37.827919129464
05: 1, 36.98909781334603: 1, 36.752531564348395: 1, 36.42354363757844: 1, 36.421
98193481807: 1, 35.90434885409839: 1, 35.669506915849475: 1, 35.55434378498556:
1, 35.433261577868564: 1, 35.32944330419822: 1, 35.22692980850258: 1, 35.1332701
97920304: 1, 34.45043482450015: 1, 34.432521128188924: 1, 34.17339716004611: 1,
33.88334246587092: 1, 33.762098236442824: 1, 33.263099824920516: 1, 32.872244190
46199: 1, 32.810249203099325: 1, 32.56995313915069: 1, 32.43697089872527: 1, 32.
39858406139637: 1, 32.38188686346656: 1, 32.37322735660578: 1, 32.220002851362594: 1, 31.965126330440555: 1, 31.959834406518542: 1, 31.901358314431523: 1, 31.52
9076001278224: 1, 31.38700388409487: 1, 31.335191644750726: 1, 31.16873613714657
4: 1, 31.122037231033477: 1, 31.11852864769814: 1, 30.99400697790504: 1, 30.9839
2416310038: 1, 30.902048230841057: 1, 30.8138791057411: 1, 30.75194071783279: 1,
30.716524745488023: 1, 30.331247840829842: 1, 30.21453374403499: 1, 30.111808137
765063: 1, 30.034186174725324: 1, 29.79303528973813: 1, 29.593590718200414: 1, 29.543002569172124: 1, 29.437380331846768: 1, 29.246573968483304: 1, 29.245647442
39168: 1, 29.040407724655136: 1, 28.932851419405473: 1, 28.796502033594095: 1, 28.69917832787265: 1, 28.073399992705724: 1, 27.76468784302131: 1, 27.59383252297
0075: 1, 27.484550844647064: 1, 27.432849605036292: 1, 27.416048023018554: 1, 27
.209285248603454: 1, 27.20082043813691: 1, 27.11793239631542: 1, 27.006896189658
153: 1, 26.97219840624218: 1, 26.863104174157986: 1, 26.57209926389624: 1, 26.56
7242726426336: 1, 26.545806988308158: 1, 26.345468891999253: 1, 26.3161731214711
24: 1, 26.182346401711854: 1, 26.159411468257296: 1, 26.08594843375746: 1, 25.88
3196436408213: 1, 25.867989332194263: 1, 25.804673020447474: 1, 25.7577750366694
6: 1, 25.59653049294496: 1, 25.529318342108287: 1, 25.405021171506977: 1, 25.256
051253470517: 1, 25.221667384616257: 1, 25.18384400656811: 1, 24.937820467535452
: 1, 24.54700472471929: 1, 24.4011380179294: 1, 24.397721812018602: 1, 24.397307
802632152: 1, 24.285417029242268: 1, 24.27641986113062: 1, 24.21014199609928: 1,
24.08678392437536: 1, 24.020504054471694: 1, 24.014141159384497: 1, 23.939946052
087613: 1, 23.904217139099053: 1, 23.839389596585146: 1, 23.754060125992655: 1,
23.73156827288999: 1, 23.617835952520725: 1, 23.551777655810465: 1, 23.501760234
722784: 1, 23.490039971850333: 1, 23.425781396893697: 1, 23.364400463413784: 1,
23.276428718352214: 1, 23.251351871316427: 1, 23.201637496783423: 1, 23.12566574
5211418: 1, 23.080493688641607: 1, 23.02497573831626: 1, 22.76827281988532: 1, 22.70619114507234: 1, 22.683225048278516: 1, 22.623299768435473: 1, 22.5963797418
48843: 1, 22.56755890695917: 1, 22.438895813358428: 1, 22.369064755952603: 1, 22
.236559420751675: 1, 22.192015977458194: 1, 22.176412375411882: 1, 22.1304052135
57125: 1, 22.035421620767938: 1, 21.96454227025943: 1, 21.963760031459785: 1, 21
.8581901408325: 1, 21.79747973466666: 1, 21.783459860429225: 1, 21.7695196968099
3: 1, 21.70656386098854: 1, 21.704162165708315: 1, 21.687955715004613: 1, 21.674
24227836494: 1, 21.497382491464464: 1, 21.461625680418: 1, 21.37641043643233: 1,
21.34132685803749: 1, 21.34034604176173: 1, 21.328649861289943: 1, 21.3249541070
01517: 1, 21.27007715417425: 1, 21.258728623223956: 1, 21.161757685481124: 1, 21
.151686313914745: 1, 21.13349134027349: 1, 21.130897486384857: 1, 21.11793390565
53: 1, 21.114704373583823: 1, 21.074326572711882: 1, 21.033103694986465: 1, 21.0
27957093735914: 1, 21.00950125214147: 1, 20.926434986625825: 1, 20.8495943185971
```

In [49]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-ho
t encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generat
ed/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_interc
ept=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 Gr
adient 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.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",lo
g_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 l
oss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log
_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.130068459566638
For values of alpha =  0.0001 The log loss is: 1.1719708101104855
For values of alpha =  0.001 The log loss is: 1.4883040060010806
For values of alpha =  0.01 The log loss is: 1.9950436934053521
For values of alpha =  0.1 The log loss is: 2.0640647608446763
For values of alpha =  1 The log loss is: 2.0491287500694195
```




```
For values of best alpha =  1e-05 The train log loss is: 0.77364299510333
For values of best alpha =  1e-05 The cross validation log loss is: 1.1300684595
66638
For values of best alpha =  1e-05 The test log loss is: 1.1203905604898086
```

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

Ans. Yes, it seems like!

In [21]:
```
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 [51]:
```
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 trai
n data")
```

```
95.1 % of word of test data appeared in train data
93.7 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [52]:
```
model_scores_table={'model_name':[],'train_score':[],'cv_score':[],'test_score':[],
'misclassified_points':[]}
```

In [53]:
```
#model_scores_table['model_name'].append('llr')
model_scores_table
```

Out[53]:
```
{'model_name': [],
 'train_score': [],
 'cv_score': [],
 'test_score': [],
 'misclassified_points': []}
```

In [22]:
```
#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))/t
est_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [23]:
```
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 [24]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

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

Stacking the three types of features

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

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

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

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

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

cv_x_onehotCoding = hstack((cv_gene_var_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 [62]:
```
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, 3197)
(number of data points * number of features) in test data =  (665, 3197)
(number of data points * number of features) in cross validation data = (532, 3197)
```

In [63]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respo
nseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_respons
eCoding.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
)
```

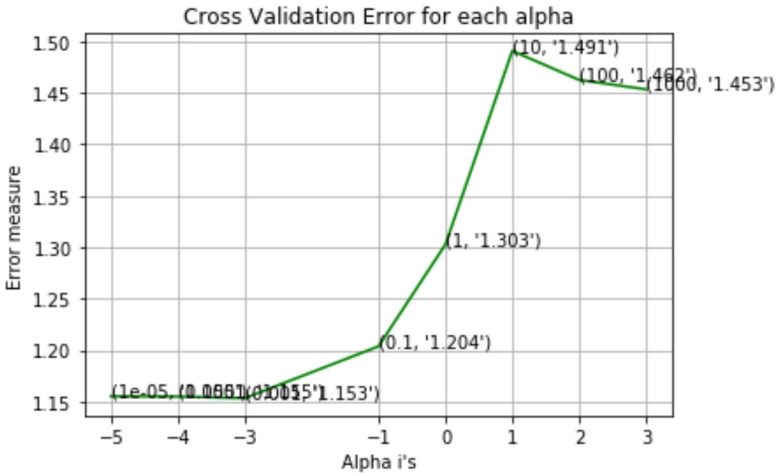
4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [73]:
```
# 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)
```

```
for alpha = 1e-05
Log Loss : 1.1550975987095586
for alpha = 0.0001
Log Loss : 1.1548826318199517
for alpha = 0.001
Log Loss : 1.153235894590452
for alpha = 0.1
Log Loss : 1.203597050187973
for alpha = 1
Log Loss : 1.302616224502697
for alpha = 10
Log Loss : 1.4907191908809783
for alpha = 100
Log Loss : 1.4624143322460792
for alpha = 1000
Log Loss : 1.4533191261472318
```

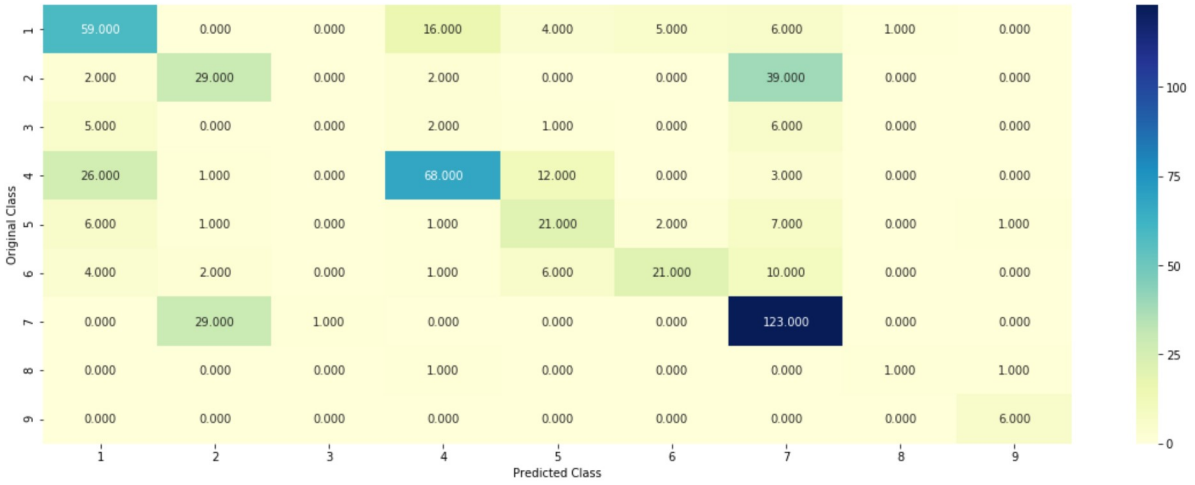



```
For values of best alpha =  0.001 The train log loss is: 0.5235053477571179
For values of best alpha =  0.001 The cross validation log loss is: 1.1532358945
90452
For values of best alpha =  0.001 The test log loss is: 1.2056351149807534
```

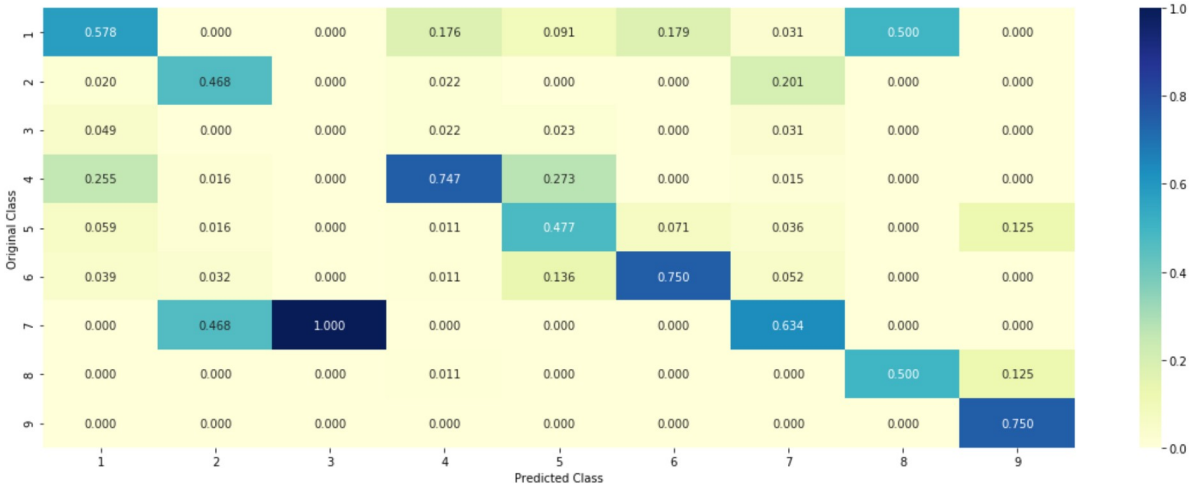
4.1.1.2. Testing the model with best hyper paramters

In [74]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

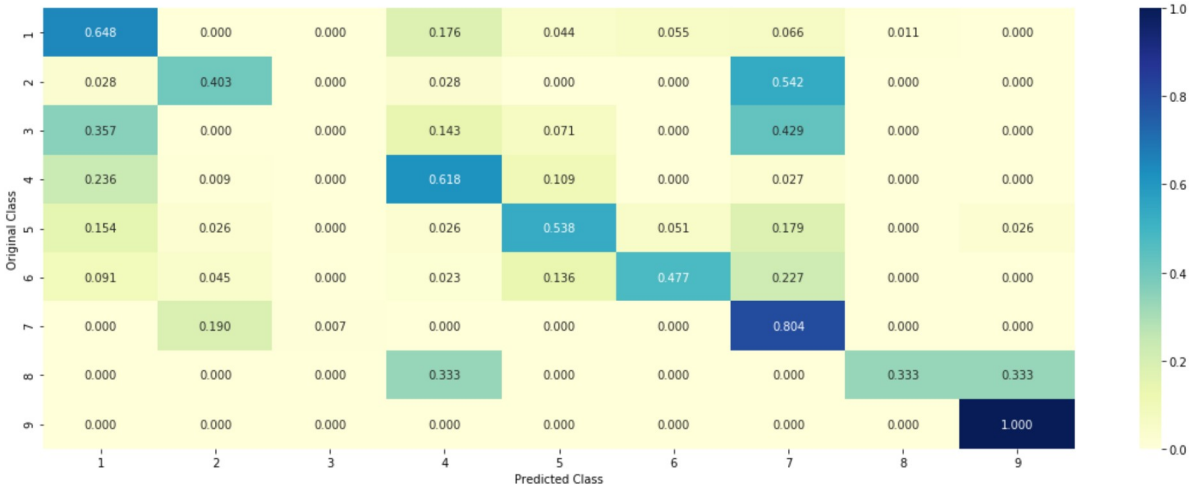
```
Log Loss : 1.153235894590452
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [75]:
```
test_point_index = 1
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0681 0.038  0.0108 0.0688 0.0343 0.6939 0.080
4 0.0029 0.0026]]
Actual Class : 6
--------------------------------------------------
6 Text feature [deleterious] present in test data point [True]
8 Text feature [classified] present in test data point [True]
10 Text feature [history] present in test data point [True]
11 Text feature [combined] present in test data point [True]
12 Text feature [brca2] present in test data point [True]
14 Text feature [family] present in test data point [True]
16 Text feature [brca1] present in test data point [True]
17 Text feature [000] present in test data point [True]
20 Text feature [model] present in test data point [True]
21 Text feature [sequence] present in test data point [True]
22 Text feature [evidence] present in test data point [True]
24 Text feature [predicted] present in test data point [True]
27 Text feature [expected] present in test data point [True]
30 Text feature [known] present in test data point [True]
31 Text feature [23] present in test data point [True]
32 Text feature [variant] present in test data point [True]
33 Text feature [neutral] present in test data point [True]
35 Text feature [56] present in test data point [True]
38 Text feature [use] present in test data point [True]
39 Text feature [substitutions] present in test data point [True]
40 Text feature [used] present in test data point [True]
41 Text feature [variants] present in test data point [True]
43 Text feature [likely] present in test data point [True]
44 Text feature [ovarian] present in test data point [True]
45 Text feature [data] present in test data point [True]
47 Text feature [tables] present in test data point [True]
48 Text feature [studied] present in test data point [True]
52 Text feature [50] present in test data point [True]
53 Text feature [would] present in test data point [True]
54 Text feature [individuals] present in test data point [True]
56 Text feature [significance] present in test data point [True]
57 Text feature [25] present in test data point [True]
59 Text feature [36] present in test data point [True]
61 Text feature [60] present in test data point [True]
62 Text feature [analysis] present in test data point [True]
64 Text feature [approach] present in test data point [True]
65 Text feature [although] present in test data point [True]
66 Text feature [significant] present in test data point [True]
67 Text feature [16] present in test data point [True]
68 Text feature [site] present in test data point [True]
69 Text feature [ring] present in test data point [True]
70 Text feature [majority] present in test data point [True]
71 Text feature [characteristics] present in test data point [True]
72 Text feature [substitution] present in test data point [True]
75 Text feature [values] present in test data point [True]
76 Text feature [available] present in test data point [True]
77 Text feature [database] present in test data point [True]
78 Text feature [methods] present in test data point [True]
79 Text feature [overall] present in test data point [True]
81 Text feature [conserved] present in test data point [True]
83 Text feature [interaction] present in test data point [True]
85 Text feature [24] present in test data point [True]
86 Text feature [missense] present in test data point [True]
87 Text feature [72] present in test data point [True]
89 Text feature [information] present in test data point [True]
91 Text feature [proportion] present in test data point [True]
93 Text feature [genetic] present in test data point [True]
94 Text feature [least] present in test data point [True]
95 Text feature [number] present in test data point [True]
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [76]:
```
test_point_index = 5
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

