

Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/> (<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> (<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=qxXRKVompl8> (<https://www.youtube.com/watch?v=qxXRKVompl8>)

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 contains 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 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 Learning 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 probabilities => 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

-----**TASK1**-----

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 collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE

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

from sklearn.datasets import make_classification
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')
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 separator in this file
data_text = pd.read_csv("training_text", sep="\|\|", engine="python", names=["ID", "TEXT"], skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [61]:

```
# 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 : 183.7006506518798 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 lineage...

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']==1109]
```

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [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 variable 'y'
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.15)
# split the train data into train and cross validation by maintaining same distribution of 'y'
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.15)
```

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

In [11]:

```
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 [12]:

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

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

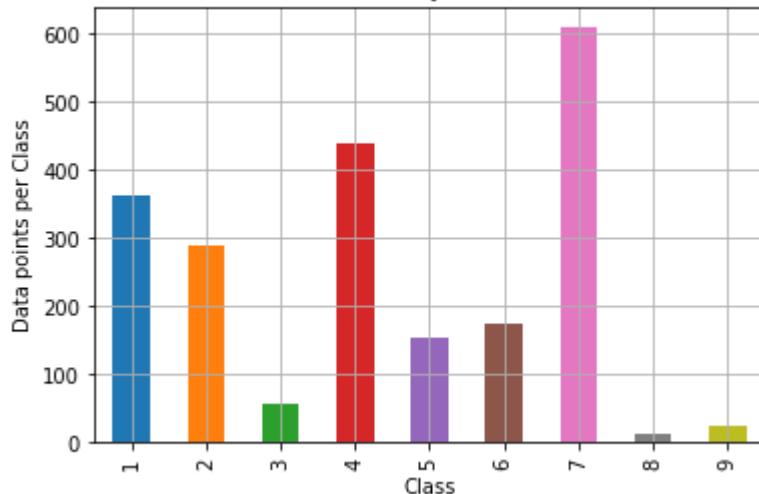
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i],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

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

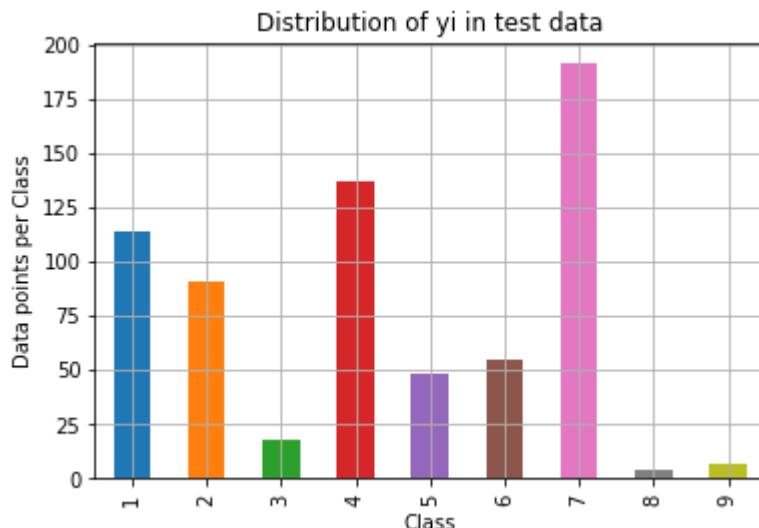
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],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

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

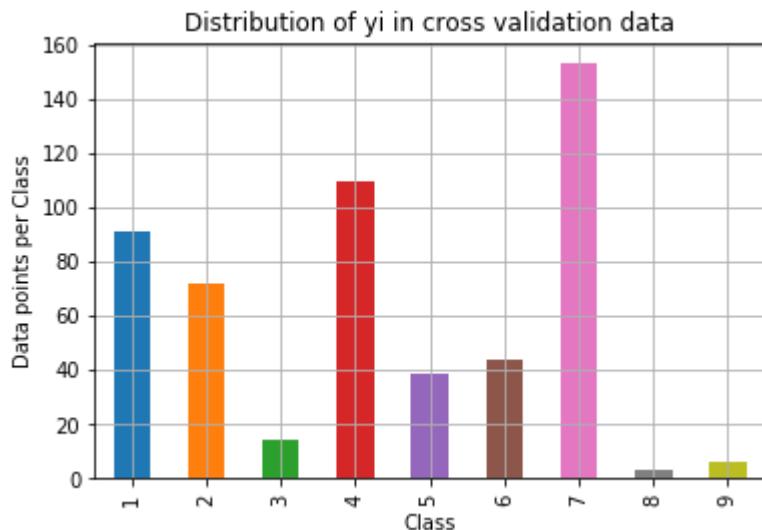
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],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

Distribution of y_i in train data

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 probabilities randomly such that they sum to 1.

In [62]:

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

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

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

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

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row

    # C = [[1, 2],
    #       [3, 4]]
    # C.sum(axis = 0) axis=0 corresonds to columns and axis=1 corresponds to rows in two a
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                       [3/4, 4/6]]

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

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

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

In [212]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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))) * 9)
print("Log loss on Cross Validation Data using Random Model", log_loss(y_cv, cv_predicted_y,

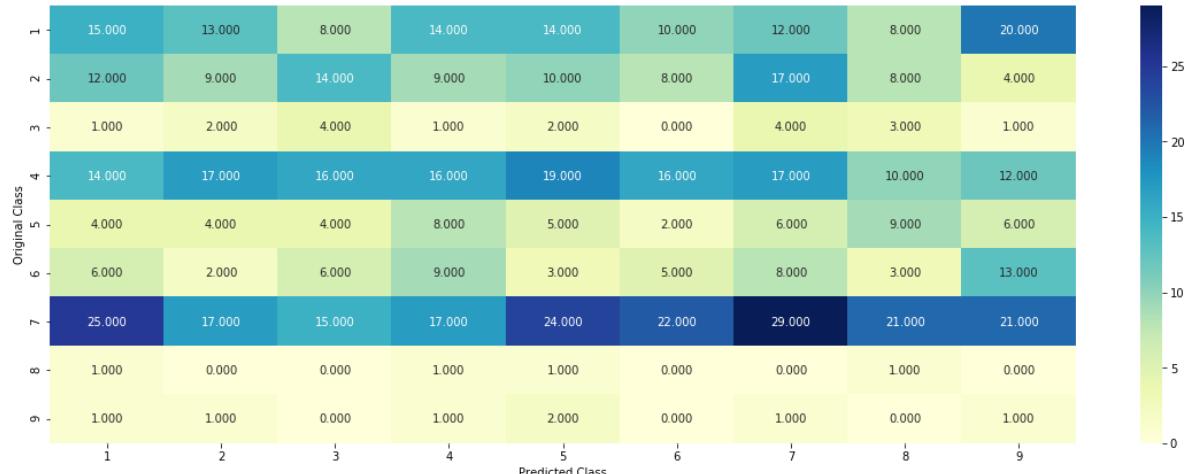
# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len, 9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1, 9)
    test_predicted_y[i] = ((rand_probs / sum(sum(rand_probs))) * 9)
print("Log loss on Test Data using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

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

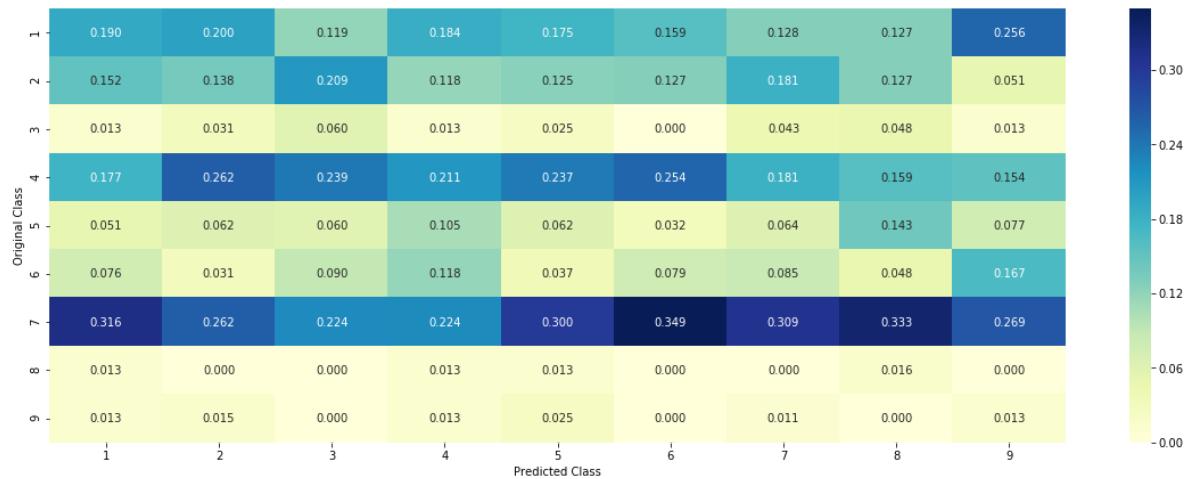
Log loss on Cross Validation Data using Random Model 2.4927691664034706

Log loss on Test Data using Random Model 2.4454599772229146

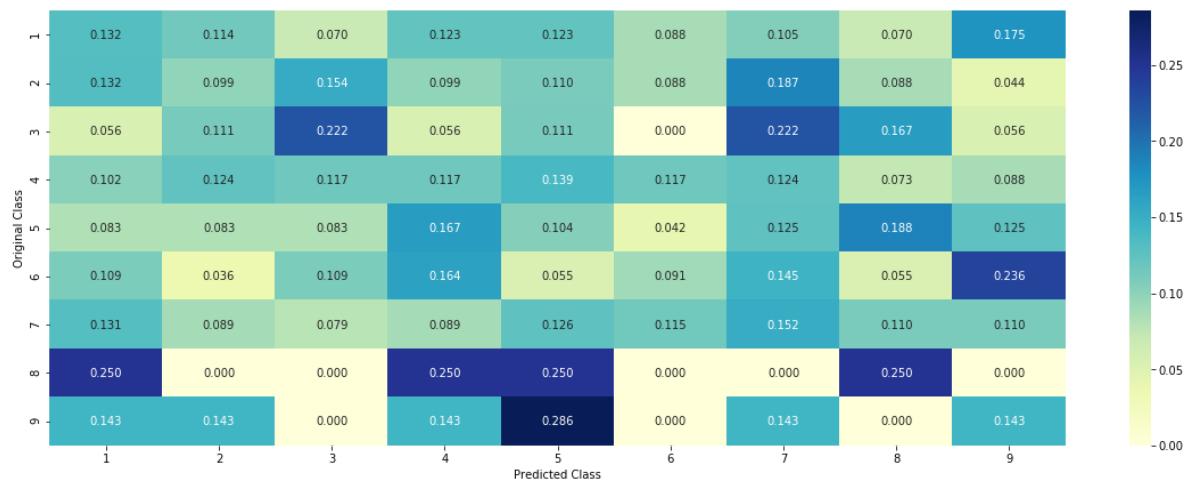
----- Confusion matrix -----



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



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



3.3 Univariate Analysis

In [13]:

```

# 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 occurrences of given feature in train data do
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha)
# gv_dict is like a look up table, for every gene it stores a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# -----
# get_gv_fea_dict: Get Gene variation 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 occurred in whole
    for i, denominator in value_count.items():
        # vec will contain ( $p(y_i=1|G_i)$ ) probability of gene/variation belongs to particular
        # vec is 9 dimensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            # ID Gene Variation Class

```

```

# 2470 2470 BRCA1           S1715C      1
# 2486 2486 BRCA1           S1841R      1
# 2614 2614 BRCA1           M1R         1
# 2432 2432 BRCA1           L1657P      1
# 2567 2567 BRCA1           T1685A      1
# 2583 2583 BRCA1           E1660G      1
# 2634 2634 BRCA1           W1718L      1
# cls_cnt.shape[0] will return the number of rows

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.2007575757575757, 0.037878787878788, 0.0681818181818177, 0.1363
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704
    # 'EGFR': [0.0568181818181816, 0.21590909090909091, 0.0625, 0.0681818181818177
    # 'BRCA2': [0.1333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333334, 0.0733
    # ...
    #     ]
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

```

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

- $(\text{numerator} + 10 * \text{alpha}) / (\text{denominator} + 90 * \text{alpha})$

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 [14]:

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

Number of Unique Genes : 242

```
BRCA1      168
TP53       110
EGFR        96
BRCA2       83
PTEN        77
BRAF        65
KIT          63
ERBB2        47
ALK          43
PDGFRA      41
Name: Gene, dtype: int64
```

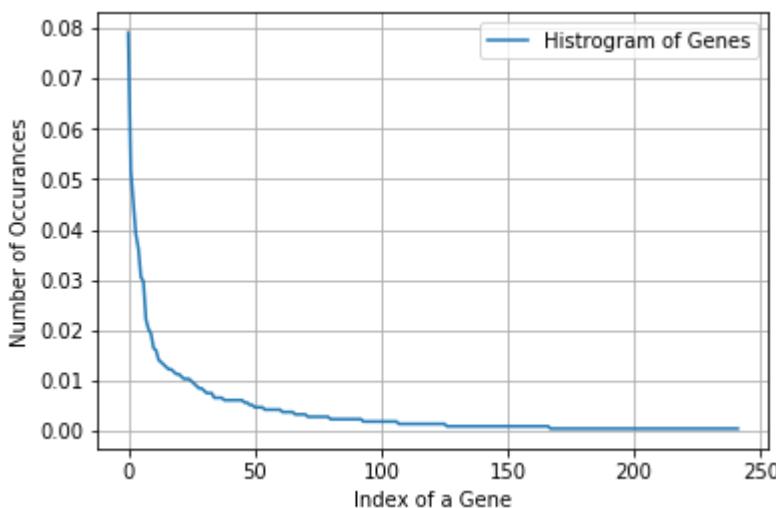
In [15]:

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

Ans: There are 242 different categories of genes in the train data, and they are distributed as follows

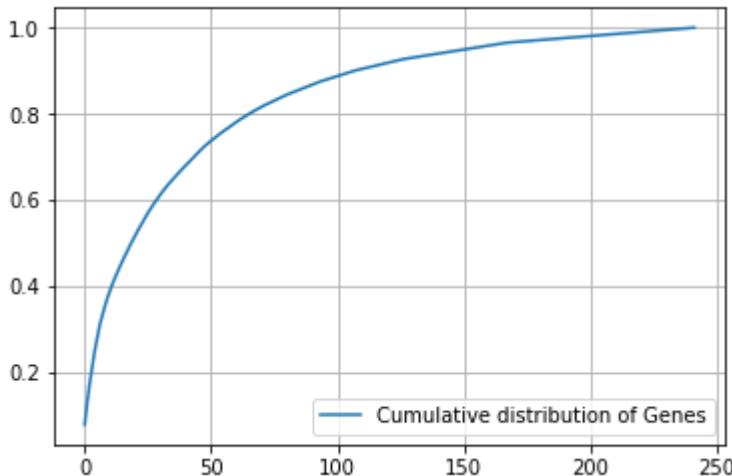
In [16]:

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurrences')
plt.legend()
plt.grid()
plt.show()
```



In [17]:

```
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/> (<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 [18]:

```
#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 [19]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method.")
```

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

In [22]:

```
# Tfifd 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 [23]:

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

Out[23]:

```
377      TP53
2107     B2M
1726     APC
2301     JAK1
45       PTPRT
Name: Gene, dtype: object
```

In [24]:

```
gene_vectorizer.get_feature_names()
```

Out[24]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx']
```

In [180]:

```
print("train_gene_feature_Tfidf is converted feature using Tfifd encoding method. The shape
```

```
train_gene_feature_Tfidf is converted feature using Tfifd encoding method. The shape of gene feature: (2124, 235)
```

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 (Tfifd encoded) to predict y_i .

In [224]:

```

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

# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/skLea
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# 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.

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

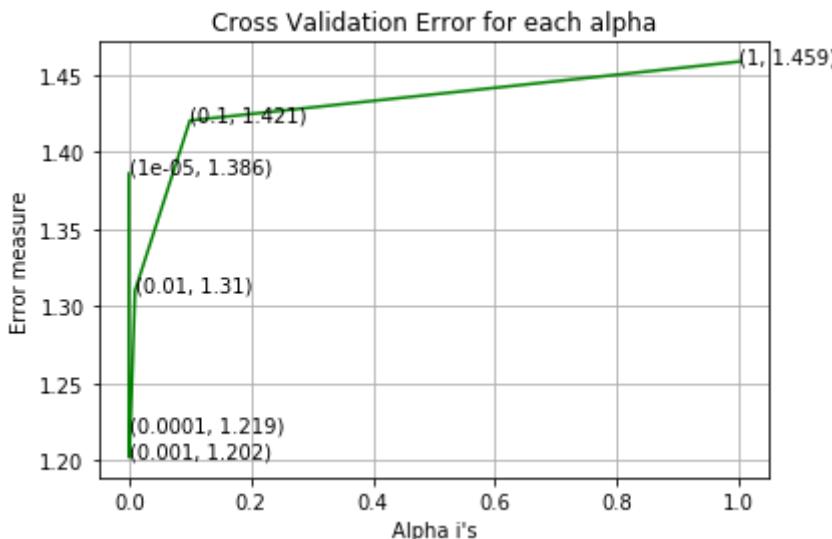
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_

```

For values of alpha = 1e-05 The log loss is: 1.3861598177029062
 For values of alpha = 0.0001 The log loss is: 1.2190580013223984
 For values of alpha = 0.001 The log loss is: 1.2015839114085385
 For values of alpha = 0.01 The log loss is: 1.3100472458561074

For values of alpha = 0.1 The log loss is: 1.4205837019572183
 For values of alpha = 1 The log loss is: 1.4588497615873561



For values of best alpha = 0.001 The train log loss is: 1.093262843281969
 For values of best alpha = 0.001 The cross validation log loss is: 1.2015839114085385
 For values of best alpha = 0.001 The test log loss is: 1.1922270287097307

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 [25]:

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

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

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

Q6. How many data points in Test and CV datasets are covered by the 236 genes in train dataset?

Ans

1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 520 out of 532 : 97.74436090225564

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 [23]:

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

Number of Unique Variations : 1938

Truncating_Mutations 55

Deletion 48

Amplification 46

Fusions 18

G12V 4

Overexpression 4

Q61H 3

Q61R 3

M1R 2

P34R 2

Name: Variation, dtype: int64

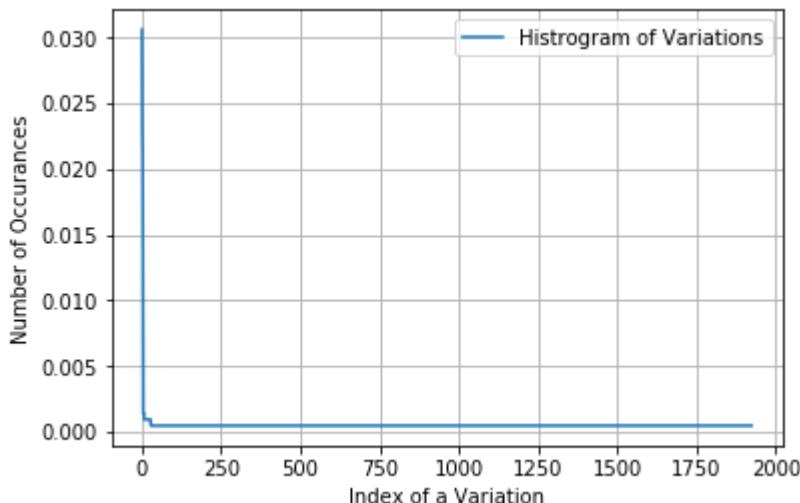
In [24]:

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

Ans: There are 1938 different categories of variations in the train data, and they are distributed as follows

In [179]:

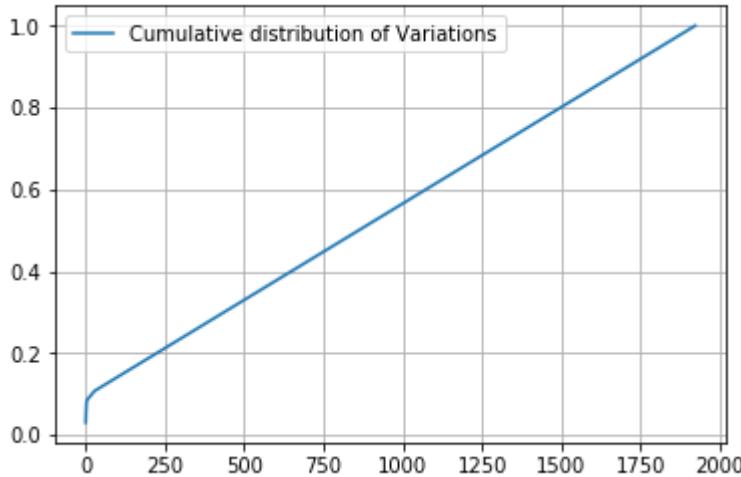
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurrences')
plt.legend()
plt.grid()
plt.show()
```



In [180]:

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

```
[0.03060264 0.05367232 0.07485876 ... 0.99905838 0.99952919 1. ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/> (<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 [25]:

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

In [26]:

```
print("train_variation_feature_responseCoding is a converted feature using the response cod
```

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

In [29]:

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

In [30]:

```
print("train_variation_feature_Tfidf is converted feature using the TfidfVectorizer encoding")
```

```
train_variation_feature_Tfidf is converted feature using the TfidfVectorizer
encoding method. The shape of Variation feature: (2124, 1949)
```

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

Let's build a model just like the earlier!

In [234]:

```

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

# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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

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

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

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

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_
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:"
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_

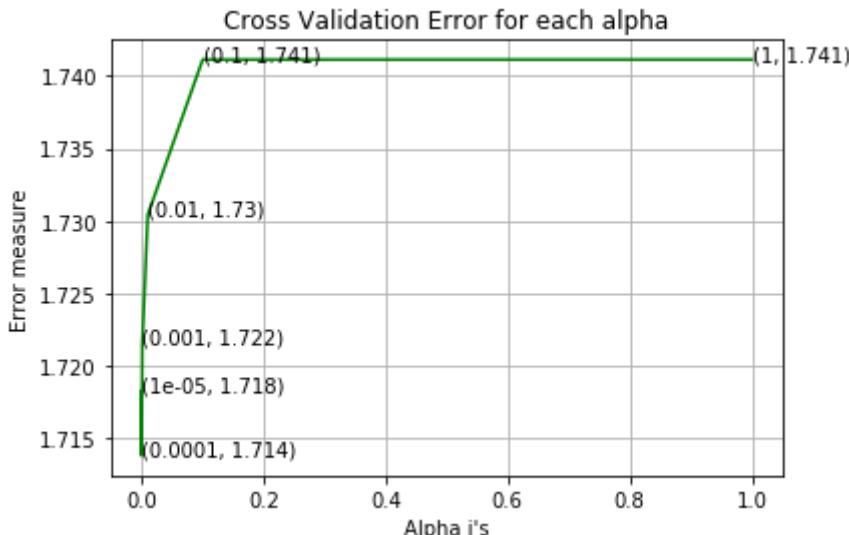
```

For values of alpha = 1e-05 The log loss is: 1.7182688218124458
 For values of alpha = 0.0001 The log loss is: 1.7138058026441398
 For values of alpha = 0.001 The log loss is: 1.7215133394904796

For values of alpha = 0.01 The log loss is: 1.7304203308226154

For values of alpha = 0.1 The log loss is: 1.7411277745200926

For values of alpha = 1 The log loss is: 1.7411264446901122



For values of best alpha = 0.0001 The train log loss is: 0.7682324124041652

For values of best alpha = 0.0001 The cross validation log loss is: 1.7138058026441398

For values of best alpha = 0.0001 The test log loss is: 1.6991731517000892

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 [235]:

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

Q12. How many data points are covered by total 1924 genes in test and cross validation data sets?

Ans

1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 51 out of 532 : 9.586466165413533

3.2.3 Univariate Analysis on Text Feature

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

In [33]:

```
# 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 [34]:

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

In [34]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3)
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) array
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 occurred
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 : 53692

In [35]:

```

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

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

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

```

In [40]:

```

#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 [41]:

```

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

```

In [39]:

```

# 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 [40]:

```

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

```

In [41]:

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

```
Counter({0.0063870312851483819: 313, 0.022410291025923196: 312, 0.04729885  
6792744419: 230, 0.075839307972481548: 198, 0.014155561320837295: 181, 0.3  
0514031900622129: 152, 0.26286096469061093: 128, 0.060667395731548585: 12  
6, 0.01671417492882591: 117, 0.02325799695515816: 114, 0.06511664849861127  
6: 108, 0.060901990336415919: 106, 0.010727453087097031: 104, 0.0448810419  
79351277: 103, 0.081319145320628325: 99, 0.013562232519774856: 97, 0.03749  
6273535580882: 94, 0.11140356129360053: 93, 0.036304734276462518: 91, 0.02  
5872560512279555: 89, 0.012977841232013333: 86, 0.030743961773341853: 84,  
0.016040418183782549: 83, 0.020524332817345364: 82, 0.032458068368984828:  
81, 0.037414235747939936: 79, 0.030778199051910152: 71, 0.0243071880002198  
63: 69, 0.054567606632715218: 68, 0.019862572241641483: 65, 0.030021437722  
506397: 64, 0.011120483751958066: 64, 0.05989166423658239: 63, 0.036630376  
917486988: 63, 0.015840177433098904: 63, 0.014629973258167662: 62, 0.01207  
617590715993: 61, 0.011353244101638306: 61, 0.055604483255333265: 60, 0.01  
5050562621356461: 60, 0.11624445037125607: 58, 0.010764407387156774: 58,  
0.053492156760679067: 57, 0.03116336935149764: 57, 0.11180359202552854: 5  
6, 0.011596914643726468: 56, 0.023744468832284114: 55, 0.00975755294580892  
69: 54, 0.0181553605475426: 53, 0.19619398315211989: 51, 0.038349307208166  
566: 51, 0.019836556581524542: 51, 0.018691081907986889: 51, 0.05281505923  
5051000: 50, 0.0072010116200005001: 50, 0.015441000046007007: 50, 0.0120500
```

In [245]:

```
# Train a Logistic regression+Calibration model using text features which are on-hot encode
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,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

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

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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
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_cv, predict_y))

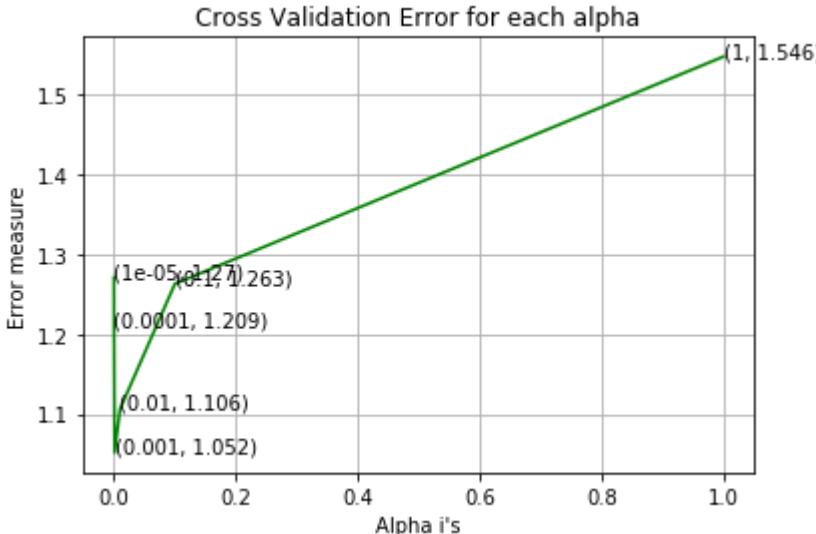


For values of alpha = 1e-05 The log loss is: 1.2704124625936264
For values of alpha = 0.0001 The log loss is: 1.209185614605813
For values of alpha = 0.001 The log loss is: 1.0518444423680169

For values of alpha = 0.01 The log loss is: 1.106391878843891

For values of alpha = 0.1 The log loss is: 1.2626012805699605

For values of alpha = 1 The log loss is: 1.5461491038587245



For values of best alpha = 0.001 The train log loss is: 0.6522870473446675

For values of best alpha = 0.001 The cross validation log loss is: 1.0518444423680169

For values of best alpha = 0.001 The test log loss is: 1.1507115914182782

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

Ans. Yes, it seems like!

In [42]:

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

    df_textfea_counts = df_textfea.sum(axis=0).A1
    df_textfea_dict = dict(zip(list(df_text_features),df_textfea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [247]:

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

95.344 % of word of test data appeared in train data
98.058 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [56]:

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

In [57]:

```
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 [58]:

```
# 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)

    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,ye
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]" .format(wc
        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,ye

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

Stacking the three types of features

In [125]:

```
# 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))
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding))
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 [126]:

```
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 = (1359, 2469)
(number of data points * number of features) in test data = (425, 2469)
(number of data points * number of features) in cross validation data = (34
0, 2469)
```

In [127]:

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

Response encoding features :

```
(number of data points * number of features) in train data = (1359, 27)
(number of data points * number of features) in test data = (425, 27)
(number of data points * number of features) in cross validation data = (34
0, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [254]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----


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 probabilités we use Log-probability estimat
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

```

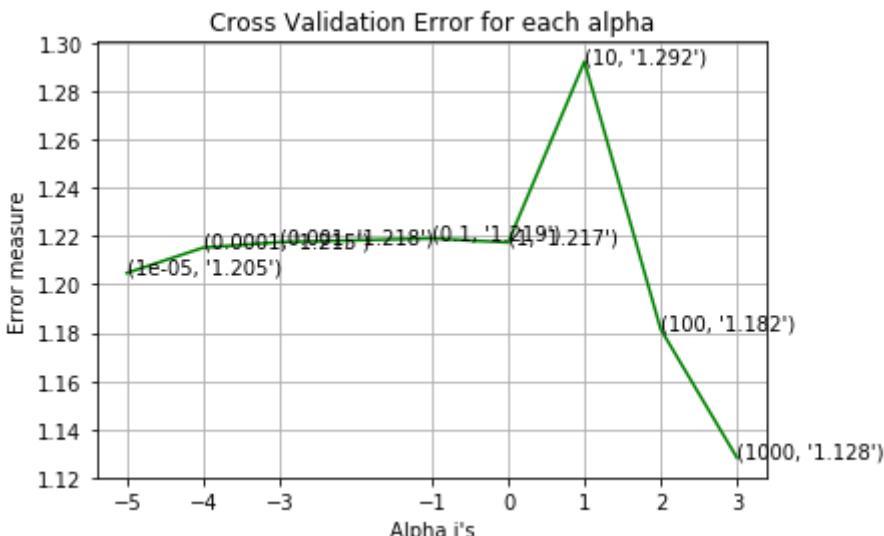
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-05
Log Loss : 1.204762791000572
for alpha = 0.0001
Log Loss : 1.2152190183343217
for alpha = 0.001
Log Loss : 1.2175133688140967
for alpha = 0.01
Log Loss : 1.2190258476975688
for alpha = 0.1
Log Loss : 1.217395717034539
for alpha = 1
Log Loss : 1.2920940527552667
for alpha = 10
Log Loss : 1.1815186644500686
for alpha = 100
Log Loss : 1.1284635720784413

```



```

For values of best alpha = 1000 The train log loss is: 0.9196816371783487
For values of best alpha = 1000 The cross validation log loss is: 1.1284635
720784413
For values of best alpha = 1000 The test log loss is: 1.1807045818657997

```

4.1.1.2. Testing the model with best hyper parameters

In [255]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----
```



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----
```

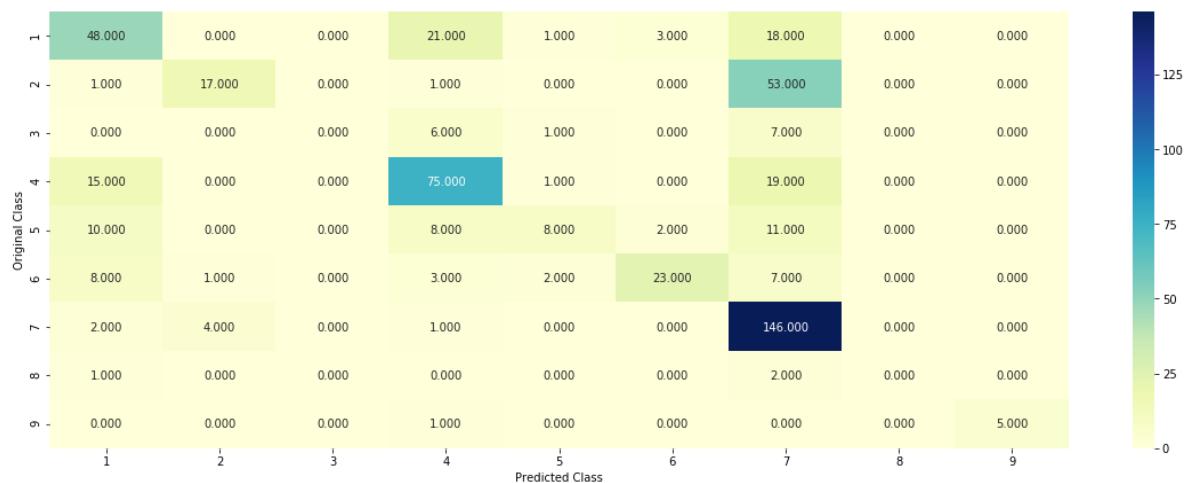


```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites 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)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

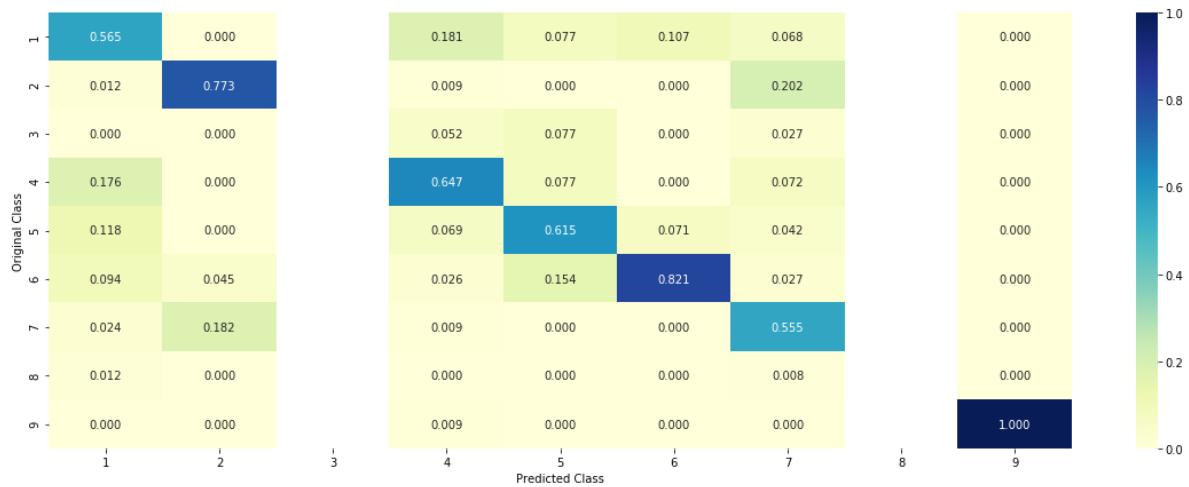
Log Loss : 1.1284635720784413

Number of missclassified point : 0.39473684210526316

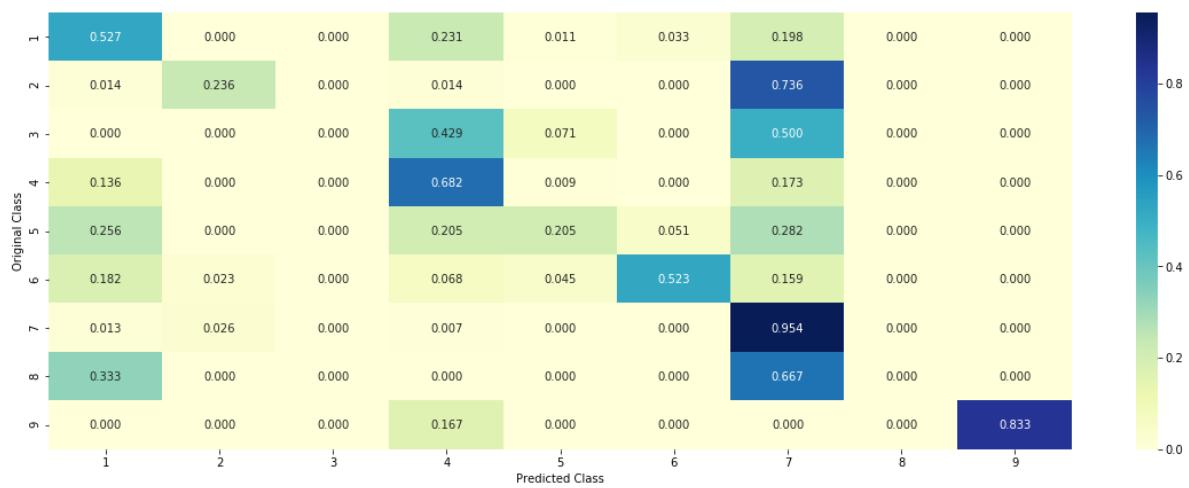
----- Confusion matrix -----



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [261]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6597 0.0136 0.0054 0.041  0.0778 0.1834
0.0158 0.0021 0.0012]]
```

```
Actual Class : 6
```

```
-----  
11 Text feature [protein] present in test data point [True]  
12 Text feature [one] present in test data point [True]  
13 Text feature [type] present in test data point [True]  
16 Text feature [involved] present in test data point [True]  
17 Text feature [two] present in test data point [True]  
18 Text feature [functions] present in test data point [True]  
19 Text feature [region] present in test data point [True]  
20 Text feature [function] present in test data point [True]  
21 Text feature [results] present in test data point [True]  
22 Text feature [role] present in test data point [True]  
23 Text feature [dna] present in test data point [True]  
25 Text feature [wild] present in test data point [True]  
26 Text feature [binding] present in test data point [True]  
27 Text feature [also] present in test data point [True]  
28 Text feature [loss] present in test data point [True]  
29 Text feature [table] present in test data point [True]  
30 Text feature [gene] present in test data point [True]  
31 Text feature [possible] present in test data point [True]  
32 Text feature [either] present in test data point [True]  
33 Text feature [affect] present in test data point [True]  
34 Text feature [expression] present in test data point [True]  
35 Text feature [containing] present in test data point [True]  
37 Text feature [indicate] present in test data point [True]  
38 Text feature [human] present in test data point [True]  
40 Text feature [determined] present in test data point [True]  
41 Text feature [three] present in test data point [True]  
42 Text feature [using] present in test data point [True]  
44 Text feature [however] present in test data point [True]  
45 Text feature [specific] present in test data point [True]  
46 Text feature [including] present in test data point [True]  
47 Text feature [genes] present in test data point [True]  
49 Text feature [four] present in test data point [True]  
50 Text feature [may] present in test data point [True]  
51 Text feature [domains] present in test data point [True]  
52 Text feature [cancer] present in test data point [True]  
54 Text feature [suggest] present in test data point [True]  
56 Text feature [effect] present in test data point [True]  
57 Text feature [analysis] present in test data point [True]  
58 Text feature [transcriptional] present in test data point [True]  
60 Text feature [well] present in test data point [True]  
61 Text feature [important] present in test data point [True]  
62 Text feature [several] present in test data point [True]  
63 Text feature [similar] present in test data point [True]
```

```
64 Text feature [multiple] present in test data point [True]
65 Text feature [shown] present in test data point [True]
66 Text feature [many] present in test data point [True]
67 Text feature [observed] present in test data point [True]
68 Text feature [sequences] present in test data point [True]
69 Text feature [form] present in test data point [True]
70 Text feature [amino] present in test data point [True]
71 Text feature [conserved] present in test data point [True]
72 Text feature [indicating] present in test data point [True]
73 Text feature [mediated] present in test data point [True]
76 Text feature [highly] present in test data point [True]
77 Text feature [previous] present in test data point [True]
78 Text feature [identify] present in test data point [True]
79 Text feature [thus] present in test data point [True]
80 Text feature [contains] present in test data point [True]
81 Text feature [previously] present in test data point [True]
82 Text feature [critical] present in test data point [True]
83 Text feature [interacts] present in test data point [True]
84 Text feature [structure] present in test data point [True]
85 Text feature [essential] present in test data point [True]
86 Text feature [described] present in test data point [True]
87 Text feature [whether] present in test data point [True]
88 Text feature [terminal] present in test data point [True]
89 Text feature [addition] present in test data point [True]
90 Text feature [likely] present in test data point [True]
91 Text feature [corresponding] present in test data point [True]
92 Text feature [following] present in test data point [True]
93 Text feature [proteins] present in test data point [True]
94 Text feature [ability] present in test data point [True]
95 Text feature [based] present in test data point [True]
96 Text feature [complex] present in test data point [True]
97 Text feature [directly] present in test data point [True]
98 Text feature [remaining] present in test data point [True]
```