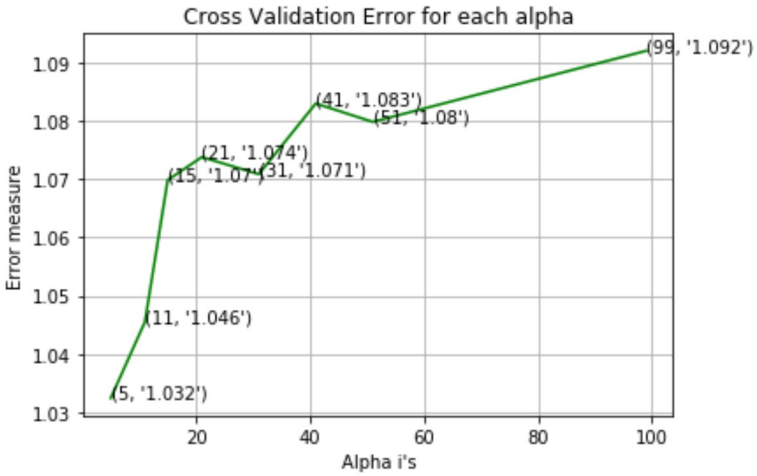
```
Predicted Class : 9
Predicted Class Probabilities: [[0.0846 0.0489 0.0137 0.2294 0.0412 0.0416 0.110
3 0.0036 0.4267]]
Actual Class : 8
--------------------------------------------------
4 Text feature [aberrant] present in test data point [True]
5 Text feature [rna] present in test data point [True]
7 Text feature [idh1] present in test data point [True]
9 Text feature [methylation] present in test data point [True]
10 Text feature [mutant] present in test data point [True]
11 Text feature [levels] present in test data point [True]
42 Text feature [observed] present in test data point [True]
43 Text feature [enzyme] present in test data point [True]
45 Text feature [genes] present in test data point [True]
46 Text feature [sequences] present in test data point [True]
47 Text feature [knockdown] present in test data point [True]
48 Text feature [tagged] present in test data point [True]
49 Text feature [specific] present in test data point [True]
50 Text feature [result] present in test data point [True]
51 Text feature [targets] present in test data point [True]
52 Text feature [different] present in test data point [True]
54 Text feature [samples] present in test data point [True]
55 Text feature [endogenous] present in test data point [True]
57 Text feature [myeloid] present in test data point [True]
58 Text feature [using] present in test data point [True]
59 Text feature [rather] present in test data point [True]
60 Text feature [lymphoma] present in test data point [True]
61 Text feature [suggest] present in test data point [True]
62 Text feature [substrate] present in test data point [True]
64 Text feature [recent] present in test data point [True]
65 Text feature [determined] present in test data point [True]
66 Text feature [compared] present in test data point [True]
67 Text feature [3b] present in test data point [True]
68 Text feature [recognition] present in test data point [True]
69 Text feature [wild] present in test data point [True]
70 Text feature [together] present in test data point [True]
71 Text feature [complexes] present in test data point [True]
72 Text feature [wt] present in test data point [True]
73 Text feature [confirmed] present in test data point [True]
74 Text feature [expression] present in test data point [True]
75 Text feature [limited] present in test data point [True]
76 Text feature [expressed] present in test data point [True]
77 Text feature [found] present in test data point [True]
78 Text feature [increase] present in test data point [True]
79 Text feature [gel] present in test data point [True]
80 Text feature [2009] present in test data point [True]
81 Text feature [significant] present in test data point [True]
82 Text feature [many] present in test data point [True]
83 Text feature [figure] present in test data point [True]
84 Text feature [cells] present in test data point [True]
85 Text feature [consequences] present in test data point [True]
88 Text feature [type] present in test data point [True]
89 Text feature [analysis] present in test data point [True]
90 Text feature [2012] present in test data point [True]
91 Text feature [complex] present in test data point [True]
92 Text feature [events] present in test data point [True]
93 Text feature [selective] present in test data point [True]
94 Text feature [associated] present in test data point [True]
95 Text feature [10] present in test data point [True]
96 Text feature [heterozygous] present in test data point [True]
97 Text feature [subsequent] present in test data point [True]
98 Text feature [confirm] present in test data point [True]
99 Text feature [go] present in test data point [True]
100 Text feature [several] present in test data point [True]
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [77]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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_, ep
s=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)
```

```
for alpha = 5
Log Loss : 1.0324353214410276
for alpha = 11
Log Loss : 1.0455162765969066
for alpha = 15
Log Loss : 1.0698125124799183
for alpha = 21
Log Loss : 1.0737531756681298
for alpha = 31
Log Loss : 1.070815814927221
for alpha = 41
Log Loss : 1.0829305457476717
for alpha = 51
Log Loss : 1.0797813686226787
for alpha = 99
Log Loss : 1.0919515108240687
```

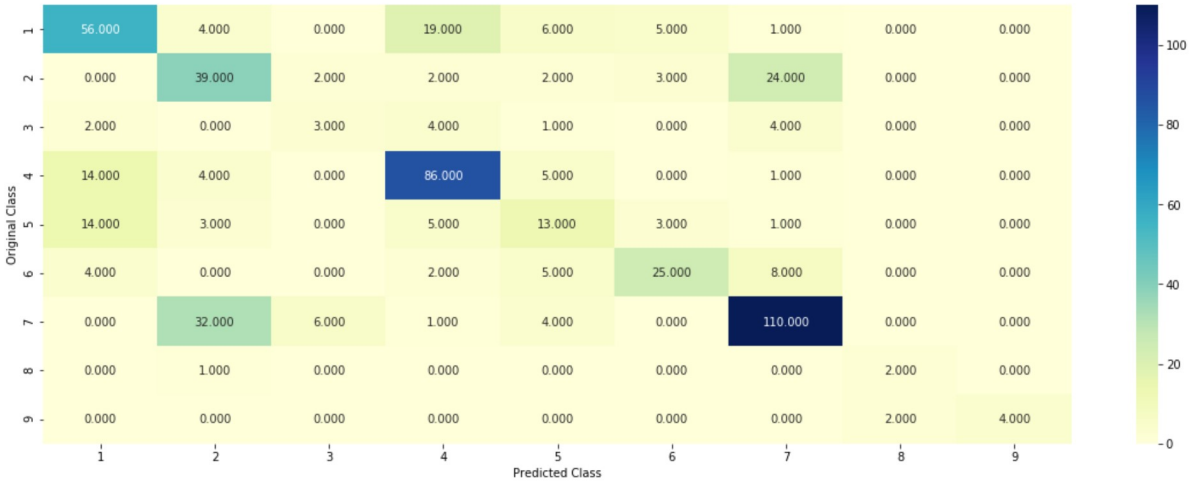



```
For values of best alpha =  5 The train log loss is: 0.4953698981613214
For values of best alpha =  5 The cross validation log loss is: 1.03243532144102
76
For values of best alpha =  5 The test log loss is: 1.0399375756964184
```

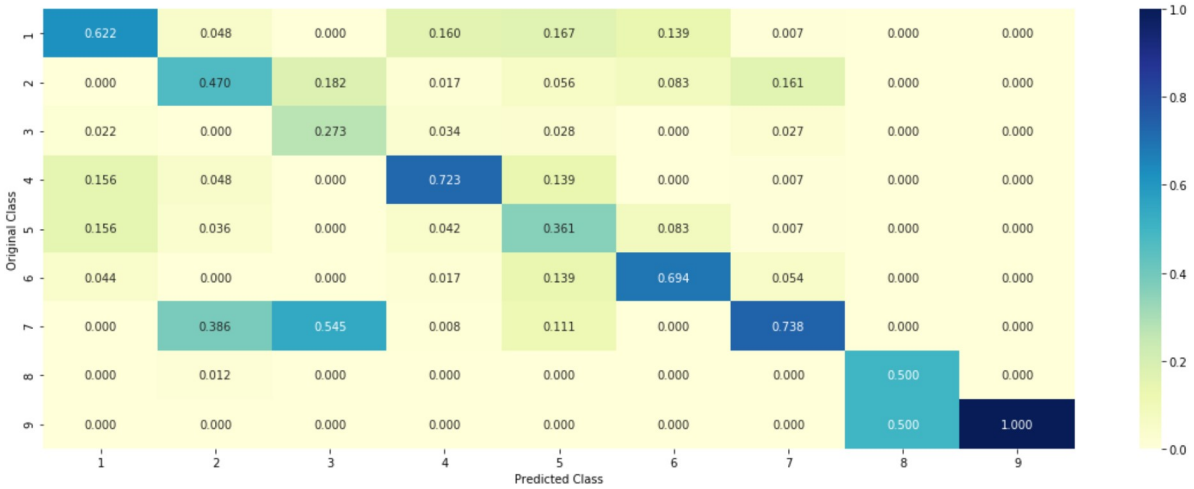
4.2.2. Testing the model with best hyper paramters

In [78]:
```
# 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-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

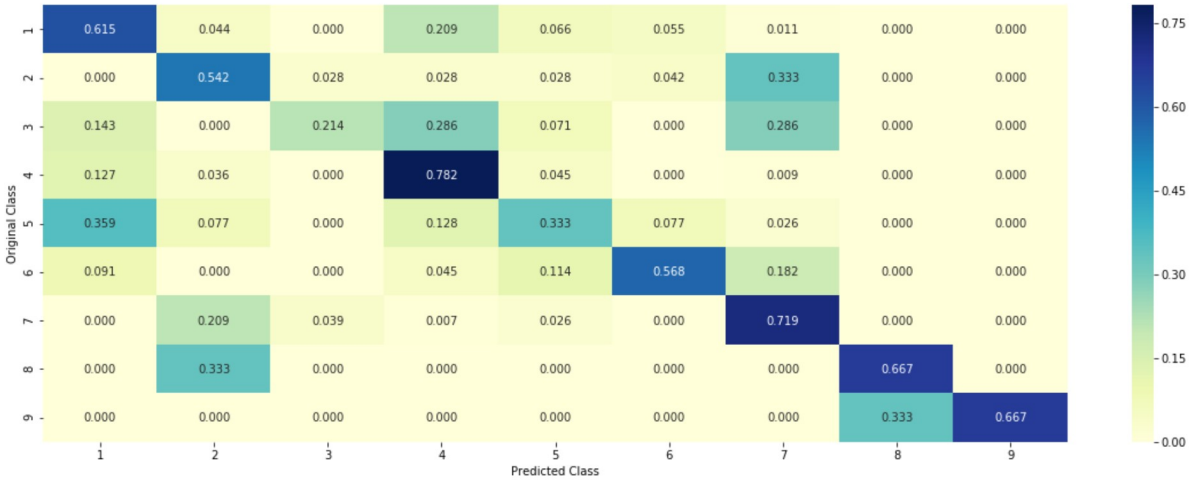
```
Log loss : 1.0324353214410276
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [79]:
```
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 c
lasses",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 6
The  5  nearest neighbours of the test points belongs to classes [6 6 6 6 6]
Fequency of nearest points : Counter({6: 5})
```

4.2.4. Sample Query Point-2

In [80]:
```
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 belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

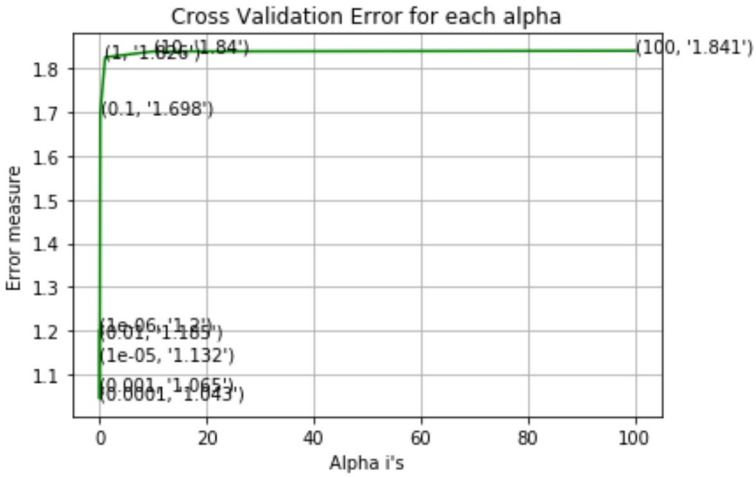
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [90]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.199861710928426
for alpha = 1e-05
Log Loss : 1.1323747623087266
for alpha = 0.0001
Log Loss : 1.043318050124558
for alpha = 0.001
Log Loss : 1.0650769137007332
for alpha = 0.01
Log Loss : 1.1847760369806273
for alpha = 0.1
Log Loss : 1.6984241160427438
for alpha = 1
Log Loss : 1.8262475042201562
for alpha = 10
Log Loss : 1.839684737820332
for alpha = 100
Log Loss : 1.841167280856162
```

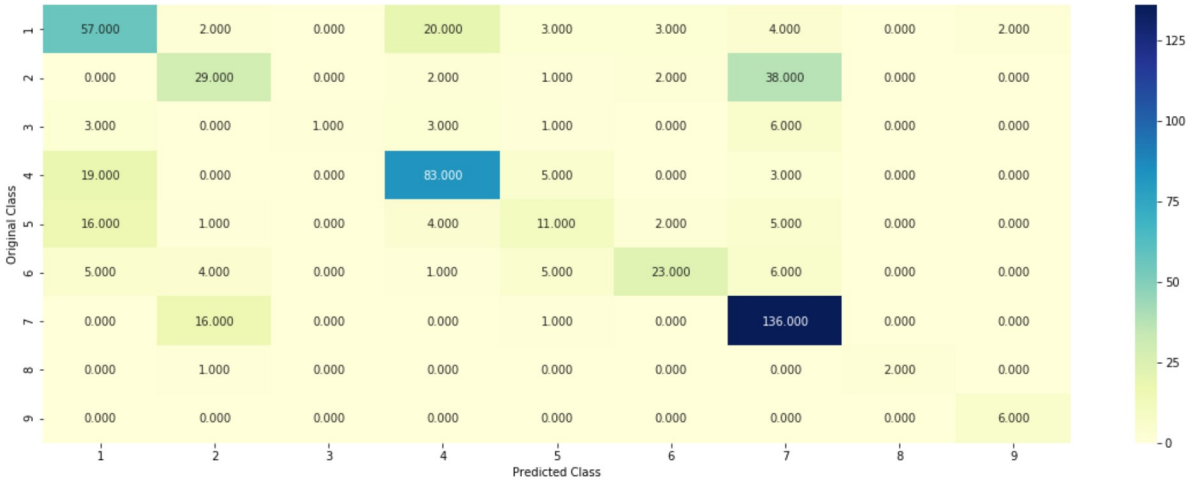



```
For values of best alpha =  0.0001 The train log loss is: 0.4482233327737296
For values of best alpha =  0.0001 The cross validation log loss is: 1.043318050
124558
For values of best alpha =  0.0001 The test log loss is: 1.012701665464298
```

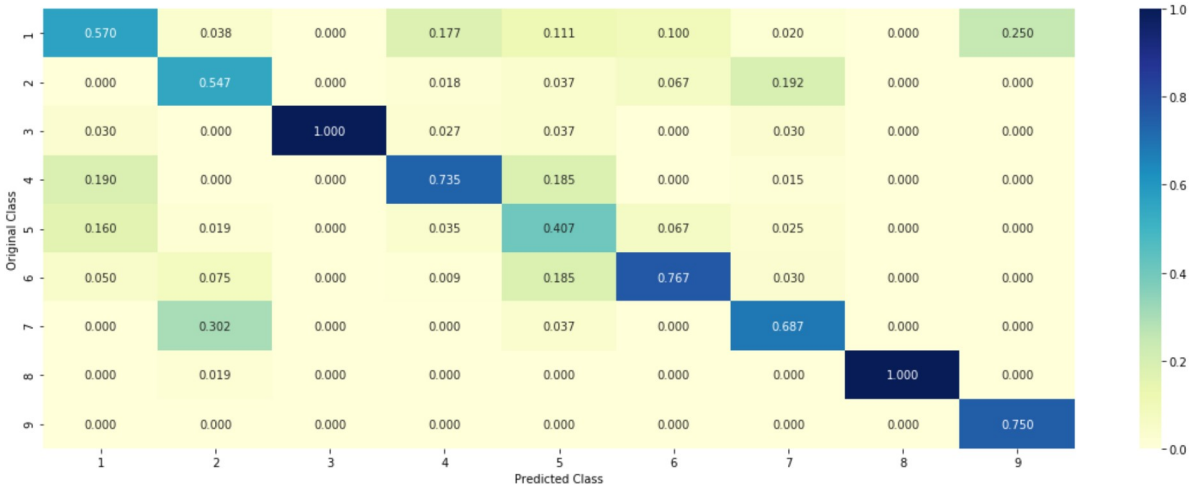
4.3.1.2. Testing the model with best hyper paramters

In [91]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

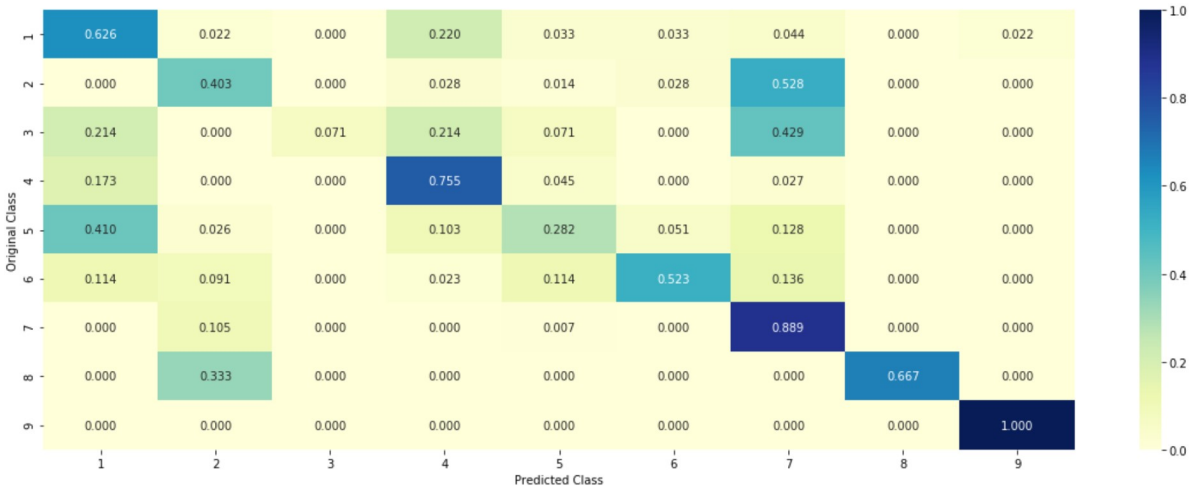
```
Log loss : 1.043318050124558
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Feature Importance

In [92]:
```
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 [93]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.152  0.0137 0.0038 0.0319 0.0919 0.7    0.002
9 0.0027 0.0012]]
Actual Class : 6
--------------------------------------------------
103 Text feature [values] present in test data point [True]
114 Text feature [interaction] present in test data point [True]
122 Text feature [studied] present in test data point [True]
127 Text feature [individuals] present in test data point [True]
134 Text feature [site] present in test data point [True]
138 Text feature [substitutions] present in test data point [True]
140 Text feature [substitution] present in test data point [True]
143 Text feature [ring] present in test data point [True]
157 Text feature [cause] present in test data point [True]
163 Text feature [development] present in test data point [True]
168 Text feature [000] present in test data point [True]
176 Text feature [60] present in test data point [True]
179 Text feature [associated] present in test data point [True]
180 Text feature [important] present in test data point [True]
181 Text feature [considered] present in test data point [True]
184 Text feature [model] present in test data point [True]
187 Text feature [five] present in test data point [True]
192 Text feature [showing] present in test data point [True]
195 Text feature [deleterious] present in test data point [True]
196 Text feature [characteristics] present in test data point [True]
199 Text feature [ovarian] present in test data point [True]
203 Text feature [significant] present in test data point [True]
208 Text feature [overall] present in test data point [True]
212 Text feature [classified] present in test data point [True]
213 Text feature [formation] present in test data point [True]
222 Text feature [95] present in test data point [True]
225 Text feature [clinically] present in test data point [True]
227 Text feature [brca1] present in test data point [True]
229 Text feature [history] present in test data point [True]
233 Text feature [family] present in test data point [True]
234 Text feature [approach] present in test data point [True]
237 Text feature [expected] present in test data point [True]
241 Text feature [time] present in test data point [True]
246 Text feature [potential] present in test data point [True]
247 Text feature [identified] present in test data point [True]
248 Text feature [90] present in test data point [True]
250 Text feature [evidence] present in test data point [True]
254 Text feature [multiple] present in test data point [True]
257 Text feature [calculated] present in test data point [True]
258 Text feature [statistical] present in test data point [True]
261 Text feature [survival] present in test data point [True]
266 Text feature [loss] present in test data point [True]
273 Text feature [times] present in test data point [True]
275 Text feature [tables] present in test data point [True]
277 Text feature [17] present in test data point [True]
287 Text feature [missense] present in test data point [True]
295 Text feature [30] present in test data point [True]
304 Text feature [19] present in test data point [True]
305 Text feature [72] present in test data point [True]
307 Text feature [low] present in test data point [True]
310 Text feature [none] present in test data point [True]
312 Text feature [residues] present in test data point [True]
315 Text feature [side] present in test data point [True]
321 Text feature [relevant] present in test data point [True]
323 Text feature [database] present in test data point [True]
329 Text feature [terminus] present in test data point [True]
330 Text feature [binding] present in test data point [True]
333 Text feature [risk] present in test data point [True]
335 Text feature [27] present in test data point [True]
```

4.3.1.3.2. Incorrectly Classified point

In [94]:
```
test_point_index = 55
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

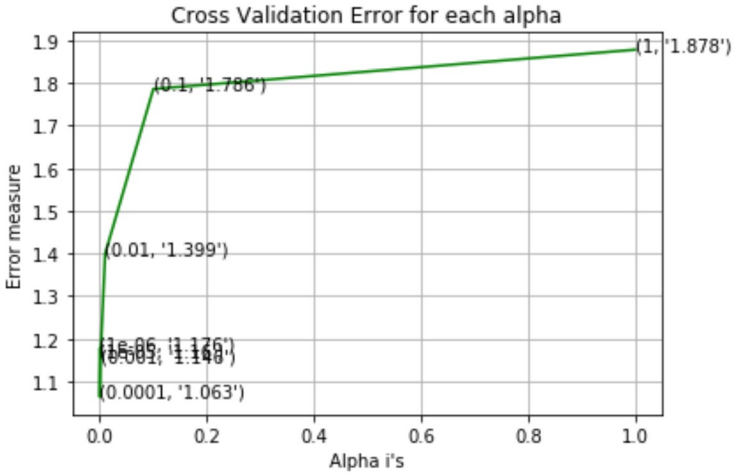
```
Predicted Class : 5
Predicted Class Probabilities: [[0.0199 0.0653 0.0016 0.0093 0.5127 0.0145 0.373
8 0.0016 0.0011]]
Actual Class : 7
--------------------------------------------------
126 Text feature [efficacy] present in test data point [True]
150 Text feature [discovery] present in test data point [True]
152 Text feature [selective] present in test data point [True]
167 Text feature [context] present in test data point [True]
168 Text feature [larger] present in test data point [True]
169 Text feature [s3] present in test data point [True]
174 Text feature [amplification] present in test data point [True]
176 Text feature [assays] present in test data point [True]
179 Text feature [vitro] present in test data point [True]
183 Text feature [structures] present in test data point [True]
186 Text feature [similarly] present in test data point [True]
188 Text feature [assessment] present in test data point [True]
189 Text feature [alterations] present in test data point [True]
191 Text feature [correlation] present in test data point [True]
193 Text feature [rare] present in test data point [True]
195 Text feature [2b] present in test data point [True]
197 Text feature [effect] present in test data point [True]
199 Text feature [addition] present in test data point [True]
201 Text feature [transforming] present in test data point [True]
204 Text feature [numbers] present in test data point [True]
207 Text feature [effective] present in test data point [True]
208 Text feature [assay] present in test data point [True]
210 Text feature [insight] present in test data point [True]
211 Text feature [stable] present in test data point [True]
212 Text feature [extracellular] present in test data point [True]
214 Text feature [s1] present in test data point [True]
215 Text feature [co] present in test data point [True]
218 Text feature [experimental] present in test data point [True]
222 Text feature [contrast] present in test data point [True]
224 Text feature [kd] present in test data point [True]
225 Text feature [number] present in test data point [True]
227 Text feature [functional] present in test data point [True]
228 Text feature [assessed] present in test data point [True]
229 Text feature [difference] present in test data point [True]
230 Text feature [represent] present in test data point [True]
232 Text feature [variants] present in test data point [True]
234 Text feature [frequently] present in test data point [True]
235 Text feature [unique] present in test data point [True]
236 Text feature [200] present in test data point [True]
238 Text feature [sensitivity] present in test data point [True]
239 Text feature [formed] present in test data point [True]
242 Text feature [purified] present in test data point [True]
243 Text feature [vivo] present in test data point [True]
246 Text feature [assess] present in test data point [True]
247 Text feature [molecule] present in test data point [True]
248 Text feature [general] present in test data point [True]
252 Text feature [core] present in test data point [True]
255 Text feature [small] present in test data point [True]
256 Text feature [specificity] present in test data point [True]
258 Text feature [include] present in test data point [True]
259 Text feature [table] present in test data point [True]
262 Text feature [clear] present in test data point [True]
264 Text feature [despite] present in test data point [True]
265 Text feature [targets] present in test data point [True]
267 Text feature [altered] present in test data point [True]
268 Text feature [similar] present in test data point [True]
269 Text feature [variant] present in test data point [True]
270 Text feature [sequence] present in test data point [True]
271 Text feature [crystal] present in test data point [True]
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [86]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

alpha = [10 ** x for x in range(-6, 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
```

```
for alpha = 1e-06
Log Loss : 1.175950158139626
for alpha = 1e-05
Log Loss : 1.1600578309735452
for alpha = 0.0001
Log Loss : 1.062988230671672
for alpha = 0.001
Log Loss : 1.1458351400061557
for alpha = 0.01
Log Loss : 1.3994092788184804
for alpha = 0.1
Log Loss : 1.7859204165173244
for alpha = 1
Log Loss : 1.8781718379282843
```

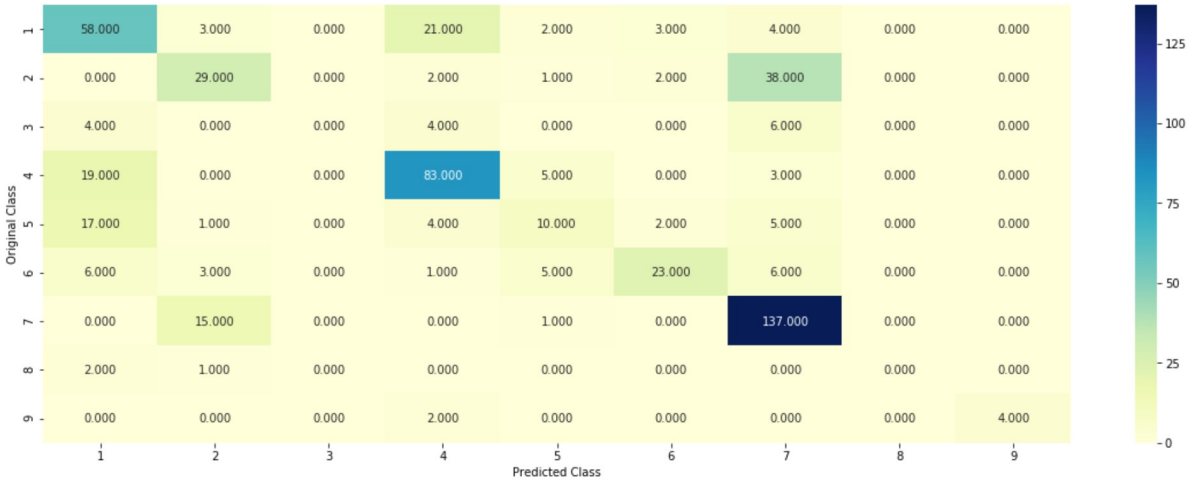



```
For values of best alpha =  0.0001 The train log loss is: 0.4386801548788915
For values of best alpha =  0.0001 The cross validation log loss is: 1.062988230
671672
For values of best alpha =  0.0001 The test log loss is: 1.029170854019561
```

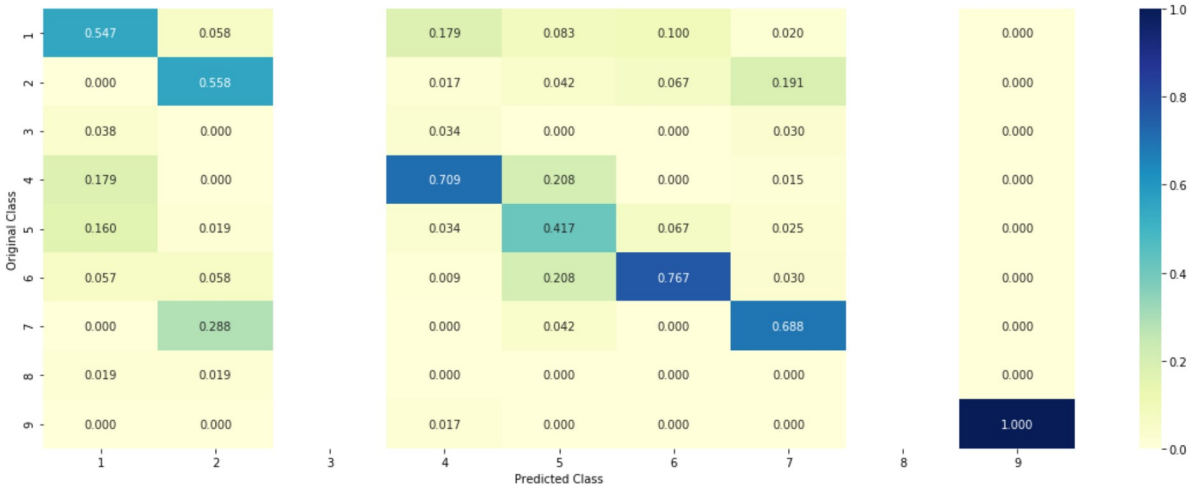
4.3.2.2. Testing model with best hyper parameters