Out of the top 100 features 76 are present in query point

4.1.1.4. Feature Importance, Incorrectly classified point

In [262]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[2.180e-02 3.620e-02 2.000e-04 4.250e-02 1.0
80e-02 8.200e-03 8.798e-01
5.000e-04 0.000e+00]]
Actual Class : 5
```

```
-----
16 Text feature [cells] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
19 Text feature [activated] present in test data point [True]
20 Text feature [activation] present in test data point [True]
21 Text feature [cell] present in test data point [True]
22 Text feature [presence] present in test data point [True]
23 Text feature [contrast] present in test data point [True]
24 Text feature [phosphorylation] present in test data point [True]
25 Text feature [downstream] present in test data point [True]
26 Text feature [expressing] present in test data point [True]
27 Text feature [shown] present in test data point [True]
28 Text feature [factor] present in test data point [True]
29 Text feature [inhibitor] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [signaling] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [growth] present in test data point [True]
35 Text feature [addition] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [suggest] present in test data point [True]
38 Text feature [compared] present in test data point [True]
39 Text feature [recently] present in test data point [True]
40 Text feature [treated] present in test data point [True]
41 Text feature [similar] present in test data point [True]
42 Text feature [independent] present in test data point [True]
43 Text feature [previously] present in test data point [True]
44 Text feature [increased] present in test data point [True]
45 Text feature [tyrosine] present in test data point [True]
46 Text feature [found] present in test data point [True]
47 Text feature [1a] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [potential] present in test data point [True]
50 Text feature [showed] present in test data point [True]
51 Text feature [treatment] present in test data point [True]
52 Text feature [mutant] present in test data point [True]
53 Text feature [using] present in test data point [True]
56 Text feature [fig] present in test data point [True]
57 Text feature [higher] present in test data point [True]
58 Text feature [figure] present in test data point [True]
59 Text feature [described] present in test data point [True]
60 Text feature [mutations] present in test data point [True]
63 Text feature [obtained] present in test data point [True]
```

```
64 Text feature [3b] present in test data point [True]
65 Text feature [constitutive] present in test data point [True]
66 Text feature [sensitive] present in test data point [True]
67 Text feature [consistent] present in test data point [True]
68 Text feature [total] present in test data point [True]
69 Text feature [interestingly] present in test data point [True]
70 Text feature [mechanism] present in test data point [True]
71 Text feature [pathways] present in test data point [True]
72 Text feature [including] present in test data point [True]
74 Text feature [expression] present in test data point [True]
75 Text feature [reported] present in test data point [True]
76 Text feature [absence] present in test data point [True]
77 Text feature [activating] present in test data point [True]
78 Text feature [expressed] present in test data point [True]
79 Text feature [observed] present in test data point [True]
80 Text feature [inhibition] present in test data point [True]
81 Text feature [proliferation] present in test data point [True]
84 Text feature [followed] present in test data point [True]
87 Text feature [respectively] present in test data point [True]
88 Text feature [may] present in test data point [True]
89 Text feature [approximately] present in test data point [True]
90 Text feature [without] present in test data point [True]
91 Text feature [mutation] present in test data point [True]
92 Text feature [3a] present in test data point [True]
93 Text feature [identified] present in test data point [True]
94 Text feature [performed] present in test data point [True]
95 Text feature [antibody] present in test data point [True]
96 Text feature [increase] present in test data point [True]
97 Text feature [4a] present in test data point [True]
98 Text feature [two] present in test data point [True]
99 Text feature [inhibitors] present in test data point [True]
```

Out of the top 100 features 73 are present in query point

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [263]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/general
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', Leaf_size=30, p=
# 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-nea
#-----

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

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

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

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

```

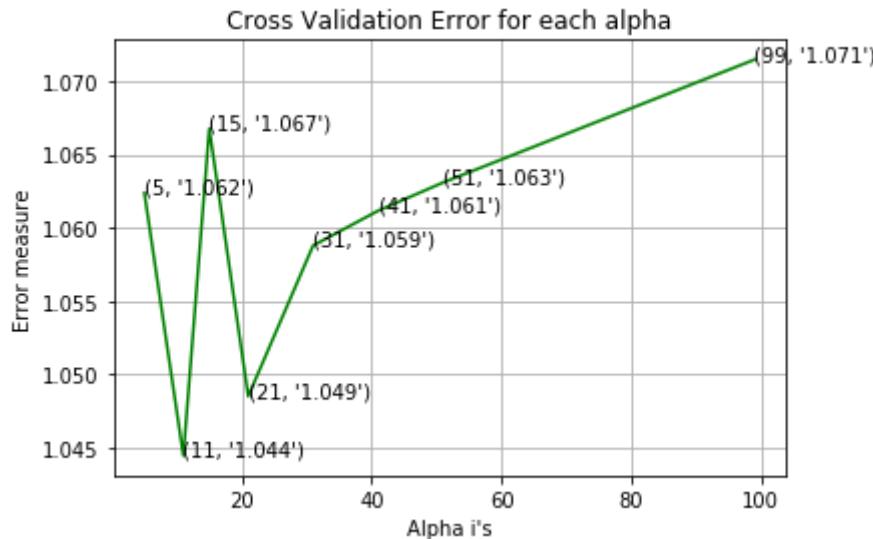
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_

```

```

for alpha = 5
Log Loss : 1.0623805615343493
for alpha = 11
Log Loss : 1.0444452750986575
for alpha = 15
Log Loss : 1.0667705679076356
for alpha = 21
Log Loss : 1.0485047777307632
for alpha = 31
Log Loss : 1.0587909760517968
for alpha = 41
Log Loss : 1.0611629992233091
for alpha = 51
Log Loss : 1.0630889689067622
for alpha = 99
Log Loss : 1.071484468392315

```



```

For values of best alpha = 11 The train log loss is: 0.6312205358693138
For values of best alpha = 11 The cross validation log loss is: 1.044445275
0986575
For values of best alpha = 11 The test log loss is: 1.0135835455369469

```

4.2.2. Testing the model with best hyper parameters

In [264]:

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

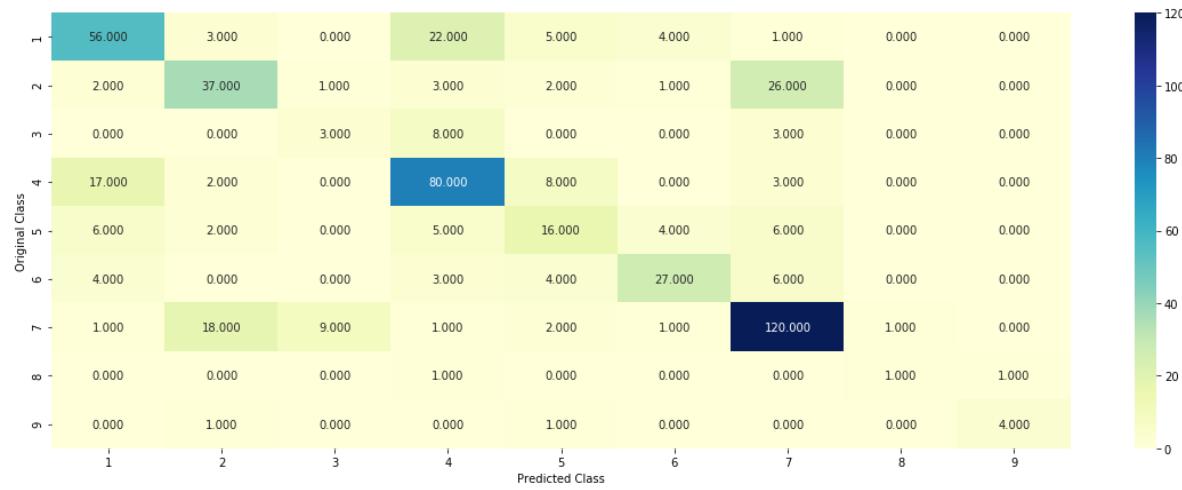
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nea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

```

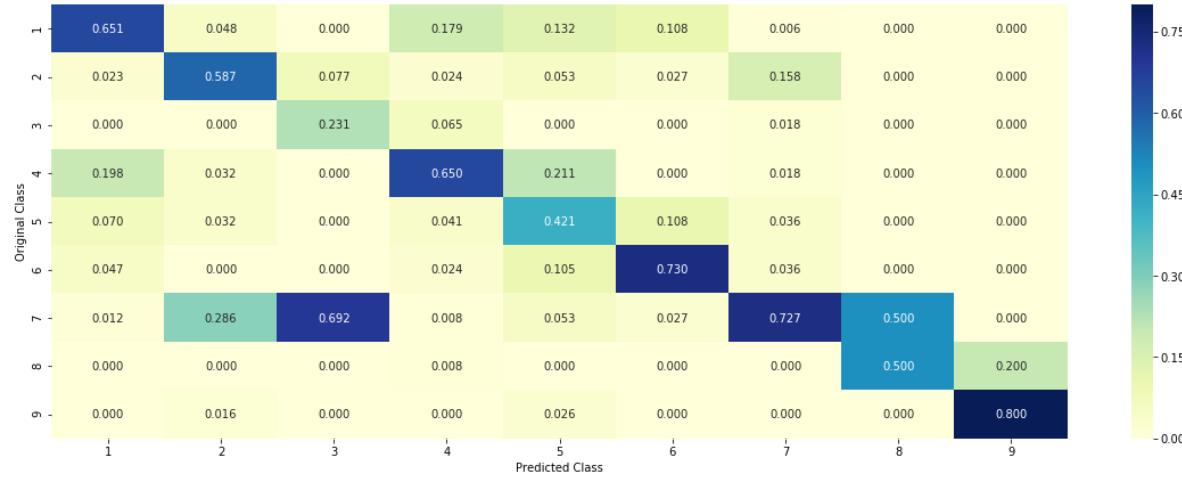
Log loss : 1.0444452750986575

Number of mis-classified points : 0.3533834586466165

----- Confusion matrix -----



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



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



4.2.3. Sample Query point -1

In [265]:

```
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[be
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]])))
```

```
Predicted Class : 7
Actual Class : 6
The 11 nearest neighbours of the test points belongs to classes [1 6 6 5 1
1 1 5 1 1 1]
Frequency of nearest points : Counter({1: 7, 6: 2, 5: 2})
```

4.2.4. Sample Query Point-2

In [266]:

```
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[be
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test pc
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]])))
```

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

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper parameter tuning

In [267]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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/geometry-and-linear-algebra-in-machine-learning/Support-vector-machines-and-kernels/Support-vector-machines-and-kernels-Part-1.ipynb
#-----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
    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 estimation
    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)