In [87]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

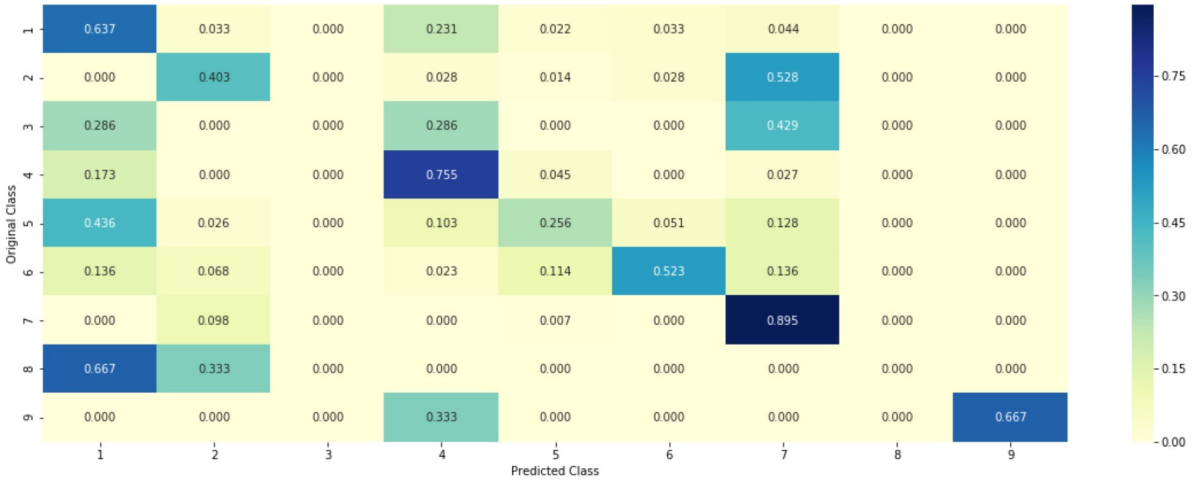
```
Log loss : 1.062988230671672
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Feature Importance, Correctly Classified point

In [88]:
```
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 = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[1.651e-01 1.480e-02 1.600e-03 4.030e-02 8.650e-
02 6.851e-01 4.300e-03
  2.100e-03 3.000e-04]]
Actual Class : 6
--------------------------------------------------
102 Text feature [values] present in test data point [True]
115 Text feature [interaction] present in test data point [True]
126 Text feature [studied] present in test data point [True]
130 Text feature [individuals] present in test data point [True]
133 Text feature [site] present in test data point [True]
134 Text feature [substitutions] present in test data point [True]
136 Text feature [substitution] present in test data point [True]
137 Text feature [ring] present in test data point [True]
160 Text feature [cause] present in test data point [True]
161 Text feature [development] present in test data point [True]
163 Text feature [000] present in test data point [True]
180 Text feature [60] present in test data point [True]
181 Text feature [showing] present in test data point [True]
186 Text feature [deleterious] present in test data point [True]
187 Text feature [characteristics] present in test data point [True]
192 Text feature [model] present in test data point [True]
195 Text feature [associated] present in test data point [True]
198 Text feature [considered] present in test data point [True]
199 Text feature [important] present in test data point [True]
200 Text feature [ovarian] present in test data point [True]
201 Text feature [five] present in test data point [True]
205 Text feature [significant] present in test data point [True]
207 Text feature [classified] present in test data point [True]
214 Text feature [overall] present in test data point [True]
218 Text feature [formation] present in test data point [True]
219 Text feature [history] present in test data point [True]
224 Text feature [approach] present in test data point [True]
227 Text feature [95] present in test data point [True]
228 Text feature [brca1] present in test data point [True]
230 Text feature [family] present in test data point [True]
231 Text feature [clinically] present in test data point [True]
235 Text feature [expected] present in test data point [True]
241 Text feature [time] present in test data point [True]
247 Text feature [90] present in test data point [True]
251 Text feature [potential] present in test data point [True]
255 Text feature [calculated] present in test data point [True]
258 Text feature [identified] present in test data point [True]
259 Text feature [evidence] present in test data point [True]
267 Text feature [multiple] present in test data point [True]
268 Text feature [statistical] present in test data point [True]
269 Text feature [loss] present in test data point [True]
272 Text feature [survival] present in test data point [True]
274 Text feature [times] present in test data point [True]
277 Text feature [tables] present in test data point [True]
288 Text feature [30] present in test data point [True]
289 Text feature [17] present in test data point [True]
294 Text feature [missense] present in test data point [True]
302 Text feature [72] present in test data point [True]
307 Text feature [none] present in test data point [True]
312 Text feature [database] present in test data point [True]
318 Text feature [200] present in test data point [True]
323 Text feature [19] present in test data point [True]
333 Text feature [side] present in test data point [True]
336 Text feature [predicted] present in test data point [True]
340 Text feature [56] present in test data point [True]
343 Text feature [residues] present in test data point [True]
346 Text feature [screening] present in test data point [True]
349 Text feature [value] present in test data point [True]
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [89]:
```
test_point_index = 2
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

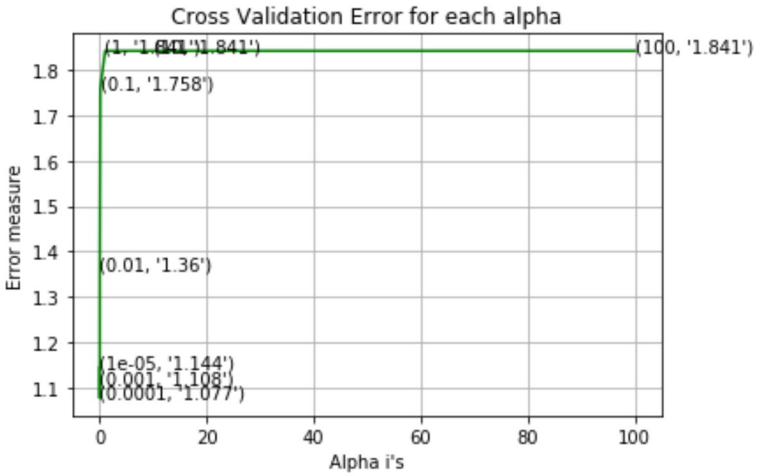
```
Predicted Class : 1
Predicted Class Probabilities: [[4.349e-01 3.200e-03 5.000e-03 1.442e-01 3.934e-
01 1.580e-02 1.100e-03
  2.300e-03 1.000e-04]]
Actual Class : 4
--------------------------------------------------
36 Text feature [surface] present in test data point [True]
91 Text feature [fraction] present in test data point [True]
97 Text feature [normalized] present in test data point [True]
107 Text feature [deletion] present in test data point [True]
117 Text feature [hydrophobic] present in test data point [True]
124 Text feature [repeats] present in test data point [True]
126 Text feature [specificity] present in test data point [True]
138 Text feature [signal] present in test data point [True]
158 Text feature [calculated] present in test data point [True]
161 Text feature [panel] present in test data point [True]
198 Text feature [upon] present in test data point [True]
199 Text feature [brct] present in test data point [True]
202 Text feature [defined] present in test data point [True]
205 Text feature [repeat] present in test data point [True]
215 Text feature [loss] present in test data point [True]
218 Text feature [transcriptional] present in test data point [True]
221 Text feature [screening] present in test data point [True]
226 Text feature [box] present in test data point [True]
229 Text feature [displayed] present in test data point [True]
236 Text feature [59] present in test data point [True]
241 Text feature [structure] present in test data point [True]
250 Text feature [affect] present in test data point [True]
257 Text feature [function] present in test data point [True]
264 Text feature [page] present in test data point [True]
273 Text feature [families] present in test data point [True]
274 Text feature [pathogenic] present in test data point [True]
276 Text feature [http] present in test data point [True]
283 Text feature [essential] present in test data point [True]
285 Text feature [molecules] present in test data point [True]
286 Text feature [indicated] present in test data point [True]
288 Text feature [ovarian] present in test data point [True]
298 Text feature [score] present in test data point [True]
305 Text feature [percentage] present in test data point [True]
306 Text feature [functions] present in test data point [True]
308 Text feature [region] present in test data point [True]
311 Text feature [encoding] present in test data point [True]
313 Text feature [2001] present in test data point [True]
316 Text feature [remaining] present in test data point [True]
319 Text feature [core] present in test data point [True]
320 Text feature [population] present in test data point [True]
323 Text feature [binding] present in test data point [True]
327 Text feature [carrying] present in test data point [True]
328 Text feature [vitro] present in test data point [True]
330 Text feature [39] present in test data point [True]
333 Text feature [domains] present in test data point [True]
335 Text feature [next] present in test data point [True]
336 Text feature [assessment] present in test data point [True]
337 Text feature [www] present in test data point [True]
340 Text feature [identify] present in test data point [True]
341 Text feature [somatic] present in test data point [True]
347 Text feature [less] present in test data point [True]
348 Text feature [deficient] present in test data point [True]
349 Text feature [value] present in test data point [True]
351 Text feature [deletions] present in test data point [True]
355 Text feature [fold] present in test data point [True]
357 Text feature [early] present in test data point [True]
358 Text feature [absence] present in test data point [True]
359 Text feature [peptide] present in test data point [True]
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [95]:
```
# read more about support vector machines with linear kernals here http://scikit-le
arn.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, proba
bility=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_
shape='ovr', random_state=None)

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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='hing
e', 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_, ep
s=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)
```

```
for C = 1e-05
Log Loss : 1.144037457907447
for C = 0.0001
Log Loss : 1.076836039361882
for C = 0.001
Log Loss : 1.1075561353640337
for C = 0.01
Log Loss : 1.3604052662738253
for C = 0.1
Log Loss : 1.7581086360162492
for C = 1
Log Loss : 1.8413488905407596
for C = 10
Log Loss : 1.8414538392530517
for C = 100
Log Loss : 1.8414538904785558
```

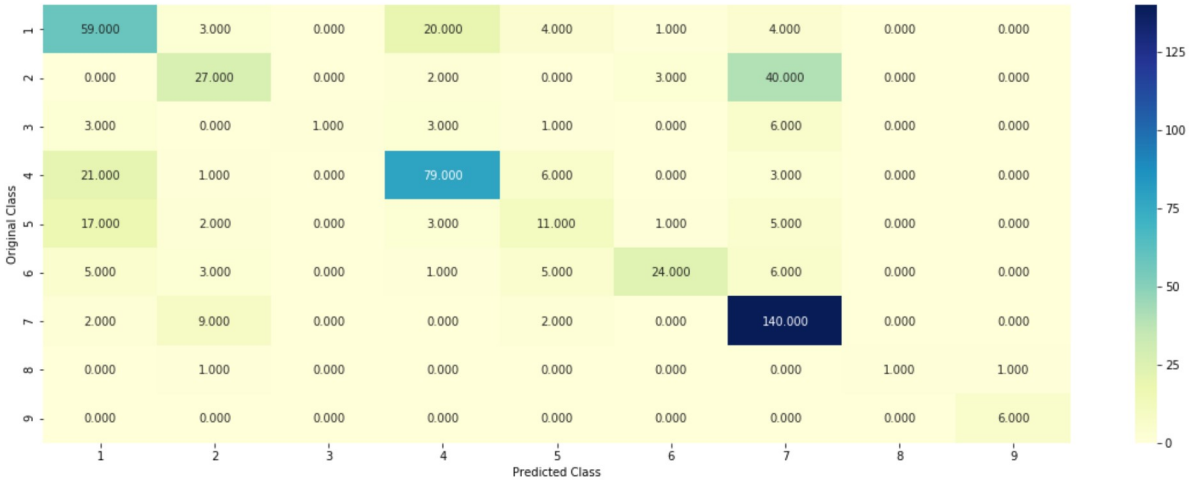



```
For values of best alpha =  0.0001 The train log loss is: 0.47239885236748824
For values of best alpha =  0.0001 The cross validation log loss is: 1.076836039
361882
For values of best alpha =  0.0001 The test log loss is: 1.0564605926840567
```

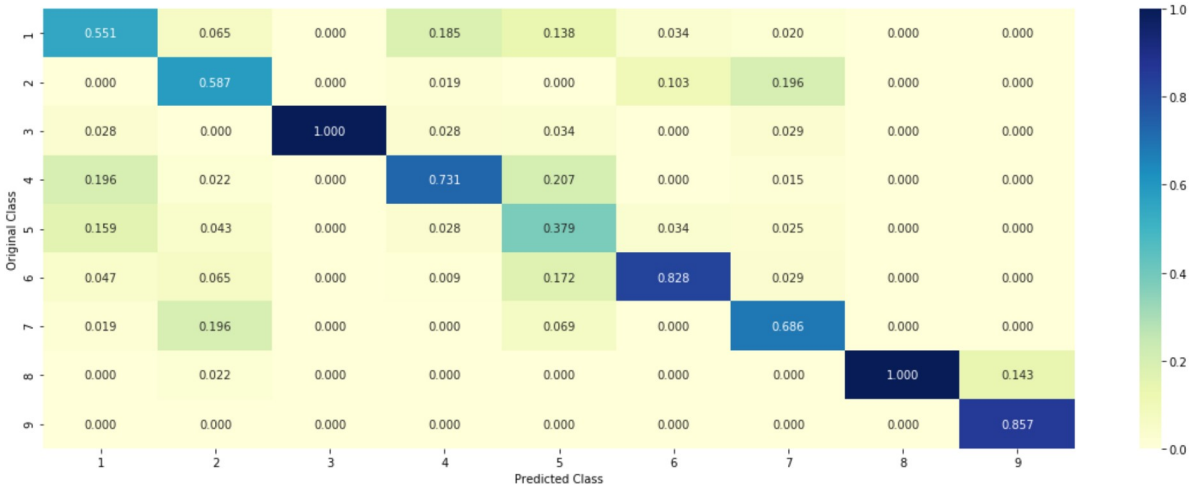
4.4.2. Testing model with best hyper parameters

In [96]:
```
# read more about support vector machines with linear kernals here http://scikit-le
arn.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, proba
bility=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_
shape='ovr', random_state=None)

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='bal
anced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_sta
te=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,c
v_y, clf)
```