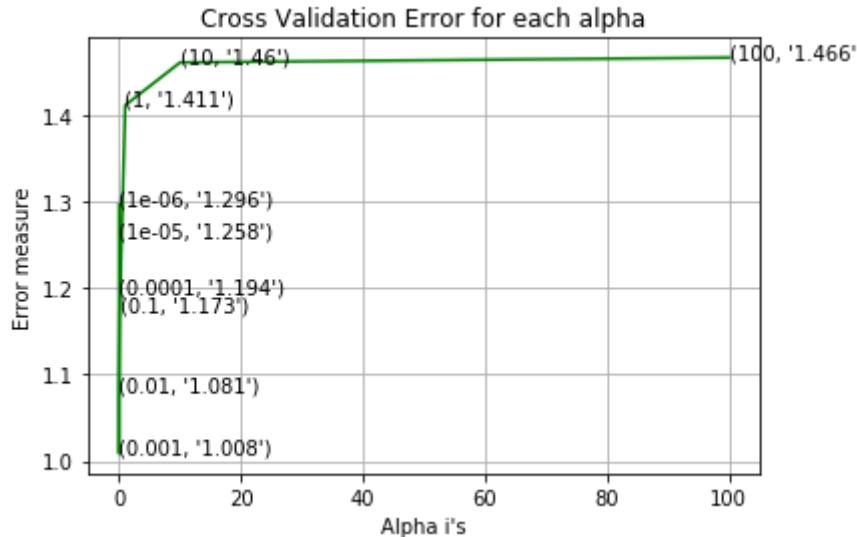
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-06
Log Loss : 1.2964381673283487
for alpha = 1e-05
Log Loss : 1.2580827417830178
for alpha = 0.0001
Log Loss : 1.1944641981873505
for alpha = 0.001
Log Loss : 1.0084458008357113
for alpha = 0.01
Log Loss : 1.0812313477527364
for alpha = 0.1
Log Loss : 1.17305635240371
for alpha = 1
Log Loss : 1.4110040710608245
for alpha = 10
Log Loss : 1.4604673889437099
for alpha = 100
Log Loss : 1.4660677316058104

```



```

For values of best alpha = 0.001 The train log loss is: 0.5727270348156766
For values of best alpha = 0.001 The cross validation log loss is: 1.008445
8008357113
For values of best alpha = 0.001 The test log loss is: 1.0846850852090135

```

4.3.1.2. Testing the model with best hyper parameters

In [268]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

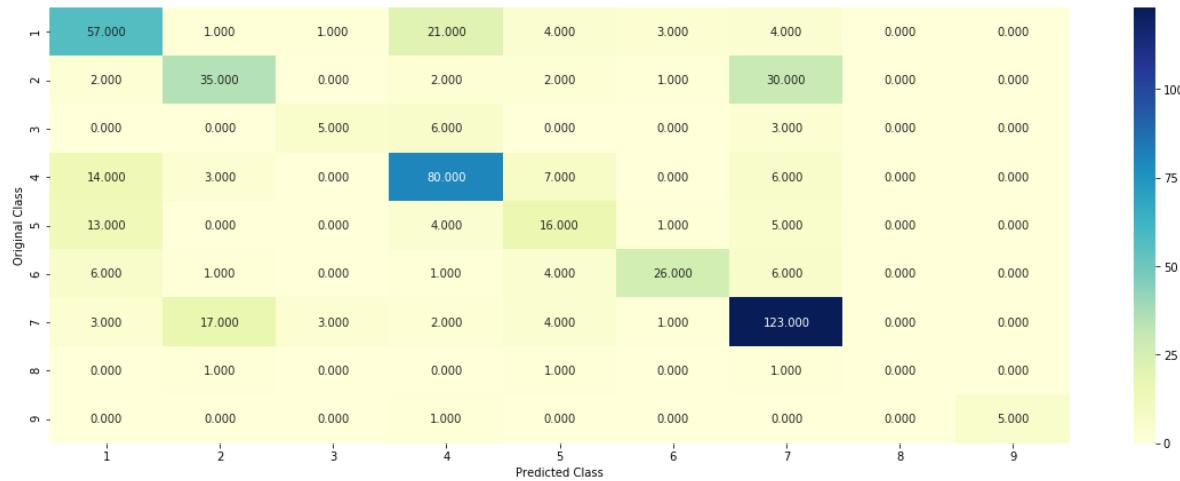
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

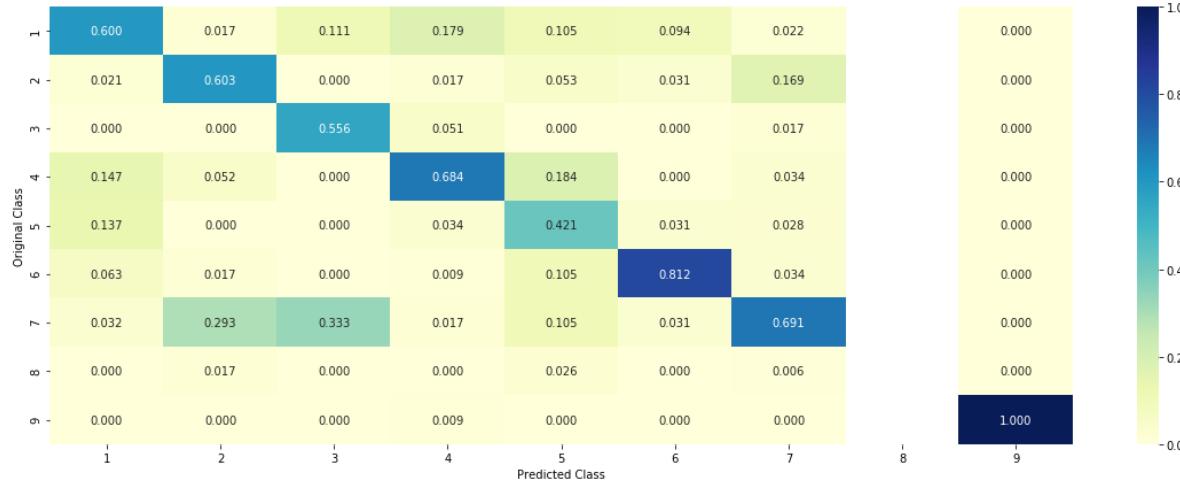
Log loss : 1.0084458008357113

Number of mis-classified points : 0.34774436090225563

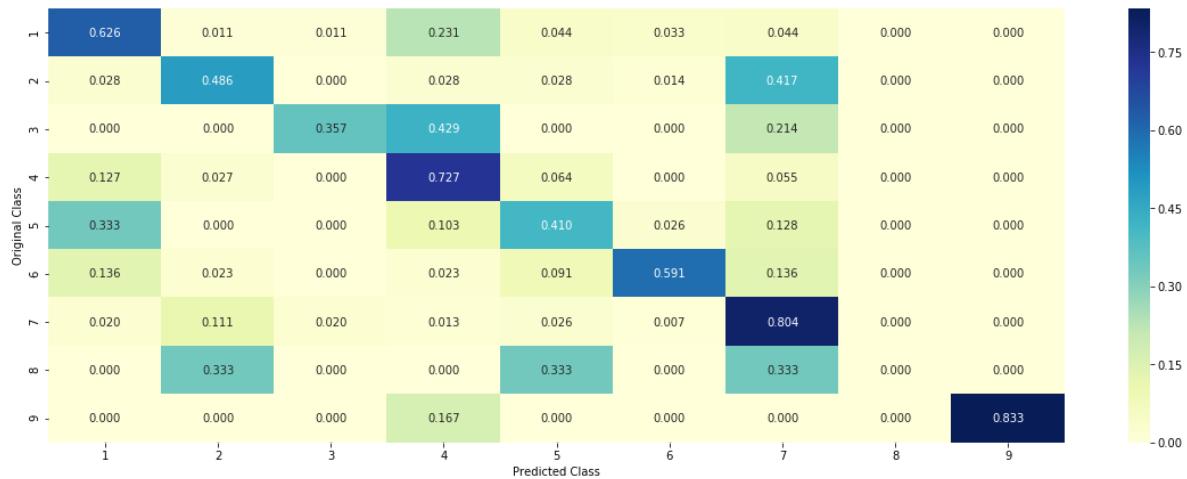
----- Confusion matrix -----



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



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



4.3.1.3. Feature Importance

In [53]:

```
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 [270]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='')
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[0])
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2929 0.0048 0.0043 0.0025 0.2399 0.4432
0.0018 0.0056 0.0051]]
Actual Class : 6
-----
250 Text feature [quartz] present in test data point [True]
441 Text feature [traces] present in test data point [True]
462 Text feature [weakened] present in test data point [True]
Out of the top 500 features 3 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [271]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0057 0.2027 0.0029 0.0074 0.354 0.0029
0.409 0.0086 0.0067]]
Actual Class : 5
```

```
-----
21 Text feature [activated] present in test data point [True]
27 Text feature [constitutive] present in test data point [True]
41 Text feature [3t3] present in test data point [True]
44 Text feature [interventions] present in test data point [True]
67 Text feature [technology] present in test data point [True]
99 Text feature [nude] present in test data point [True]
108 Text feature [activation] present in test data point [True]
118 Text feature [oncogenes] present in test data point [True]
137 Text feature [activate] present in test data point [True]
141 Text feature [oncogene] present in test data point [True]
205 Text feature [transforming] present in test data point [True]
206 Text feature [phosphorylation] present in test data point [True]
208 Text feature [transformation] present in test data point [True]
227 Text feature [agar] present in test data point [True]
259 Text feature [erk] present in test data point [True]
269 Text feature [murine] present in test data point [True]
273 Text feature [expressing] present in test data point [True]
348 Text feature [guanine] present in test data point [True]
379 Text feature [abd] present in test data point [True]
394 Text feature [lay] present in test data point [True]
410 Text feature [downstream] present in test data point [True]
425 Text feature [activating] present in test data point [True]
457 Text feature [definitively] present in test data point [True]
458 Text feature [transform] present in test data point [True]
470 Text feature [akt3] present in test data point [True]
Out of the top 500 features 25 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper parameter tuning

In [272]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



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



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

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



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

```

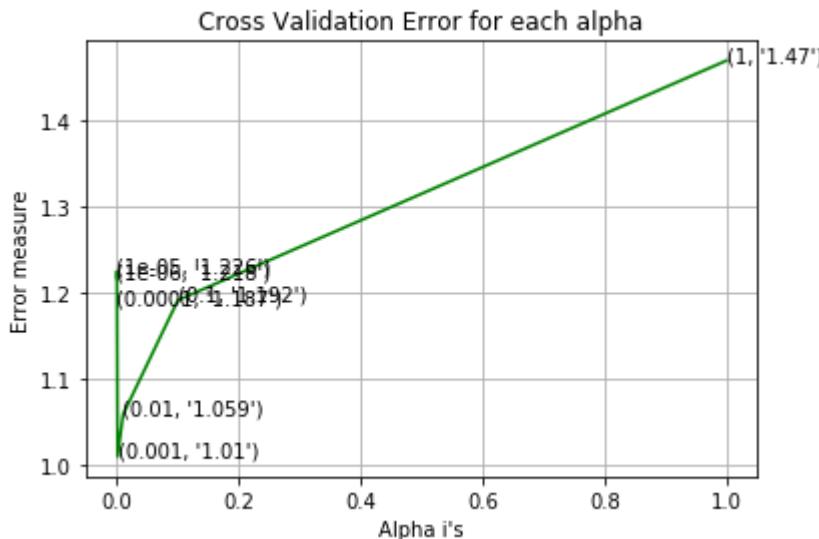
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-06
Log Loss : 1.2176384198207126
for alpha = 1e-05
Log Loss : 1.225526999973934
for alpha = 0.0001
Log Loss : 1.1871724676919033
for alpha = 0.001
Log Loss : 1.010082176546666
for alpha = 0.01
Log Loss : 1.0590243743653187
for alpha = 0.1
Log Loss : 1.1918003360203913
for alpha = 1
Log Loss : 1.4700114623430167

```



```

For values of best alpha = 0.001 The train log loss is: 0.5572454189207187
For values of best alpha = 0.001 The cross validation log loss is: 1.010082
176546666
For values of best alpha = 0.001 The test log loss is: 1.1028651252976287