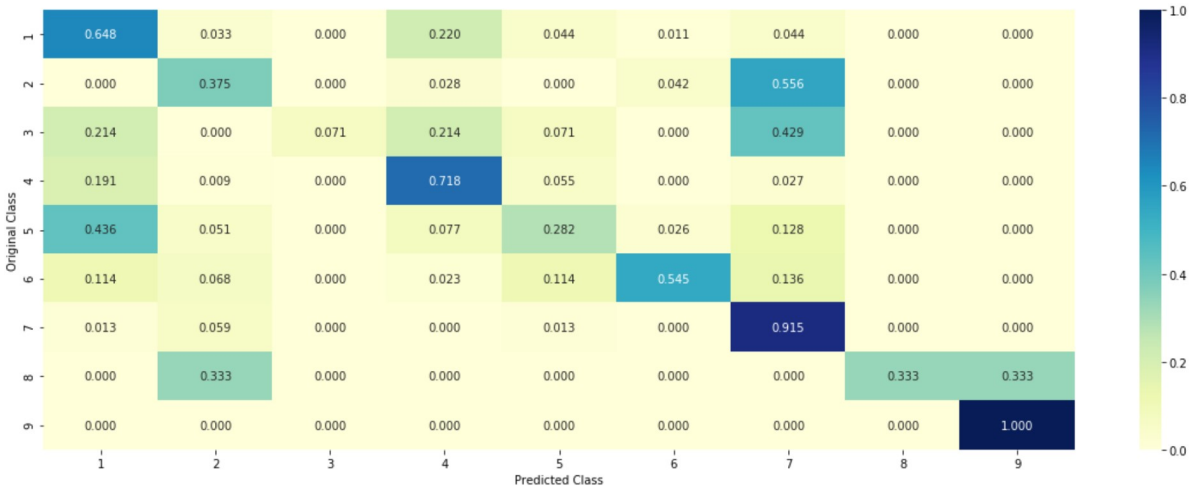
```
Log loss : 1.076836039361882
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [99]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_sta
te=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 1000
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2131 0.0659 0.012  0.0404 0.0825 0.5196 0.063
9 0.0017 0.0009]]
Actual Class : 6
--------------------------------------------------
100 Text feature [values] present in test data point [True]
103 Text feature [interaction] present in test data point [True]
106 Text feature [studied] present in test data point [True]
109 Text feature [development] present in test data point [True]
113 Text feature [substitution] present in test data point [True]
118 Text feature [site] present in test data point [True]
122 Text feature [60] present in test data point [True]
124 Text feature [000] present in test data point [True]
126 Text feature [showing] present in test data point [True]
127 Text feature [substitutions] present in test data point [True]
129 Text feature [cause] present in test data point [True]
131 Text feature [individuals] present in test data point [True]
204 Text feature [five] present in test data point [True]
207 Text feature [considered] present in test data point [True]
208 Text feature [associated] present in test data point [True]
210 Text feature [significant] present in test data point [True]
211 Text feature [ring] present in test data point [True]
214 Text feature [times] present in test data point [True]
219 Text feature [multiple] present in test data point [True]
228 Text feature [deleterious] present in test data point [True]
229 Text feature [classified] present in test data point [True]
230 Text feature [identified] present in test data point [True]
233 Text feature [expected] present in test data point [True]
234 Text feature [model] present in test data point [True]
235 Text feature [90] present in test data point [True]
239 Text feature [family] present in test data point [True]
244 Text feature [potential] present in test data point [True]
245 Text feature [factor] present in test data point [True]
249 Text feature [200] present in test data point [True]
251 Text feature [formation] present in test data point [True]
253 Text feature [survival] present in test data point [True]
255 Text feature [characteristics] present in test data point [True]
258 Text feature [terminus] present in test data point [True]
260 Text feature [important] present in test data point [True]
262 Text feature [tables] present in test data point [True]
263 Text feature [er] present in test data point [True]
270 Text feature [calculated] present in test data point [True]
271 Text feature [carried] present in test data point [True]
272 Text feature [overall] present in test data point [True]
276 Text feature [clinically] present in test data point [True]
278 Text feature [six] present in test data point [True]
279 Text feature [17] present in test data point [True]
281 Text feature [according] present in test data point [True]
286 Text feature [none] present in test data point [True]
287 Text feature [30] present in test data point [True]
290 Text feature [history] present in test data point [True]
292 Text feature [missense] present in test data point [True]
295 Text feature [including] present in test data point [True]
296 Text feature [brca1] present in test data point [True]
301 Text feature [binding] present in test data point [True]
302 Text feature [approach] present in test data point [True]
303 Text feature [sequence] present in test data point [True]
305 Text feature [27] present in test data point [True]
308 Text feature [proportion] present in test data point [True]
310 Text feature [sequencing] present in test data point [True]
311 Text feature [predicted] present in test data point [True]
312 Text feature [loss] present in test data point [True]
315 Text feature [statistical] present in test data point [True]
317 Text feature [ovarian] present in test data point [True]
```

4.3.3.2. For Incorrectly classified point

In [98]:
```
test_point_index = 10
no_feature = 1000
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_oneho
tCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Ge
ne'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6292 0.0128 0.0036 0.276  0.0236 0.0264 0.026
8 0.0008 0.0008]]
Actual Class : 4
--------------------------------------------------
62 Text feature [surface] present in test data point [True]
63 Text feature [hydrophobic] present in test data point [True]
209 Text feature [deletion] present in test data point [True]
212 Text feature [displayed] present in test data point [True]
213 Text feature [repeats] present in test data point [True]
214 Text feature [screening] present in test data point [True]
215 Text feature [signal] present in test data point [True]
216 Text feature [box] present in test data point [True]
217 Text feature [specificity] present in test data point [True]
219 Text feature [less] present in test data point [True]
220 Text feature [loss] present in test data point [True]
223 Text feature [nucleus] present in test data point [True]
226 Text feature [upon] present in test data point [True]
229 Text feature [39] present in test data point [True]
231 Text feature [vivo] present in test data point [True]
236 Text feature [transcriptional] present in test data point [True]
237 Text feature [molecules] present in test data point [True]
238 Text feature [function] present in test data point [True]
239 Text feature [binding] present in test data point [True]
243 Text feature [fig] present in test data point [True]
244 Text feature [fold] present in test data point [True]
250 Text feature [defined] present in test data point [True]
251 Text feature [vitro] present in test data point [True]
252 Text feature [somatic] present in test data point [True]
257 Text feature [deletions] present in test data point [True]
258 Text feature [structure] present in test data point [True]
372 Text feature [type] present in test data point [True]
373 Text feature [transcription] present in test data point [True]
378 Text feature [identify] present in test data point [True]
379 Text feature [wild] present in test data point [True]
382 Text feature [2001] present in test data point [True]
383 Text feature [negative] present in test data point [True]
389 Text feature [cause] present in test data point [True]
391 Text feature [essential] present in test data point [True]
395 Text feature [reduction] present in test data point [True]
399 Text feature [present] present in test data point [True]
400 Text feature [25] present in test data point [True]
401 Text feature [might] present in test data point [True]
402 Text feature [within] present in test data point [True]
403 Text feature [frequency] present in test data point [True]
404 Text feature [75] present in test data point [True]
405 Text feature [regulatory] present in test data point [True]
406 Text feature [critical] present in test data point [True]
408 Text feature [regions] present in test data point [True]
410 Text feature [possible] present in test data point [True]
412 Text feature [proteins] present in test data point [True]
413 Text feature [nucleotide] present in test data point [True]
415 Text feature [showing] present in test data point [True]
418 Text feature [conserved] present in test data point [True]
419 Text feature [region] present in test data point [True]
421 Text feature [therefore] present in test data point [True]
422 Text feature [34] present in test data point [True]
424 Text feature [42] present in test data point [True]
425 Text feature [provide] present in test data point [True]
428 Text feature [structural] present in test data point [True]
429 Text feature [large] present in test data point [True]
432 Text feature [absence] present in test data point [True]
433 Text feature [domains] present in test data point [True]
434 Text feature [core] present in test data point [True]
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

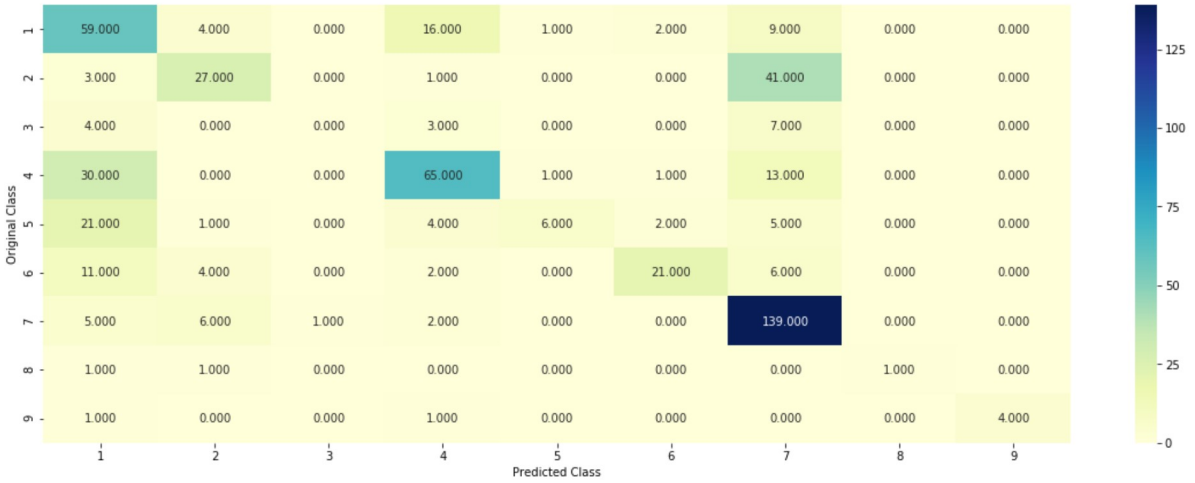
In [100]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


# 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]))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2179063598108473
for n_estimators = 100 and max depth =  10
Log Loss : 1.2342589469356224
for n_estimators = 200 and max depth =  5
Log Loss : 1.2040839126914036
for n_estimators = 200 and max depth =  10
Log Loss : 1.2276468586320186
for n_estimators = 500 and max depth =  5
Log Loss : 1.197866726793652
for n_estimators = 500 and max depth =  10
Log Loss : 1.225371256196731
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1998964770895288
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2231203080378432
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1999988047581869
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2231800898796503
For values of best estimator =  500 The train log loss is: 0.8748742050071634
For values of best estimator =  500 The cross validation log loss is: 1.19786672
67936522
For values of best estimator =  500 The test log loss is: 1.2107589723522947
```

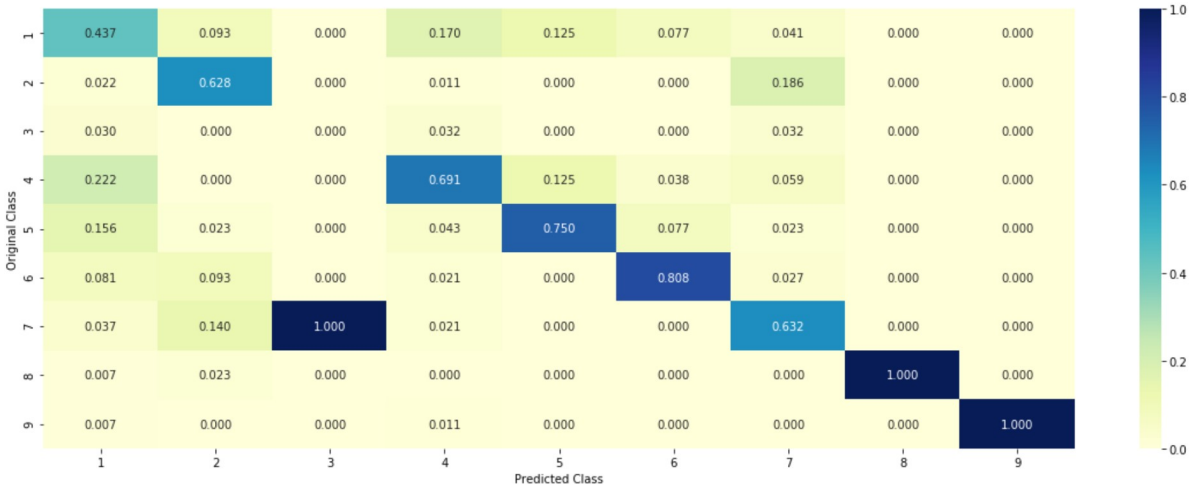
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [101]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

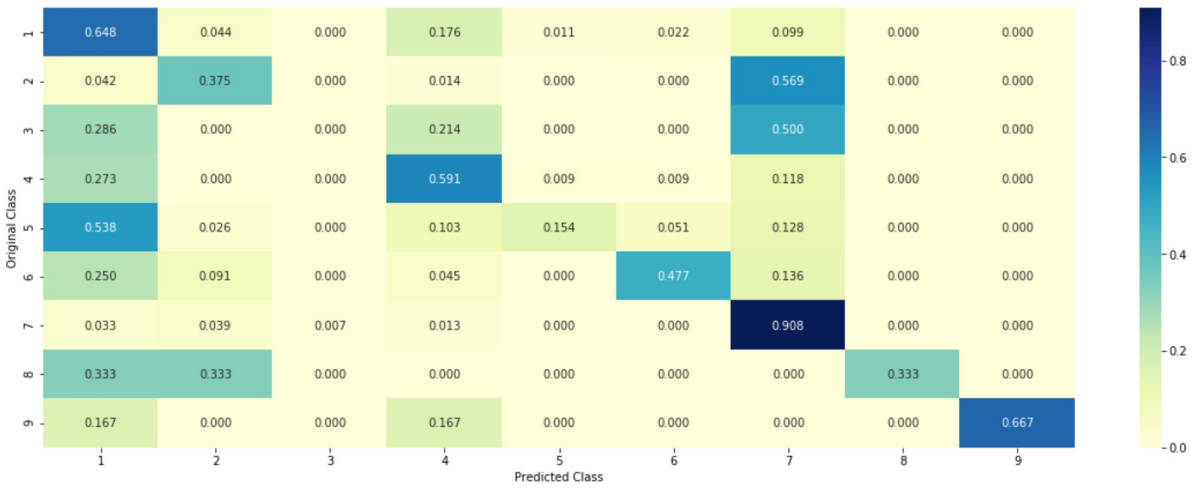
```
Log loss : 1.1978667267936522
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```




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




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [102]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=2000, criterion='gini', max_depth=5, ran
dom_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 = 1000
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_oneh
otCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index]
, no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1565 0.0091 0.0099 0.0783 0.0949 0.636  0.011
6 0.0022 0.0015]]
Actual Class : 6
--------------------------------------------------
8 Text feature [function] present in test data point [True]
10 Text feature [missense] present in test data point [True]
13 Text feature [brca1] present in test data point [True]
15 Text feature [deleterious] present in test data point [True]
17 Text feature [functional] present in test data point [True]
18 Text feature [loss] present in test data point [True]
25 Text feature [variants] present in test data point [True]
33 Text feature [protein] present in test data point [True]
34 Text feature [pathogenic] present in test data point [True]
35 Text feature [neutral] present in test data point [True]
39 Text feature [ovarian] present in test data point [True]
42 Text feature [receptor] present in test data point [True]
45 Text feature [patients] present in test data point [True]
47 Text feature [expression] present in test data point [True]
48 Text feature [repair] present in test data point [True]
49 Text feature [classified] present in test data point [True]
50 Text feature [brca2] present in test data point [True]
51 Text feature [brct] present in test data point [True]
58 Text feature [clinical] present in test data point [True]
60 Text feature [cell] present in test data point [True]
62 Text feature [predicted] present in test data point [True]
64 Text feature [inactivation] present in test data point [True]
65 Text feature [proteins] present in test data point [True]
67 Text feature [variant] present in test data point [True]
71 Text feature [use] present in test data point [True]
72 Text feature [survival] present in test data point [True]
77 Text feature [ring] present in test data point [True]
82 Text feature [assays] present in test data point [True]
90 Text feature [functions] present in test data point [True]
91 Text feature [activity] present in test data point [True]
92 Text feature [dna] present in test data point [True]
94 Text feature [combined] present in test data point [True]
96 Text feature [potential] present in test data point [True]
98 Text feature [expected] present in test data point [True]
100 Text feature [likely] present in test data point [True]
105 Text feature [risk] present in test data point [True]
109 Text feature [core] present in test data point [True]
111 Text feature [type] present in test data point [True]
114 Text feature [evidence] present in test data point [True]
115 Text feature [domains] present in test data point [True]
116 Text feature [patient] present in test data point [True]
117 Text feature [history] present in test data point [True]
118 Text feature [classification] present in test data point [True]
119 Text feature [values] present in test data point [True]
122 Text feature [database] present in test data point [True]
123 Text feature [nuclear] present in test data point [True]
128 Text feature [families] present in test data point [True]
129 Text feature [sequence] present in test data point [True]
132 Text feature [splice] present in test data point [True]
133 Text feature [known] present in test data point [True]
134 Text feature [based] present in test data point [True]
135 Text feature [21] present in test data point [True]
136 Text feature [substitutions] present in test data point [True]
137 Text feature [binding] present in test data point [True]
139 Text feature [conserved] present in test data point [True]
141 Text feature [site] present in test data point [True]
142 Text feature [deficient] present in test data point [True]
143 Text feature [one] present in test data point [True]
146 Text feature [control] present in test data point [True]
```

4.5.3.2. Inorrectly Classified point

In [103]:
```
test_point_index = 100
no_feature = 1000
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_oneh
otCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index]
, no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3996 0.0236 0.0155 0.3455 0.0676 0.0918 0.024
9 0.009  0.0225]]
Actuall Class : 4
--------------------------------------------------
8 Text feature [function] present in test data point [True]
10 Text feature [missense] present in test data point [True]
13 Text feature [brca1] present in test data point [True]
17 Text feature [functional] present in test data point [True]
18 Text feature [loss] present in test data point [True]
21 Text feature [defective] present in test data point [True]
23 Text feature [cells] present in test data point [True]
33 Text feature [protein] present in test data point [True]
34 Text feature [pathogenic] present in test data point [True]
44 Text feature [downstream] present in test data point [True]
45 Text feature [patients] present in test data point [True]
47 Text feature [expression] present in test data point [True]
48 Text feature [repair] present in test data point [True]
49 Text feature [classified] present in test data point [True]
50 Text feature [brca2] present in test data point [True]
54 Text feature [sensitivity] present in test data point [True]
58 Text feature [clinical] present in test data point [True]
60 Text feature [cell] present in test data point [True]
65 Text feature [proteins] present in test data point [True]
67 Text feature [variant] present in test data point [True]
75 Text feature [damage] present in test data point [True]
91 Text feature [activity] present in test data point [True]
92 Text feature [dna] present in test data point [True]
96 Text feature [potential] present in test data point [True]
98 Text feature [expected] present in test data point [True]
100 Text feature [likely] present in test data point [True]
101 Text feature [basis] present in test data point [True]
102 Text feature [lines] present in test data point [True]
107 Text feature [affect] present in test data point [True]
111 Text feature [type] present in test data point [True]
114 Text feature [evidence] present in test data point [True]
116 Text feature [patient] present in test data point [True]
123 Text feature [nuclear] present in test data point [True]
124 Text feature [homozygous] present in test data point [True]
125 Text feature [pathway] present in test data point [True]
126 Text feature [affected] present in test data point [True]
127 Text feature [43] present in test data point [True]
128 Text feature [families] present in test data point [True]
129 Text feature [sequence] present in test data point [True]
132 Text feature [splice] present in test data point [True]
133 Text feature [known] present in test data point [True]
135 Text feature [21] present in test data point [True]
136 Text feature [substitutions] present in test data point [True]
139 Text feature [conserved] present in test data point [True]
141 Text feature [site] present in test data point [True]
143 Text feature [one] present in test data point [True]
146 Text feature [control] present in test data point [True]
148 Text feature [sequencing] present in test data point [True]
150 Text feature [enhanced] present in test data point [True]
152 Text feature [structure] present in test data point [True]
154 Text feature [wild] present in test data point [True]
155 Text feature [results] present in test data point [True]
156 Text feature [26] present in test data point [True]
158 Text feature [effect] present in test data point [True]
162 Text feature [previously] present in test data point [True]
171 Text feature [presence] present in test data point [True]
173 Text feature [splicing] present in test data point [True]
177 Text feature [gene] present in test data point [True]
186 Text feature [region] present in test data point [True]
```

4.5.3. Hyper paramter tuning (With Response Coding)

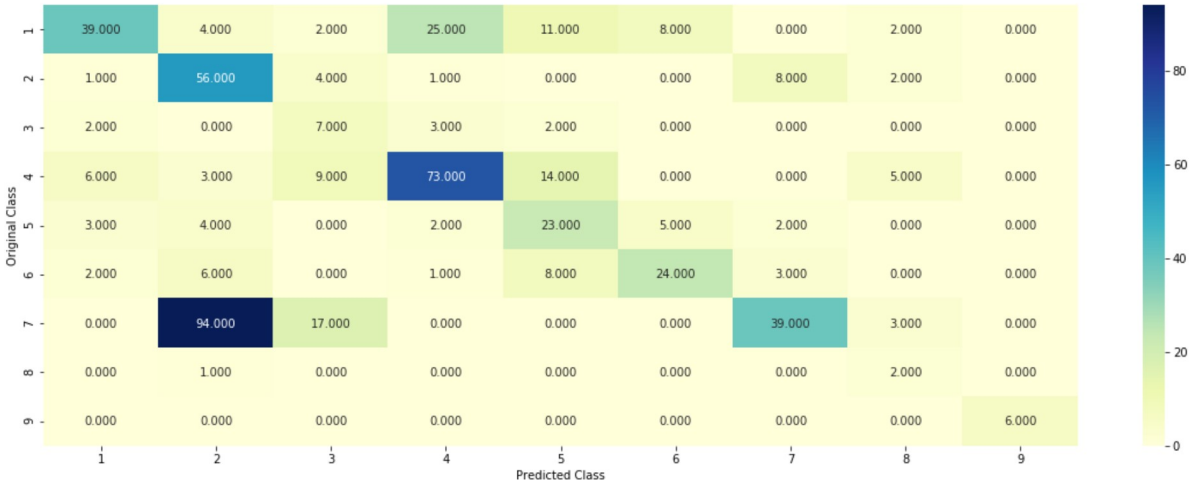
In [104]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


# 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]))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.043532359757118
for n_estimators = 10 and max depth =  3
Log Loss : 1.6848666872267464
for n_estimators = 10 and max depth =  5
Log Loss : 1.5485091390999148
for n_estimators = 10 and max depth =  10
Log Loss : 1.6730968655730925
for n_estimators = 50 and max depth =  2
Log Loss : 1.5955384063889686
for n_estimators = 50 and max depth =  3
Log Loss : 1.382443253784459
for n_estimators = 50 and max depth =  5
Log Loss : 1.3361780076298773
for n_estimators = 50 and max depth =  10
Log Loss : 1.6491291001162642
for n_estimators = 100 and max depth =  2
Log Loss : 1.5004405322042007
for n_estimators = 100 and max depth =  3
Log Loss : 1.4144312173928197
for n_estimators = 100 and max depth =  5
Log Loss : 1.3352069773071837
for n_estimators = 100 and max depth =  10
Log Loss : 1.6827442796052339
for n_estimators = 200 and max depth =  2
Log Loss : 1.5414021942833556
for n_estimators = 200 and max depth =  3
Log Loss : 1.4111093687068375
for n_estimators = 200 and max depth =  5
Log Loss : 1.3763166559684137
for n_estimators = 200 and max depth =  10
Log Loss : 1.7160104945993897
for n_estimators = 500 and max depth =  2
Log Loss : 1.6070539207264039
for n_estimators = 500 and max depth =  3
Log Loss : 1.4868328989923085
for n_estimators = 500 and max depth =  5
Log Loss : 1.3968685935259155
for n_estimators = 500 and max depth =  10
Log Loss : 1.7065511934705704
for n_estimators = 1000 and max depth =  2
Log Loss : 1.588452505337816
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4917688279496668
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3905271468718754
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6908263270254764
For values of best alpha =  100 The train log loss is: 0.05534318629572702
For values of best alpha =  100 The cross validation log loss is: 1.335206977307
1837
For values of best alpha =  100 The test log loss is: 1.3678092052971058
```

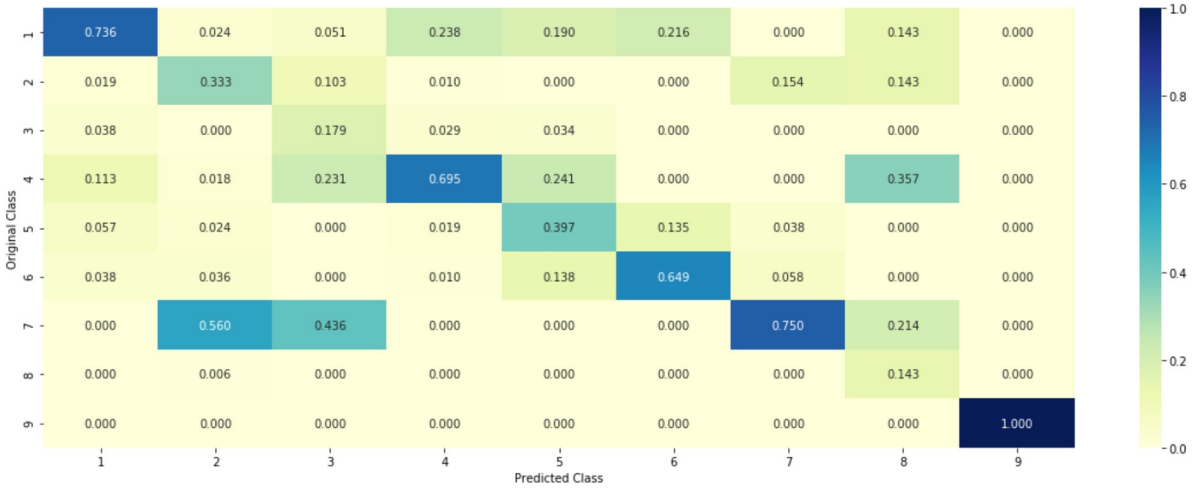
4.5.4. Testing model with best hyper parameters (Response Coding)

In [105]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

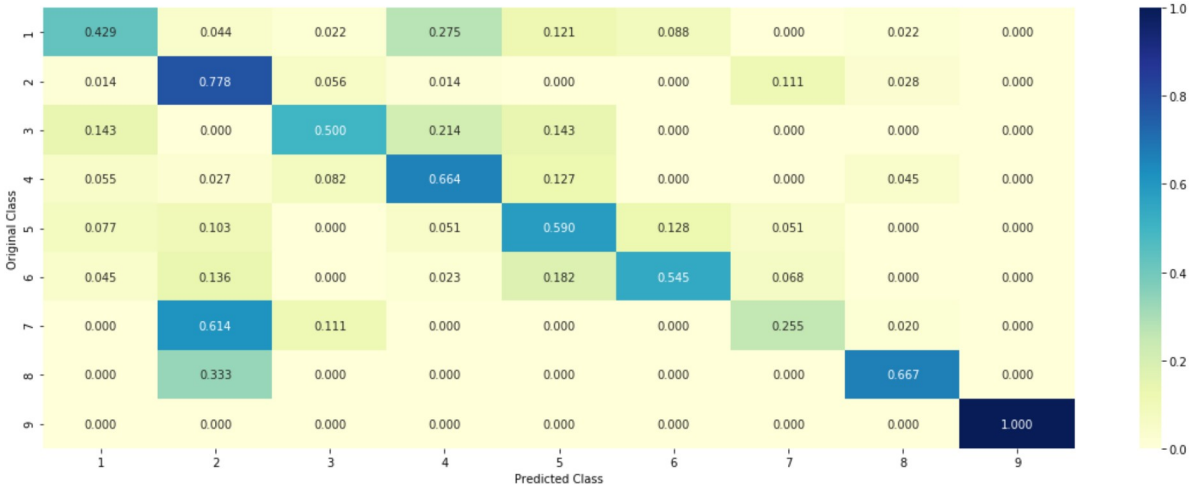
```
Log loss : 1.3352069773071837
Number of mis-classified points : 0.4943609022556391
-------------------- Confusion matrix --------------------
```




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




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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [106]:
```
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_resp
onseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2267 0.0248 0.0843 0.4519 0.051  0.0595 0.013
4 0.0282 0.0605]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [108]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gin
i', 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 = 5
no_feature = 1000
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_resp
onseCoding[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 : 9
Predicted Class Probabilities: [[0.02   0.0218 0.0443 0.12   0.0441 0.0222 0.013
7 0.056  0.6579]]
Actual Class : 8
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

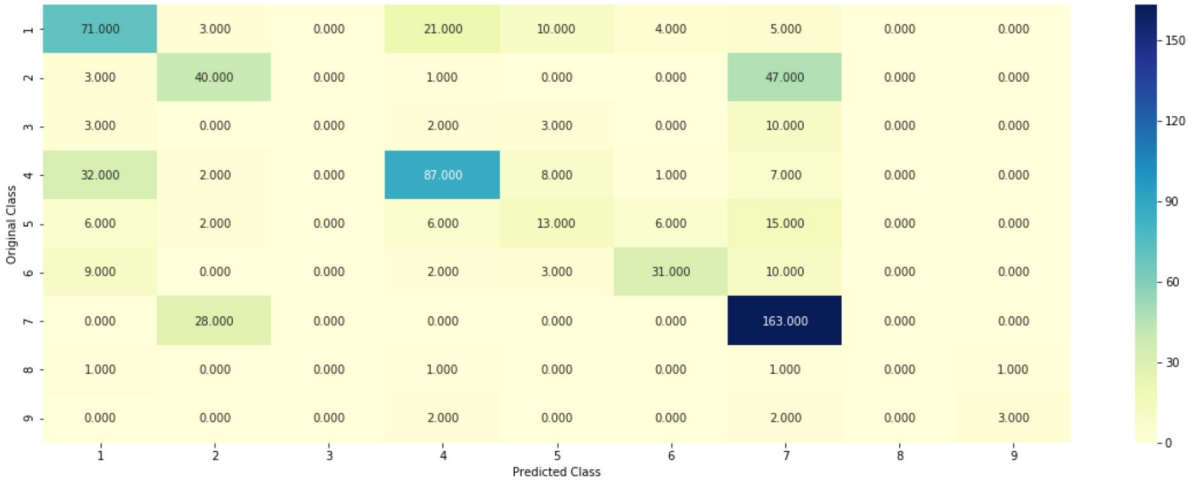
In [109]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
ted/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inter
cept=True, max_iter=None, 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 G
radient Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/less
ons/geometric-intuition-1/
#-------------------------------


# read more about support vector machines with linear kernals here http://scikit-l
earn.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, prob
ability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function
_shape='ovr', random_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/less
ons/mathematical-derivation-copy-8/
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit-l
earn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_d
epth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state
=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/less
```