```

4.3.2.2. Testing model with best hyper parameters

In [273]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

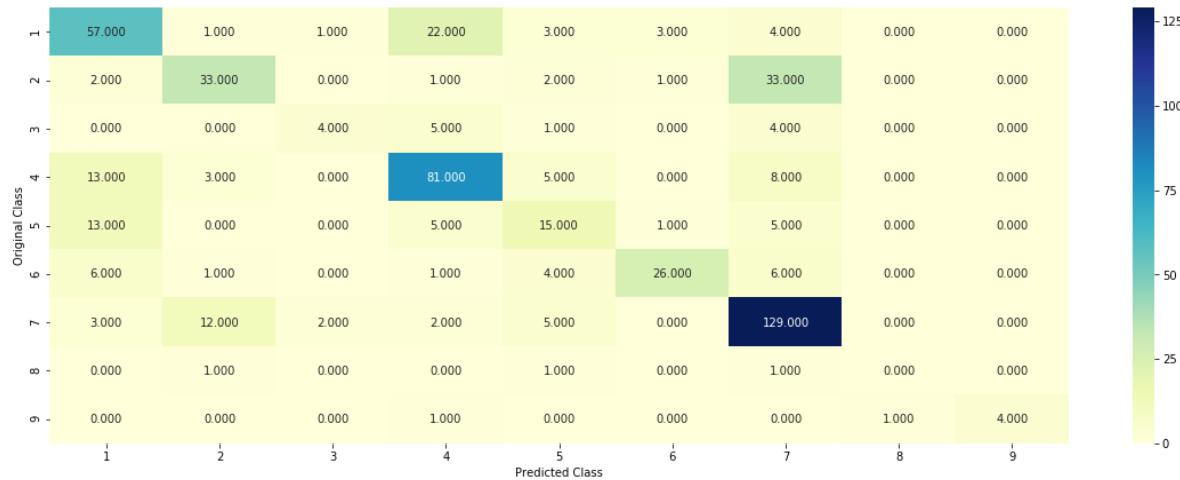
#-----
# video link:
#-----
```

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, c

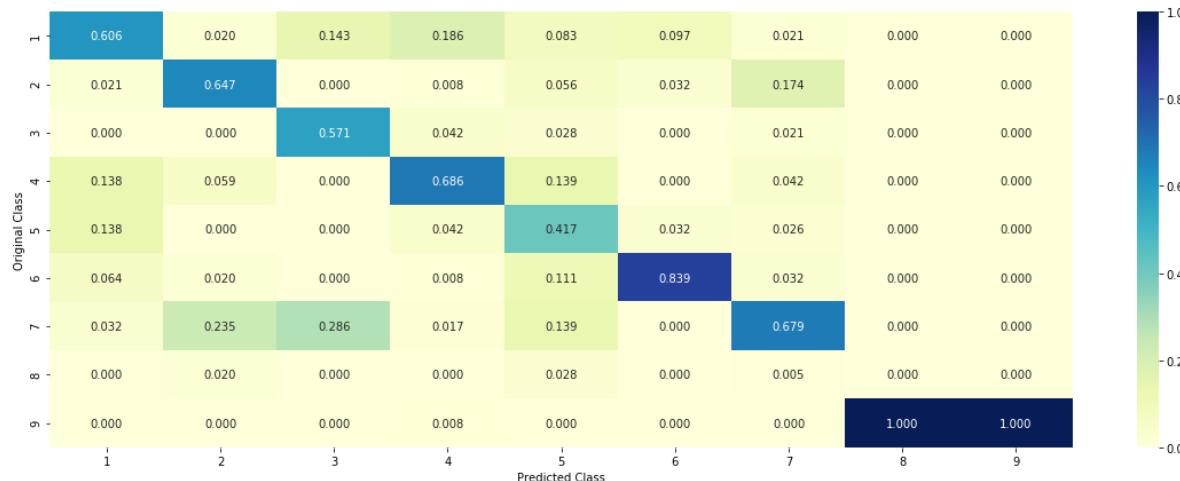
Log loss : 1.010082176546666

Number of mis-classified points : 0.34398496240601506

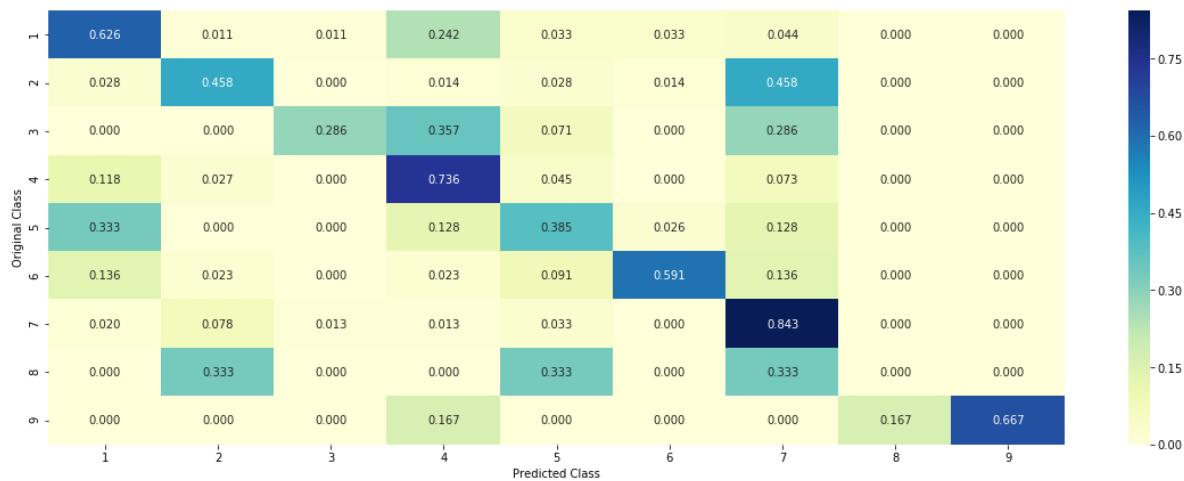
----- Confusion matrix -----



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [274]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.329  0.0057  0.0011  0.0044  0.2125  0.4412
0.0051  0.0005  0.0005]]
Actual Class : 6
-----
291 Text feature [quartz] present in test data point [True]
400 Text feature [traces] present in test data point [True]
450 Text feature [weakened] present in test data point [True]
Out of the top 500 features 3 are present in query point
```

4.3.2.4. Feature Importance, Incorrectly Classified point

In [275]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[6.900e-03 2.054e-01 4.000e-04 1.100e-02 2.2
32e-01 1.800e-03 5.512e-01
 1.000e-04 2.000e-04]]
Actual Class : 5
-----
54 Text feature [activated] present in test data point [True]
79 Text feature [3t3] present in test data point [True]
86 Text feature [constitutive] present in test data point [True]
98 Text feature [interventions] present in test data point [True]
160 Text feature [activation] present in test data point [True]
161 Text feature [transformation] present in test data point [True]
164 Text feature [phosphorylation] present in test data point [True]
166 Text feature [activate] present in test data point [True]
181 Text feature [expressing] present in test data point [True]
183 Text feature [agar] present in test data point [True]
185 Text feature [nude] present in test data point [True]
187 Text feature [technology] present in test data point [True]
198 Text feature [transforming] present in test data point [True]
208 Text feature [abd] present in test data point [True]
210 Text feature [oncogenes] present in test data point [True]
219 Text feature [oncogene] present in test data point [True]
258 Text feature [activating] present in test data point [True]
308 Text feature [definitively] present in test data point [True]
327 Text feature [2d] present in test data point [True]
334 Text feature [downstream] present in test data point [True]
353 Text feature [erk] present in test data point [True]
380 Text feature [transform] present in test data point [True]
392 Text feature [phospho] present in test data point [True]
397 Text feature [signaling] present in test data point [True]
411 Text feature [murine] present in test data point [True]
432 Text feature [ba] present in test data point [True]
448 Text feature [f3] present in test data point [True]
459 Text feature [adenocarcinoma] present in test data point [True]
463 Text feature [akt3] present in test data point [True]
490 Text feature [independently] present in test data point [True]
498 Text feature [transformed] present in test data point [True]
Out of the top 500 features 31 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper parameter tuning

In [276]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



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



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

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



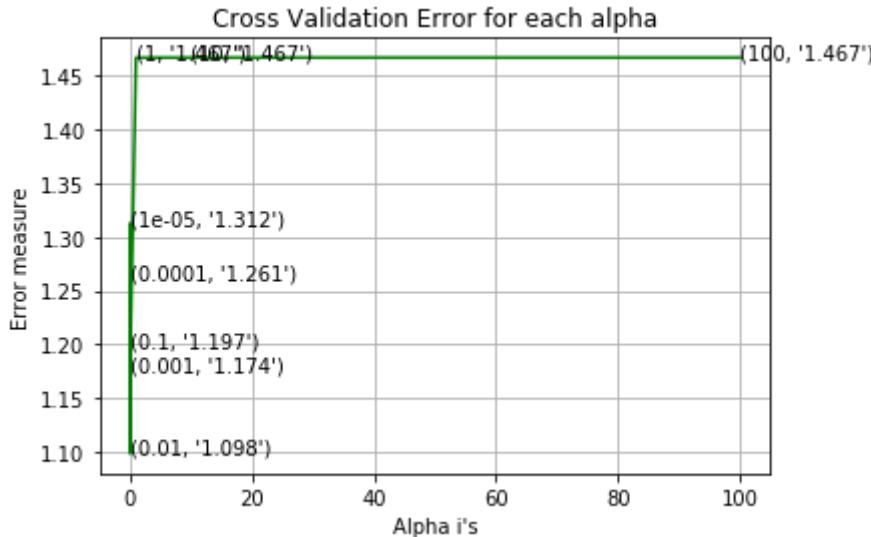
```
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge')
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")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