```
Logistic Regression :  Log Loss: 1.06
Support vector machines : Log Loss: 1.84
Naive Bayes : Log Loss: 1.15
-------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.487
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.139
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.341
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.796
```

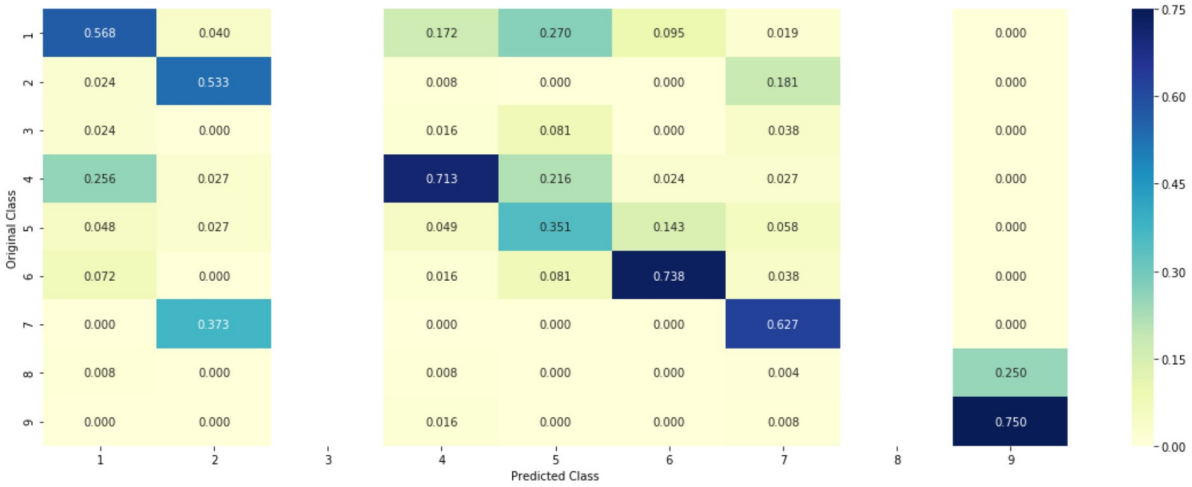
4.7.2 testing the model with the best hyper parameters

In [110]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classif
ier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

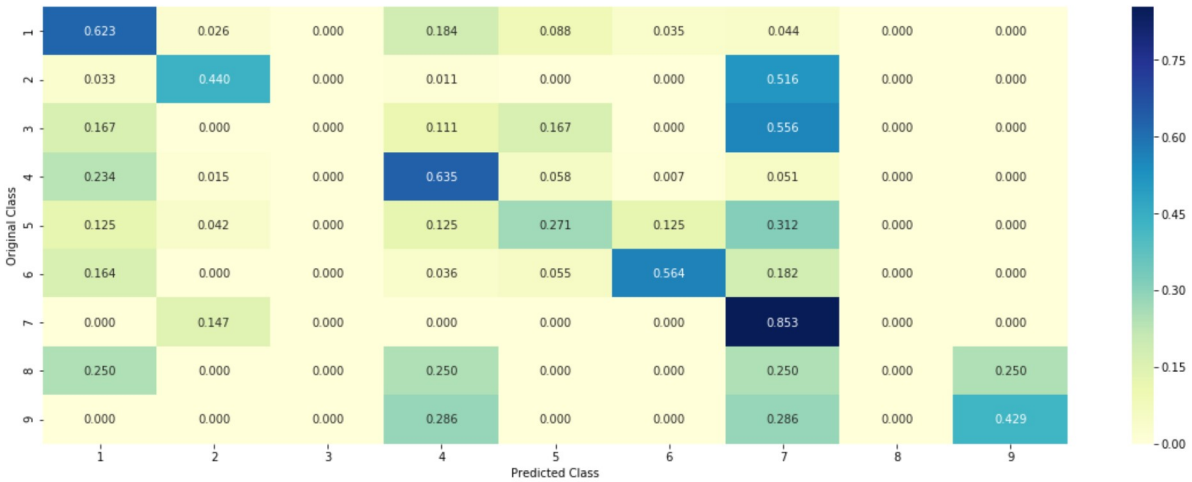
```
Log loss (train) on the stacking classifier : 0.5386754023282136
Log loss (CV) on the stacking classifier : 1.138717562146062
Log loss (test) on the stacking classifier : 1.1742087492677697
Number of missclassified point : 0.38646616541353385
-------------------- Confusion matrix --------------------
```




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



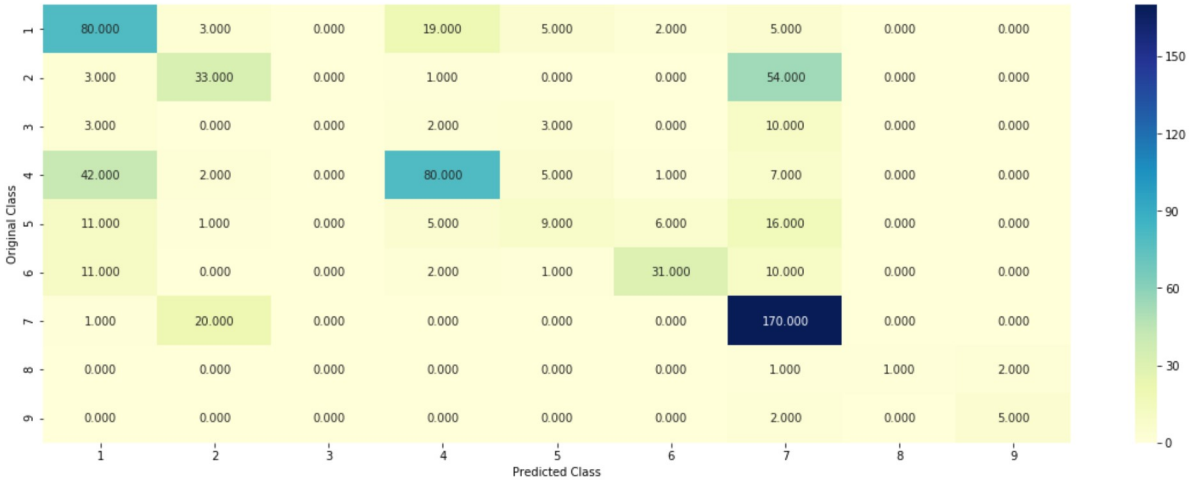

```
-------------------- Recall matrix (Row sum=1) --------------------
```



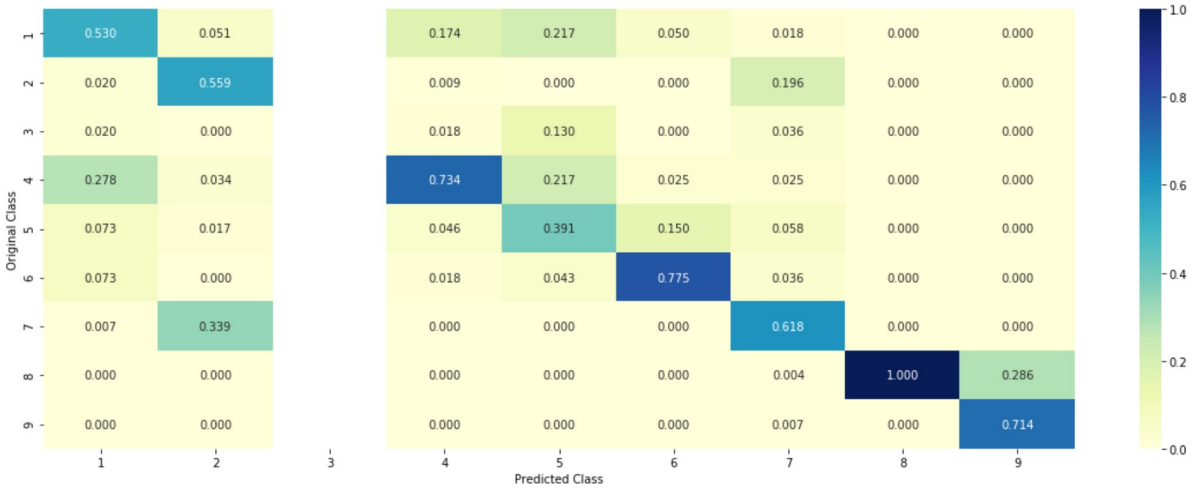
4.7.3 Maximum Voting classifier

In [111]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

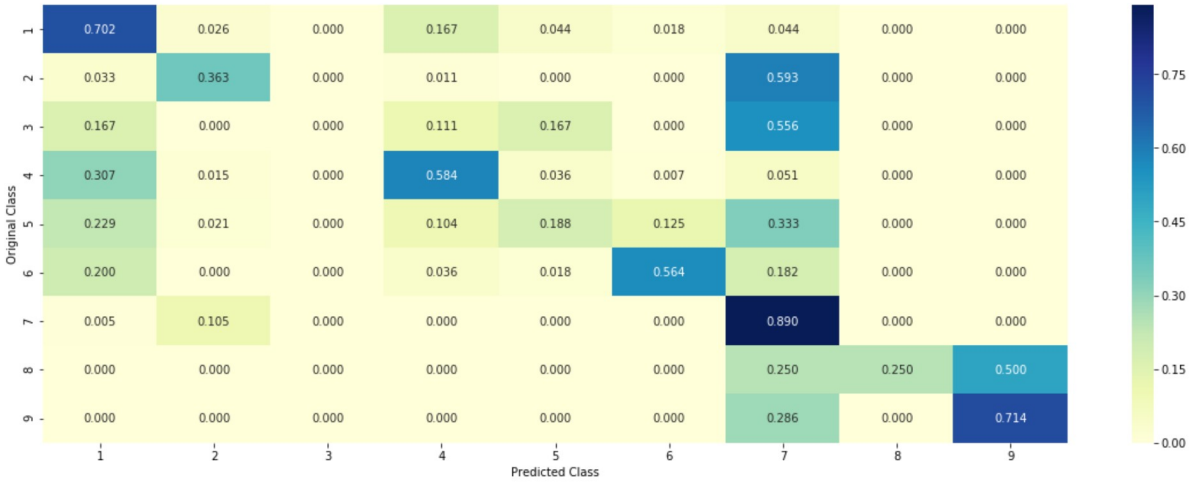
```
Log loss (train) on the VotingClassifier : 0.8329702627479129
Log loss (CV) on the VotingClassifier : 1.1887678593349613
Log loss (test) on the VotingClassifier : 1.2061284826287209
Number of missclassified point : 0.3849624060150376
--------------------- Confusion matrix ----------
```




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




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



5. Assignments

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

Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [18]:
```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variation'].va
lues;cv_variation=cv_df['Variation'].values
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv_df['
Gene'].values
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv_df['
TEXT'].values

from sklearn.feature_extraction.text import CountVectorizer
encode=CountVectorizer(ngram_range=(1, 2))
train_variation=encode.fit_transform(train_variation);test_variation=encode.transfo
rm(test_variation);cv_variation=encode.transform(cv_variation)
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);c
v_gene=encode.transform(cv_gene)
#train_text=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);
cv_gene=encode.transform(cv_gene)
encode=CountVectorizer(min_df=10,ngram_range=(1,2))
train_variation=normalize(train_variation,axis=0);test_variation=normalize(test_var
iation,axis=0);cv_variation=normalize(cv_variation,axis=0);
train_gene=normalize(train_gene,axis=0);test_gene=normalize(test_gene,axis=0);cv_ge
ne=normalize(cv_gene,axis=0);
print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_text);c
v_text=encode.transform(cv_text)
train_text=normalize(train_text,axis=0);test_text=normalize(test_text,axis=0);cv_te
xt=normalize(cv_text,axis=0)
print(train_text.shape)
```

```
(2124, 232)
  (0, 133)      0.20851441405707477
(2124, 2068)
  (0, 1897)     1.0
(2124, 236306)
```

In [19]:
```
from scipy.sparse import hstack
print(train_variation.shape,train_gene.shape,train_text.shape)
train_data=hstack([train_variation,train_gene,train_text]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text]).tocsr()
test_data=hstack([test_variation,test_gene,test_text]).tocsr()
print(train_data.shape,cv_data.shape,test_data.shape)
train_y = np.array(list(y_train['Class']))
test_y = np.array(list(y_test['Class']))
cv_y = np.array(list(y_cv['Class']))
```

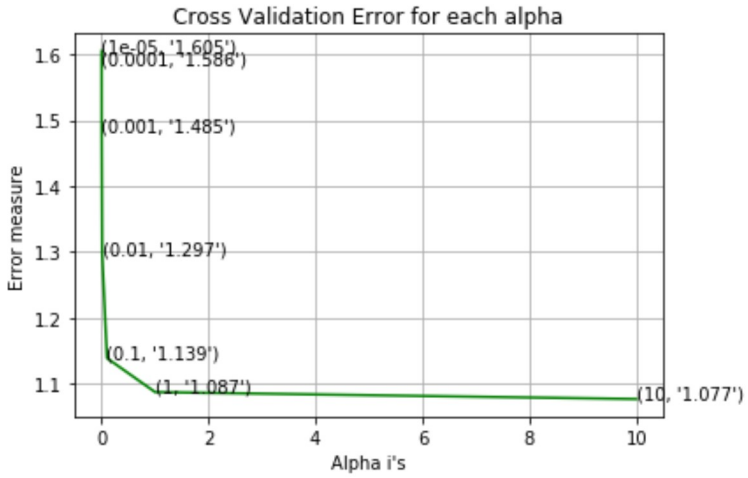
```
(2124, 2068) (2124, 232) (2124, 236306)
(2124, 238606) (532, 238606) (665, 238606)
```

In [20]:
```
cv_scores=[]
alpha=[10 ** x for x in range(-5, 2)]
for c in alpha:
    print("for alpha =", c)
    lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_jobs=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_op=clf.predict_proba(cv_data)
    print(c,log_loss(y_cv, cv_op))
    cv_scores.append(log_loss(y_cv, cv_op))
```

```
for alpha = 1e-05
1e-05 1.6052006049440848
for alpha = 0.0001
0.0001 1.5862844259217301
for alpha = 0.001
0.001 1.4851353019665983
for alpha = 0.01
0.01 1.296507606095338
for alpha = 0.1
0.1 1.1389029301249398
for alpha = 1
1 1.0873807948800525
for alpha = 10
10 1.0768641779703876
```

In [21]:
```
print(alpha,cv_scores)
print(len(alpha),len(cv_scores))
print(type(cv_scores))
fig, ax = plt.subplots()
ax.plot(alpha, cv_scores,c='g')
for i, txt in enumerate(np.round(cv_scores,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_scores[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10] [1.6052006049440848, 1.5862844259217301
, 1.4851353019665983, 1.296507606095338, 1.1389029301249398, 1.0873807948800525,
1.0768641779703876]
7 7
<class 'list'>
```

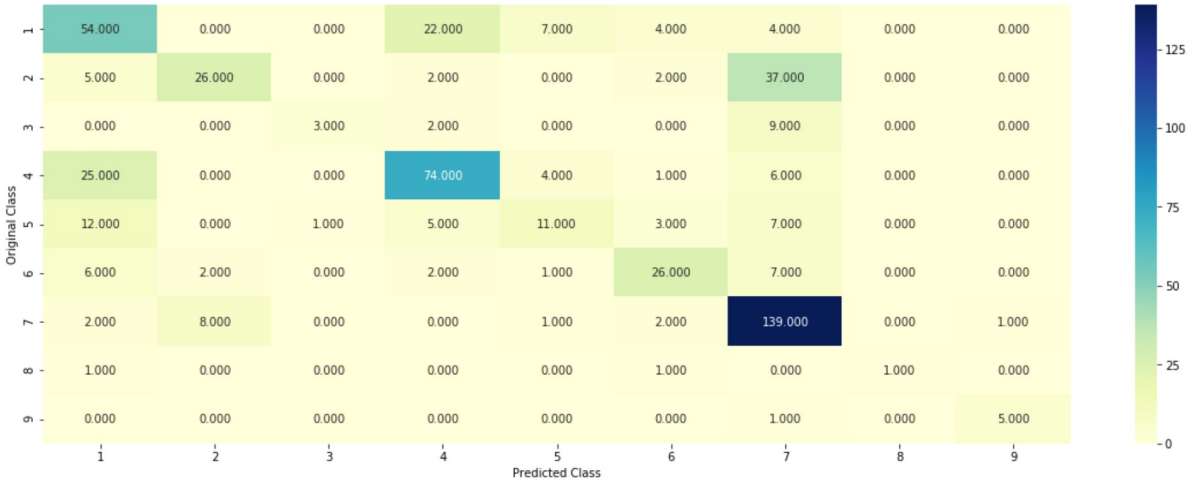


In [22]:
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
print("Log Loss value for the test data is(10) ",log_loss(y_test, test_op))
```