```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
```

for C = 1e-05
Log Loss : 1.312448071918245
for C = 0.0001
Log Loss : 1.2607466879772955
for C = 0.001
Log Loss : 1.1744907878804276
for C = 0.01
Log Loss : 1.098443262552807
for C = 0.1
Log Loss : 1.1969904932915472
for C = 1
Log Loss : 1.4669986302138944
for C = 10
Log Loss : 1.46709293532397
for C = 100
Log Loss : 1.4670929616388797



For values of best alpha = 0.01 The train log loss is: 0.6871337792590879
For values of best alpha = 0.01 The cross validation log loss is: 1.0984432
62552807
For values of best alpha = 0.01 The test log loss is: 1.1834357651739926

4.4.2. Testing model with best hyper parameters

In [277]:

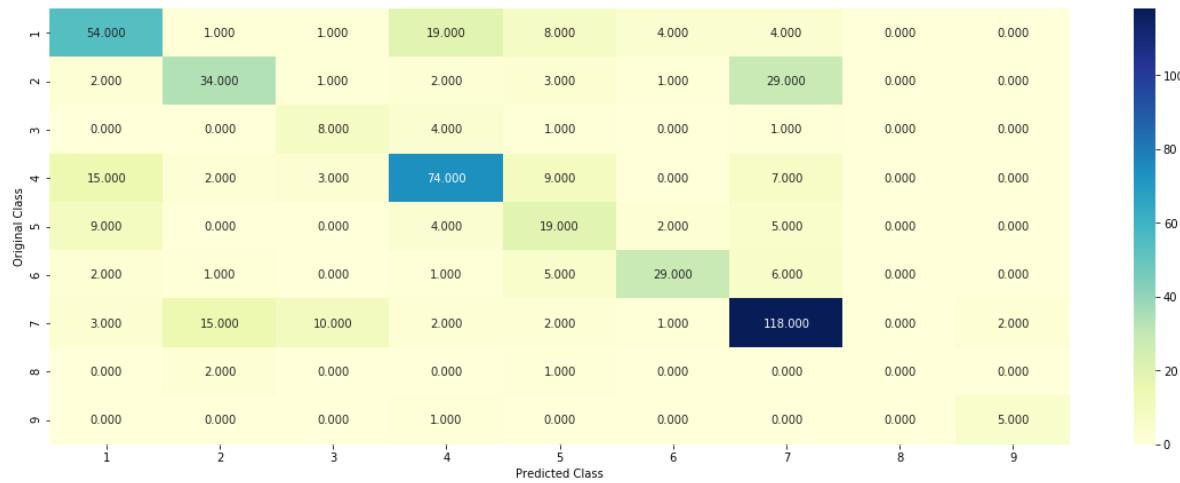
```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video Link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM
# -----
```

`clf = SVC(C=alpha[best_alpha],kernel='Linear',probability=True, class_weight='balanced')`
`clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42, class_weight='balanced')`

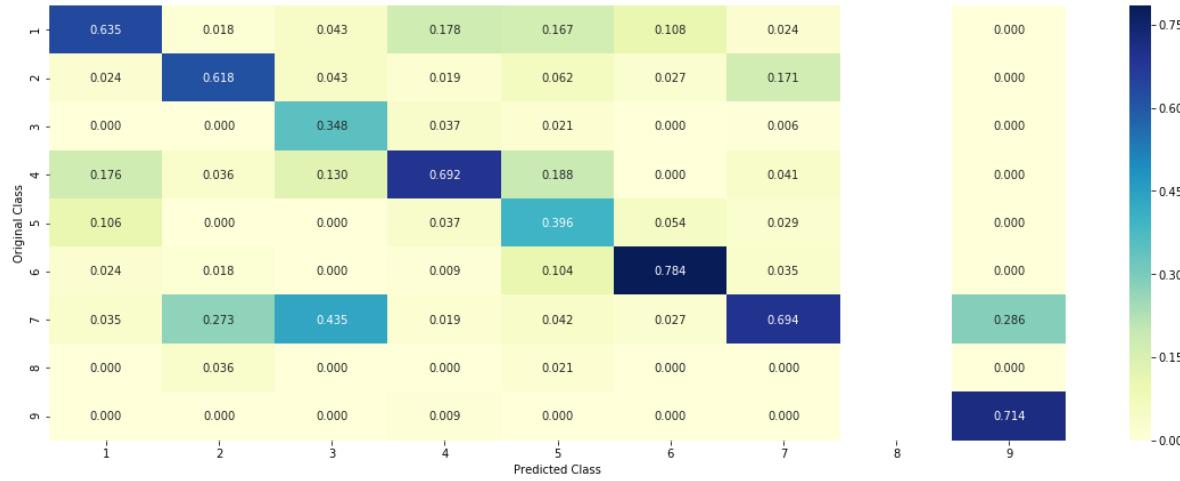
Log loss : 1.098443262552807

Number of mis-classified points : 0.35902255639097747

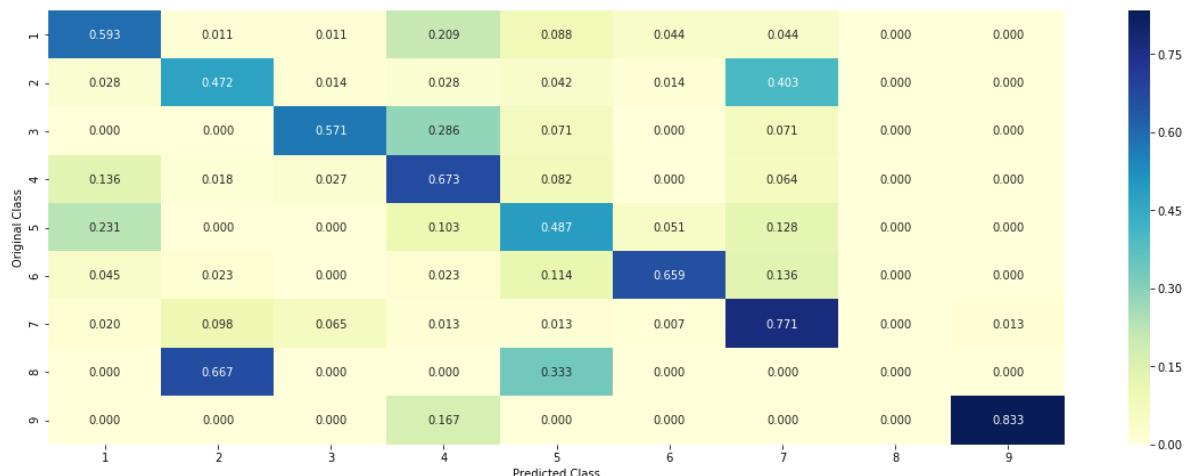
----- Confusion matrix -----



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [278]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2554 0.0413 0.0204 0.0336 0.1244 0.4091
0.0815 0.0179 0.0164]]
Actual Class : 6
-----
222 Text feature [traces] present in test data point [True]
367 Text feature [models] present in test data point [True]
Out of the top 500 features 2 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [279]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0264 0.0916 0.0081 0.0662 0.2687 0.0252
0.49 0.0127 0.0111]]
```

```
Actual Class : 5
```

```
-----  
20 Text feature [constitutive] present in test data point [True]  
21 Text feature [activated] present in test data point [True]  
32 Text feature [3t3] present in test data point [True]  
39 Text feature [activation] present in test data point [True]  
43 Text feature [transformation] present in test data point [True]  
44 Text feature [interventions] present in test data point [True]  
47 Text feature [technology] present in test data point [True]  
54 Text feature [expressing] present in test data point [True]  
56 Text feature [activate] present in test data point [True]  
60 Text feature [nude] present in test data point [True]  
81 Text feature [phosphorylation] present in test data point [True]  
86 Text feature [oncogenes] present in test data point [True]  
89 Text feature [oncogene] present in test data point [True]  
97 Text feature [murine] present in test data point [True]  
104 Text feature [agar] present in test data point [True]  
105 Text feature [transforming] present in test data point [True]  
132 Text feature [akt3] present in test data point [True]  
138 Text feature [dimers] present in test data point [True]  
142 Text feature [transform] present in test data point [True]  
149 Text feature [1222] present in test data point [True]  
151 Text feature [f3] present in test data point [True]  
152 Text feature [ba] present in test data point [True]  
156 Text feature [il] present in test data point [True]  
199 Text feature [erk] present in test data point [True]  
207 Text feature [signaling] present in test data point [True]  
221 Text feature [independently] present in test data point [True]  
228 Text feature [transformed] present in test data point [True]  
230 Text feature [activating] present in test data point [True]  
239 Text feature [phospho] present in test data point [True]  
320 Text feature [epitope] present in test data point [True]  
330 Text feature [abd] present in test data point [True]  
368 Text feature [cysteine] present in test data point [True]  
372 Text feature [2d] present in test data point [True]  
374 Text feature [washout] present in test data point [True]  
391 Text feature [downstream] present in test data point [True]  
399 Text feature [1221] present in test data point [True]  
431 Text feature [course] present in test data point [True]  
440 Text feature [3a] present in test data point [True]  
447 Text feature [absence] present in test data point [True]  
453 Text feature [tyrosine] present in test data point [True]  
456 Text feature [afatinib] present in test data point [True]  
483 Text feature [kinase] present in test data point [True]  
Out of the top 500 features 42 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper parameter tuning

In [280]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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-9))
        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_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```
'''  
  
best_alpha = np.argmin(cv_log_error_array)  
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c  
clf.fit(train_x_onehotCoding, train_y)  
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig_clf.fit(train_x_onehotCoding, train_y)  
  
predict_y = sig_clf.predict_proba(train_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:")  
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo  
predict_y = sig_clf.predict_proba(test_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",  
  
for n_estimators = 100 and max depth = 5  
Log Loss : 1.1940656686312001  
for n_estimators = 100 and max depth = 10  
Log Loss : 1.135820863791144  
for n_estimators = 200 and max depth = 5  
Log Loss : 1.178869720918387  
for n_estimators = 200 and max depth = 10  
Log Loss : 1.1265677229312059  
for n_estimators = 500 and max depth = 5  
Log Loss : 1.1732905768825026  
for n_estimators = 500 and max depth = 10  
Log Loss : 1.125541772599846  
for n_estimators = 1000 and max depth = 5  
Log Loss : 1.170457970772764  
for n_estimators = 1000 and max depth = 10  
Log Loss : 1.125216195951569  
for n_estimators = 2000 and max depth = 5  
Log Loss : 1.1706984784716132  
for n_estimators = 2000 and max depth = 10  
Log Loss : 1.1261352079980538  
For values of best estimator = 1000 The train log loss is: 0.63850571383954  
82  
For values of best estimator = 1000 The cross validation log loss is: 1.125  
216195951569  
For values of best estimator = 1000 The test log loss is: 1.129120464382871  
8
```

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

In [281]:

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

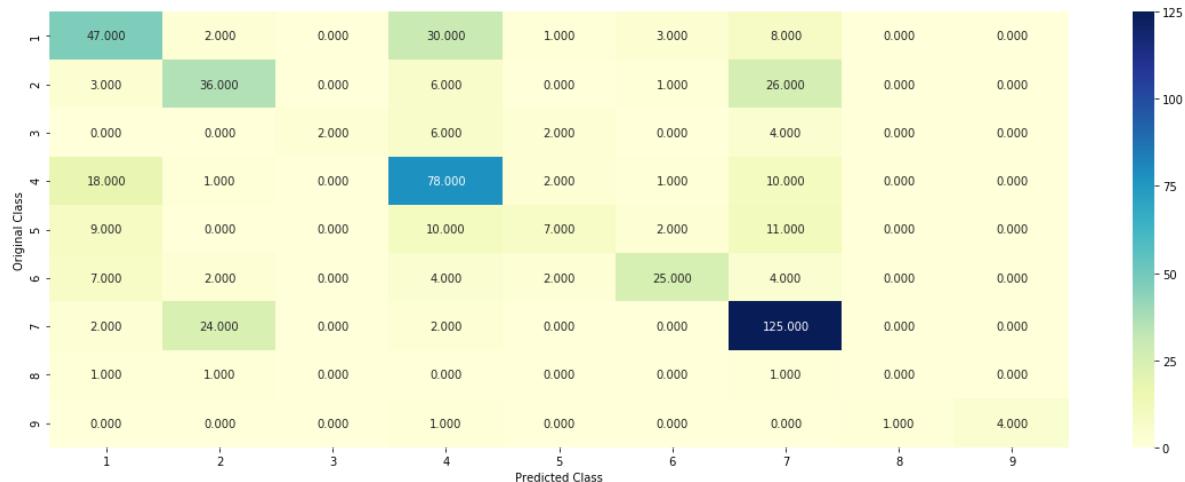
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/rando
# -----
```

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf
```

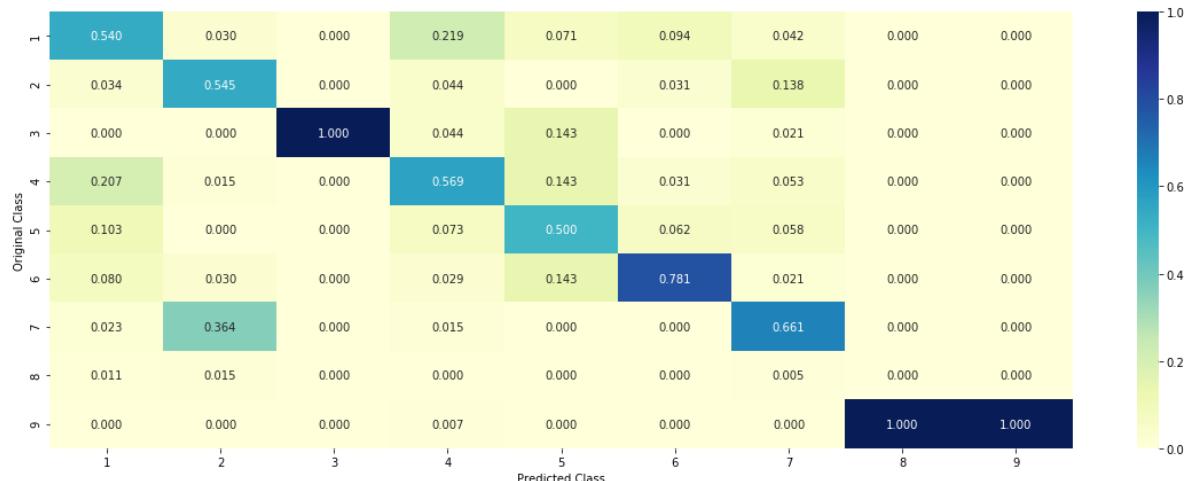
Log loss : 1.125216195951569

Number of mis-classified points : 0.39097744360902253

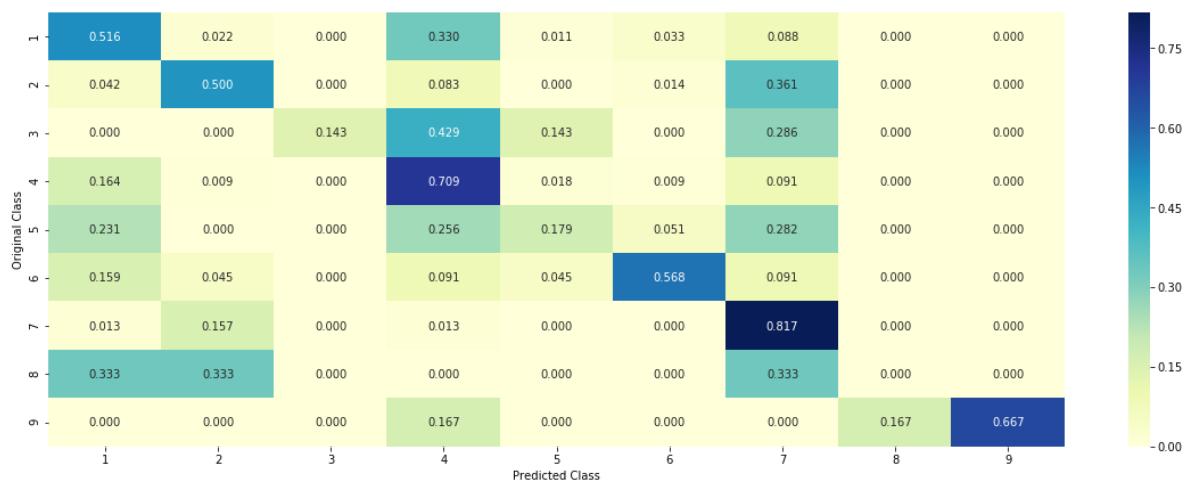
----- Confusion matrix -----



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [282]:

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

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3862 0.0621 0.0222 0.1707 0.0856 0.1779
0.0802 0.0054 0.0099]]
Actual Class : 6
```

```
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [suppressor] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [loss] present in test data point [True]
9 Text feature [phosphorylation] present in test data point [True]
13 Text feature [function] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
17 Text feature [receptor] present in test data point [True]
18 Text feature [missense] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
23 Text feature [kinases] present in test data point [True]
27 Text feature [inhibited] present in test data point [True]
28 Text feature [stability] present in test data point [True]
30 Text feature [phospho] present in test data point [True]
31 Text feature [inhibition] present in test data point [True]
32 Text feature [activate] present in test data point [True]
33 Text feature [downstream] present in test data point [True]
34 Text feature [protein] present in test data point [True]
39 Text feature [cells] present in test data point [True]
54 Text feature [functional] present in test data point [True]
58 Text feature [phosphorylated] present in test data point [True]
64 Text feature [functions] present in test data point [True]
67 Text feature [response] present in test data point [True]
68 Text feature [cell] present in test data point [True]
73 Text feature [potential] present in test data point [True]
75 Text feature [predicted] present in test data point [True]
82 Text feature [proliferation] present in test data point [True]
88 Text feature [pathway] present in test data point [True]
91 Text feature [lines] present in test data point [True]
94 Text feature [lung] present in test data point [True]
Out of the top 100 features 33 are present in query point
```

4.5.3.2. Incorrectly Classified point

In [283]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_df['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0505 0.2897 0.0221 0.0458 0.1556 0.048
0.3764 0.0058 0.0061]]
```

```
Actual Class : 5
```

```
-----  
0 Text feature [kinase] present in test data point [True]  
1 Text feature [suppressor] present in test data point [True]  
2 Text feature [tyrosine] present in test data point [True]  
3 Text feature [activating] present in test data point [True]  
4 Text feature [activation] present in test data point [True]  
5 Text feature [inhibitors] present in test data point [True]  
6 Text feature [activated] present in test data point [True]  
7 Text feature [loss] present in test data point [True]  
8 Text feature [inhibitor] present in test data point [True]  
9 Text feature [phosphorylation] present in test data point [True]  
10 Text feature [oncogenic] present in test data point [True]  
11 Text feature [growth] present in test data point [True]  
12 Text feature [constitutive] present in test data point [True]  
13 Text feature [function] present in test data point [True]  
14 Text feature [treatment] present in test data point [True]  
15 Text feature [nonsense] present in test data point [True]  
16 Text feature [erk] present in test data point [True]  
17 Text feature [receptor] present in test data point [True]  
18 Text feature [missense] present in test data point [True]  
19 Text feature [signaling] present in test data point [True]  
20 Text feature [therapeutic] present in test data point [True]  
22 Text feature [drug] present in test data point [True]  
23 Text feature [kinases] present in test data point [True]  
24 Text feature [f3] present in test data point [True]  
25 Text feature [frameshift] present in test data point [True]  
26 Text feature [transforming] present in test data point [True]  
27 Text feature [inhibited] present in test data point [True]  
29 Text feature [akt] present in test data point [True]  
30 Text feature [phospho] present in test data point [True]  
31 Text feature [inhibition] present in test data point [True]  
32 Text feature [activate] present in test data point [True]  
33 Text feature [downstream] present in test data point [True]  
34 Text feature [protein] present in test data point [True]  
35 Text feature [autophosphorylation] present in test data point [True]  
36 Text feature [resistance] present in test data point [True]  
37 Text feature [clinical] present in test data point [True]  
38 Text feature [patients] present in test data point [True]  
39 Text feature [cells] present in test data point [True]  
40 Text feature [months] present in test data point [True]  
41 Text feature [therapy] present in test data point [True]  
44 Text feature [expressing] present in test data point [True]  
48 Text feature [ba] present in test data point [True]  
49 Text feature [defective] present in test data point [True]
```

```
50 Text feature [3t3] present in test data point [True]
51 Text feature [oncogene] present in test data point [True]
54 Text feature [functional] present in test data point [True]
58 Text feature [phosphorylated] present in test data point [True]
59 Text feature [treated] present in test data point [True]
60 Text feature [extracellular] present in test data point [True]
61 Text feature [mek] present in test data point [True]
62 Text feature [repair] present in test data point [True]
66 Text feature [ic50] present in test data point [True]
67 Text feature [response] present in test data point [True]
68 Text feature [cell] present in test data point [True]
69 Text feature [patient] present in test data point [True]
71 Text feature [sensitive] present in test data point [True]
73 Text feature [potential] present in test data point [True]
74 Text feature [ras] present in test data point [True]
75 Text feature [predicted] present in test data point [True]
76 Text feature [advanced] present in test data point [True]
79 Text feature [respond] present in test data point [True]
80 Text feature [amplification] present in test data point [True]
81 Text feature [nuclear] present in test data point [True]
82 Text feature [proliferation] present in test data point [True]
83 Text feature [trial] present in test data point [True]
86 Text feature [transform] present in test data point [True]
87 Text feature [harboring] present in test data point [True]
88 Text feature [pathway] present in test data point [True]
90 Text feature [pten] present in test data point [True]
91 Text feature [lines] present in test data point [True]
92 Text feature [pi3k] present in test data point [True]
94 Text feature [lung] present in test data point [True]
95 Text feature [classified] present in test data point [True]
99 Text feature [sensitivity] present in test data point [True]
```

Out of the top 100 features 74 are present in query point

4.5.3. Hyper paramter tuning (With Response Coding)

In [284]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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-9))
        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_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```
'''  
  
best_alpha = np.argmin(cv_log_error_array)  
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c  
clf.fit(train_x_responseCoding, train_y)  
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig_clf.fit(train_x_responseCoding, train_y)  
  
predict_y = sig_clf.predict_proba(train_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log  
predict_y = sig_clf.predict_proba(cv_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log lo  
predict_y = sig_clf.predict_proba(test_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_
```

```
for n_estimators = 10 and max depth =  2  
Log Loss : 2.134559183542347  
for n_estimators = 10 and max depth =  3  
Log Loss : 1.6986595236413289  
for n_estimators = 10 and max depth =  5  
Log Loss : 1.3087805771055592  
for n_estimators = 10 and max depth =  10  
Log Loss : 1.7473129038487056  
for n_estimators = 50 and max depth =  2  
Log Loss : 1.713611972456188  
for n_estimators = 50 and max depth =  3  
Log Loss : 1.348759034314188  
for n_estimators = 50 and max depth =  5  
Log Loss : 1.1631729403859215  
for n_estimators = 50 and max depth =  10  
Log Loss : 1.6206307856002518  
for n_estimators = 100 and max depth =  2  
Log Loss : 1.547499750624011  
for n_estimators = 100 and max depth =  3  
Log Loss : 1.367175685104317  
for n_estimators = 100 and max depth =  5  
Log Loss : 1.2096820653557683  
for n_estimators = 100 and max depth =  10  
Log Loss : 1.5865651771717832  
for n_estimators = 200 and max depth =  2  
Log Loss : 1.5744674423408953  
for n_estimators = 200 and max depth =  3  
Log Loss : 1.317098017675394  
for n_estimators = 200 and max depth =  5  
Log Loss : 1.221262283475889  
for n_estimators = 200 and max depth =  10  
Log Loss : 1.5666903796057539  
for n_estimators = 500 and max depth =  2  
Log Loss : 1.5816470808142502  
for n_estimators = 500 and max depth =  3  
Log Loss : 1.4119029712913718  
for n_estimators = 500 and max depth =  5  
Log Loss : 1.2599288508644928  
for n_estimators = 500 and max depth =  10  
Log Loss : 1.6016271902270371  
for n_estimators = 1000 and max depth =  2  
Log Loss : 1.5546387320729438  
for n_estimators = 1000 and max depth =  3  
Log Loss : 1.4105956670693958  
for n_estimators = 1000 and max depth =  5
```

```
Log Loss : 1.260818064638381
for n_estimators = 1000 and max depth = 10
Log Loss : 1.572973476510051
For values of best alpha = 50 The train log loss is: 0.06676936890896665
For values of best alpha = 50 The cross validation log loss is: 1.163172940
3859215
For values of best alpha = 50 The test log loss is: 1.197920364468122
```

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

In [285]:

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

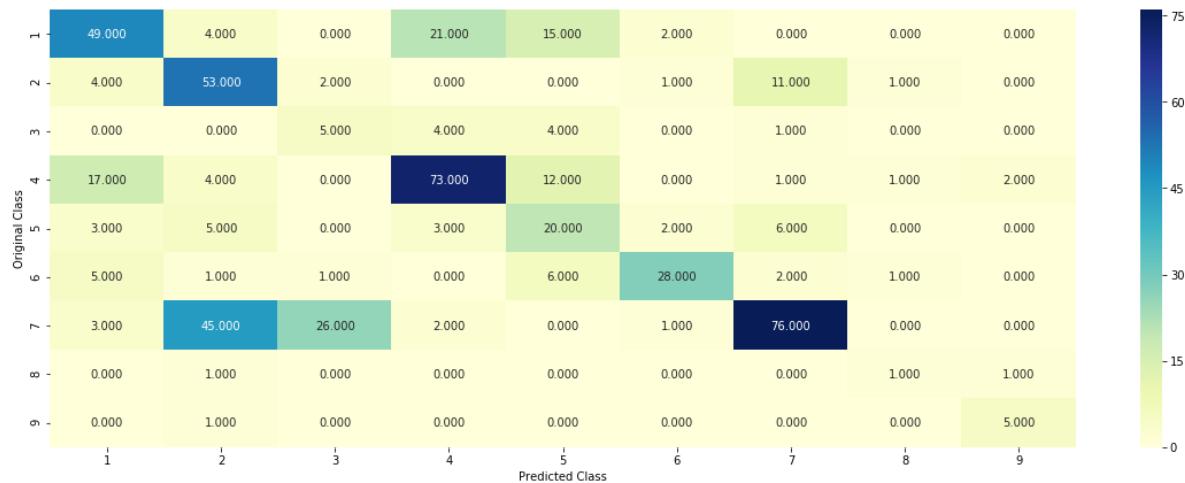
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/random-forest-classifier
# -----
```

```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha%4)])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y,
```

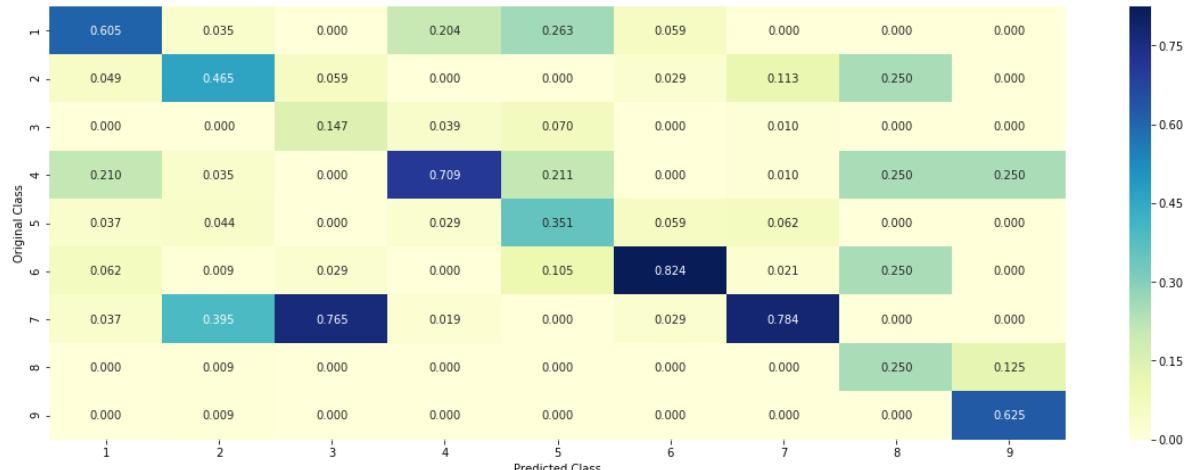
Log loss : 1.1631729403859217

Number of mis-classified points : 0.41729323308270677

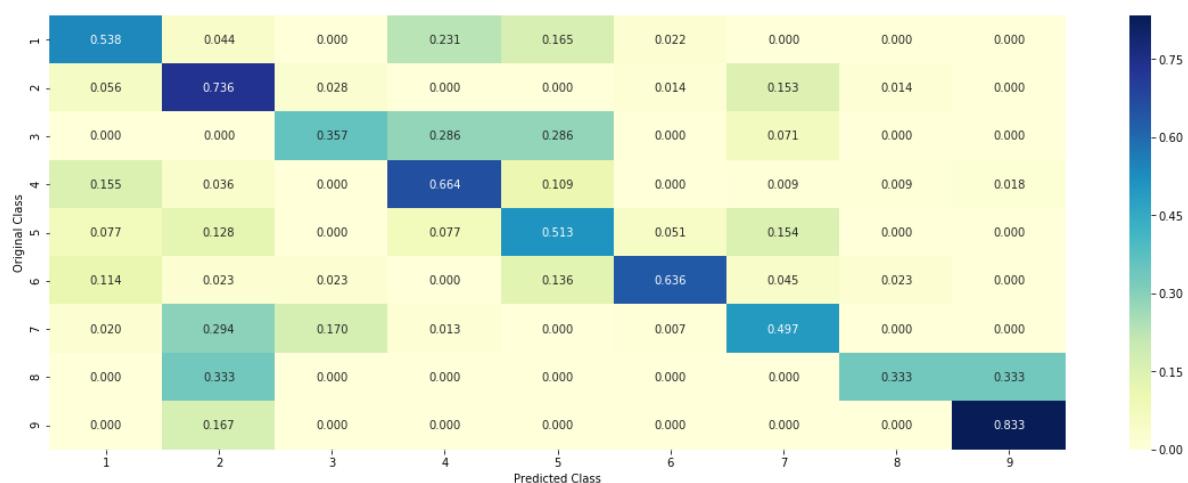
----- Confusion matrix -----



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [286]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCodir
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 : 1
Predicted Class Probabilities: [[0.3371 0.0094 0.0903 0.069  0.2148 0.2541
0.0043 0.0084 0.0126]]
Actual Class : 6
```

```
-----
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
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
```

4.5.5.2. Incorrectly Classified point

In [287]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0238 0.4701 0.0882 0.0295 0.0351 0.0694
0.2534 0.0198 0.0107]]
Actual Class : 5
-----
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
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
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [288]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



```
# read more about support vector machines with Linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=F
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='o

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



```
# read more about support vector machines with Linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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
# -----
```



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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_s
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")
```

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

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

```
Logistic Regression : Log Loss: 1.02
Support vector machines : Log Loss: 1.47
Naive Bayes : Log Loss: 1.22
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.487
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.107
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.198
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.396
```

4.7.2 testing the model with the best hyper parameters

In [289]:

```

lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, n_jobs=-1)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_onehotCoding) != test_y)))
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))

```