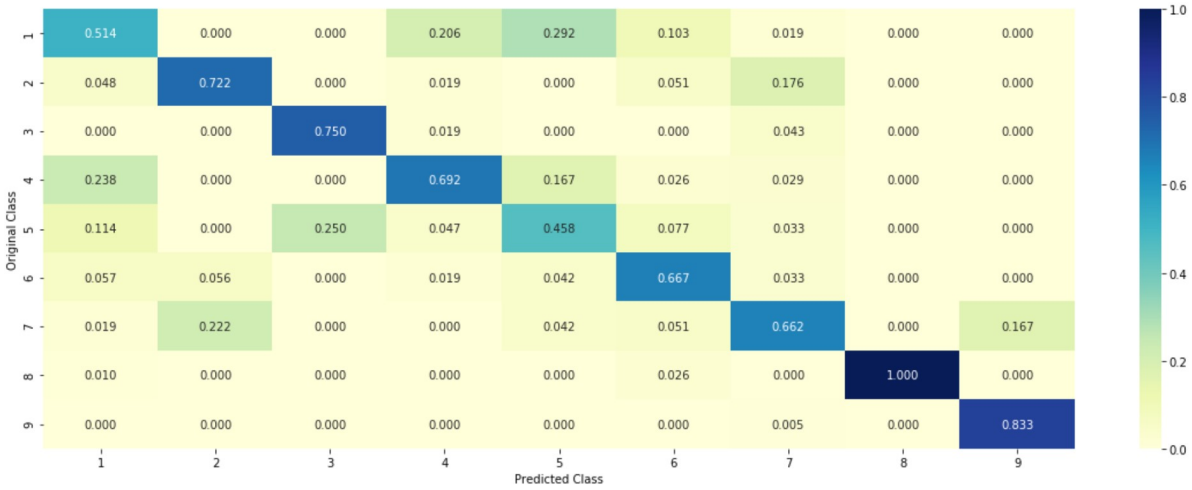
```
Log Loss value for the test data is(10)  1.1063249920724818
```

In [30]:
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
predict_and_plot_confusion_matrix(train_data, y_train, cv_data,y_cv, clf)
```

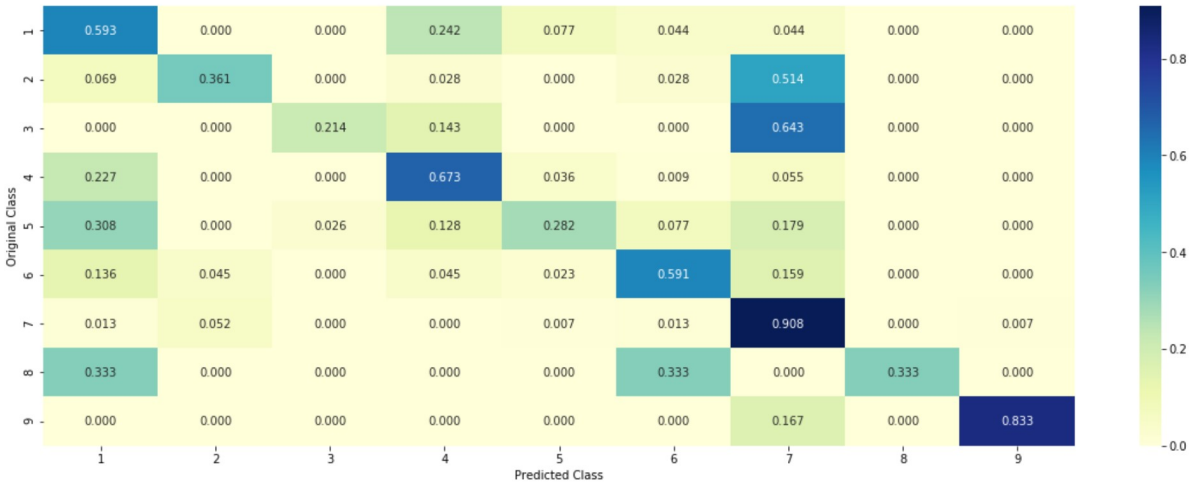
```
Log loss : 1.1025061826224287
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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




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



feature engineering techniques to reduce log loss

In [25]:
```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation'
]
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true
, test_size=0.2)

x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train,
test_size=0.2)
```

In [26]:
```
def get_gv_fea_dict(alpha, feature, df):
    value_count = x_train[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    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])
    return gv_fea
```

In [27]:
```
alpha = 1

# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_train)
)

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_test))

# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_cv))
```

In [28]:
```
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

Variation Feature

In [29]:
```
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation"
, x_train))

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",
x_test))

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x
_cv))
```

In [30]:
```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['
Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variat
ion'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'
])
```

Text Feature

In [31]:
```
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


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.ge
t(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 [32]:
```
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['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 features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 126933
```

In [33]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['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(x_train)


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 [34]:
```
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# 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_fe
ature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_
responseCoding.sum(axis=1)).T
```

In [35]:
```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['TEXT'])
```

Features after feature engineering

In [36]:
```
gene_variation = []

for gene in data['Gene'].values:
    gene_variation.append(gene)

for variation in data['Variation'].values:
    gene_variation.append(variation)
```

In [37]:
```
tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
cv_text = tfidfVectorizer.transform(x_cv['TEXT'])
```

Stack above three features

In [38]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variati
on_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_featur
e_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_onehotCodin
g)).tocsr()
train_y = np.array(list(x_train['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(x_test['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(x_cv['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_var
iation_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_featu
re_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_respon
seCoding))
```

In [39]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_oneh
otCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehot
Coding.shape)
print("(number of data points * number of features) in cross validation data =", cv
_x_onehotCoding.shape)
```

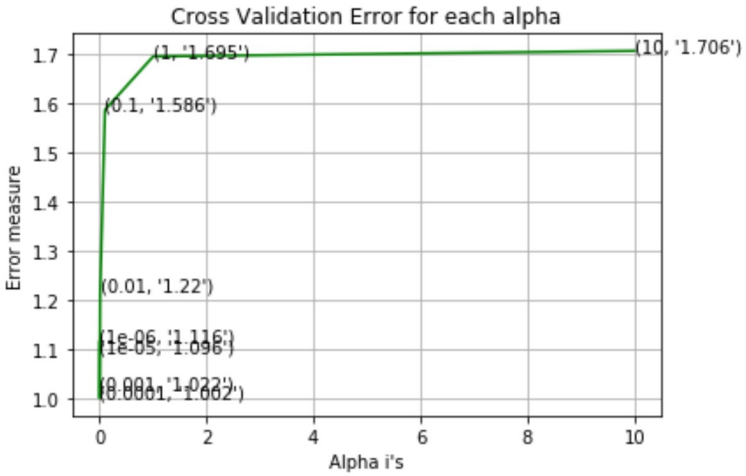
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 130132)
(number of data points * number of features) in test data =  (665, 130132)
(number of data points * number of features) in cross validation data = (532, 13
0132)
```

In [40]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_resp
onseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_respon
seCoding.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
)
```

In [41]:
```
alpha = [10 ** x for x in range(-6, 2)]
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=41)
    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_, ep
s=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',)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

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

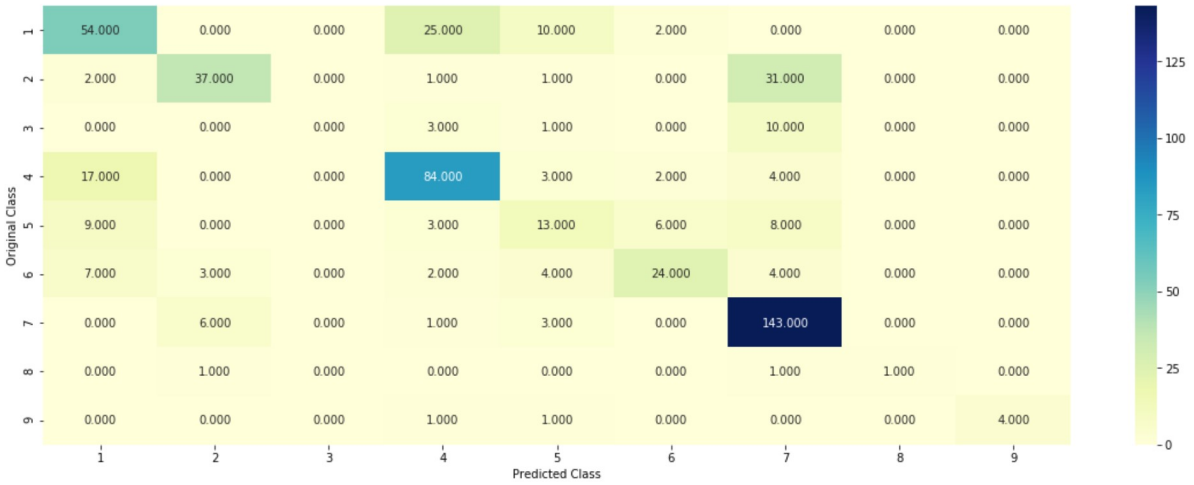
```
for alpha = 1e-06
Log Loss : 1.1158948500930739
for alpha = 1e-05
Log Loss : 1.095833448152242
for alpha = 0.0001
Log Loss : 1.0020072168975935
for alpha = 0.001
Log Loss : 1.0223136748469
for alpha = 0.01
Log Loss : 1.2195261488131441
for alpha = 0.1
Log Loss : 1.58629762046778
for alpha = 1
Log Loss : 1.6945407124528586
for alpha = 10
Log Loss : 1.7059613023913913
```



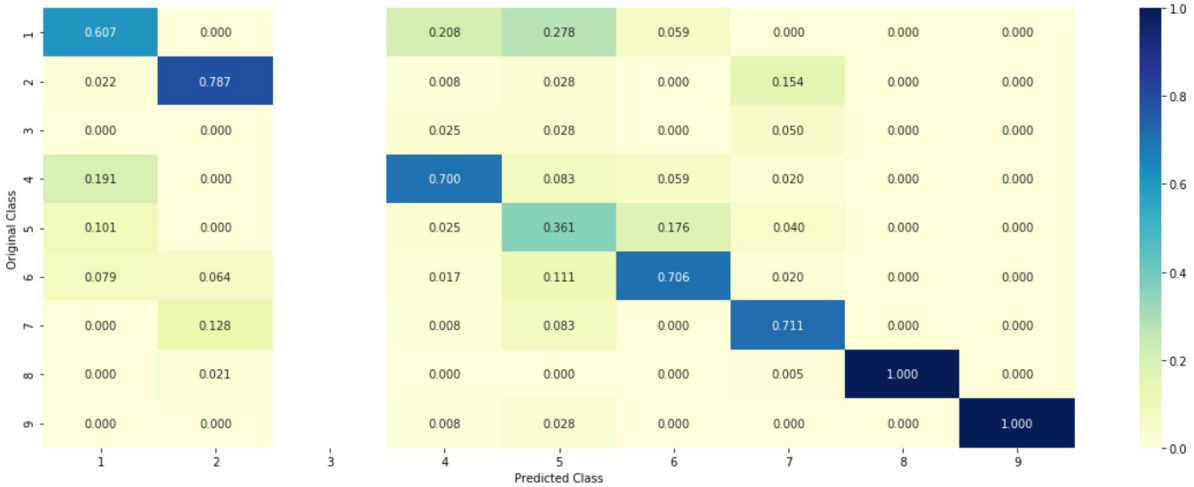

```
For values of best alpha =  0.0001 The train log loss is: 0.4284816894923472
For values of best alpha =  0.0001 The cross validation log loss is: 0.995426043
0011568
For values of best alpha =  0.0001 The test log loss is: 1.0420484387516074
```

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

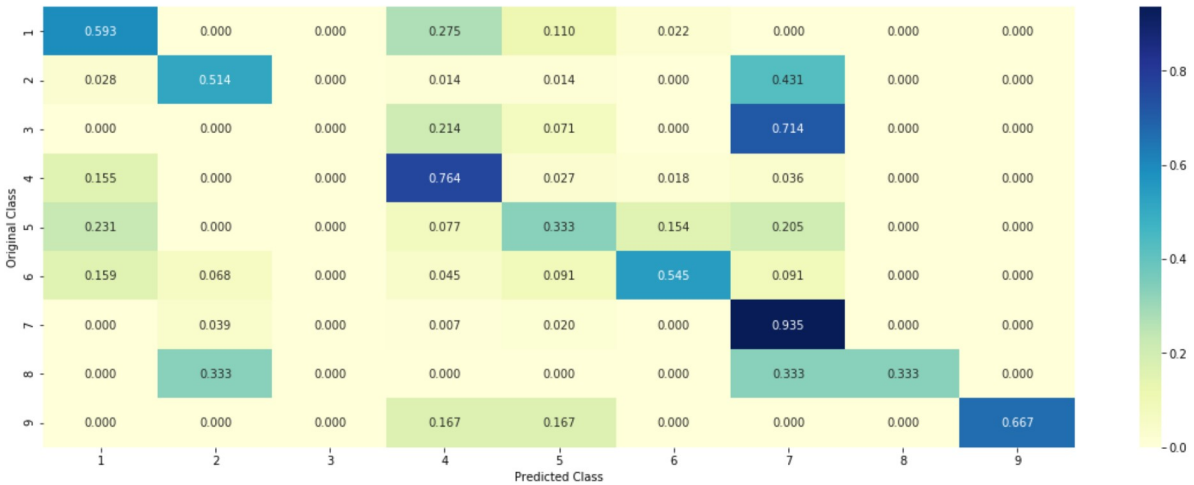
```
Log loss : 0.9976654523552164
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```




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




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



After some feature engineering we manage to decrease the log loss below < 1. We can adopt more feature enginnering methods and reduce the log loss furhermore.

In [45]:
```
from prettytable import PrettyTable

# Names of models
model=['Naive Bayes  ','KNN','Logistic Regression With Class balancing  ','Logistic
Regression Without Class balancing','Linear SVM  ','Random Forest Classifier With O
ne hot Encoding','Random Forest Classifier With Response Coding','Stack Models:LR+N
B+SVM','Maximum Voting classifier','CountVectorizer Features, including both unigra
ms and bigrams','after feature engineering']



train =[0.52,0.49,0.44,0.43,0.47, 0.87,0.05,0.53,0.83,1.102,0.42]
cv=[1.15,1.03,1.04,1.06,1.07,1.19,1.33,1.13,1.18,1.07,0.995]
test = [1.20,1.03,1.01,1.02,1.05,1.21,1.36,1.17,1.20,1.06,0.997]
mp=[38,36,34,35,34,39,49,38,38,36,32]
numbering=[1,2,3,4,5,6,7,8,9,10,11]
# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("model",model)
ptable.add_column("train",train)
ptable.add_column("cv",cv)
ptable.add_column("test",test)
ptable.add_column("% Misclassified Points",mp)

# Printing the Table
print(ptable)
```

```
+-------+-------------------------------------------------------------------+-------
+-------+-------+-------------------------+
| S.NO. |                                model                              | train
|   cv  |  test | % Misclassified Points  |
+-------+-------+-------------------------+-------------------------------------------
+-------+-------+-------------------------+
|   1   |                           Naive Bayes                           |  0.52
|  1.15 |  1.2  |           38            |
|   2   |                                KNN                              |  0.49
|  1.03 |  1.03 |           36            |
|   3   |           Logistic Regression With Class balancing              |  0.44
|  1.04 |  1.01 |           34            |
|   4   |          Logistic Regression Without Class balancing            |  0.43
|  1.06 |  1.02 |           35            |
|   5   |                          Linear SVM                             |  0.47
|  1.07 |  1.05 |           34            |
|   6   |         Random Forest Classifier With One hot Encoding          |  0.87
|  1.19 |  1.21 |           39            |
|   7   |         Random Forest Classifier With Response Coding           |  0.05
|  1.33 |  1.36 |           49            |
|   8   |                        Stack Models:LR+NB+SVM                   |  0.53
|  1.13 |  1.17 |           38            |
|   9   |                    Maximum Voting classifier                    |  0.83
|  1.18 |  1.2  |           38            |
|   10  | CountVectorizer Features, including both unigrams and bigrams | 1.102
|  1.07 |  1.06 |           36            |
|   11  |                    after feature engineering                    |  0.42
| 0.995 | 0.997 |           32            |
+-------+-------+-------------------------+-------------------------------------------
+-------+-------+-------------------------+
```

In [ ]:

In [ ]: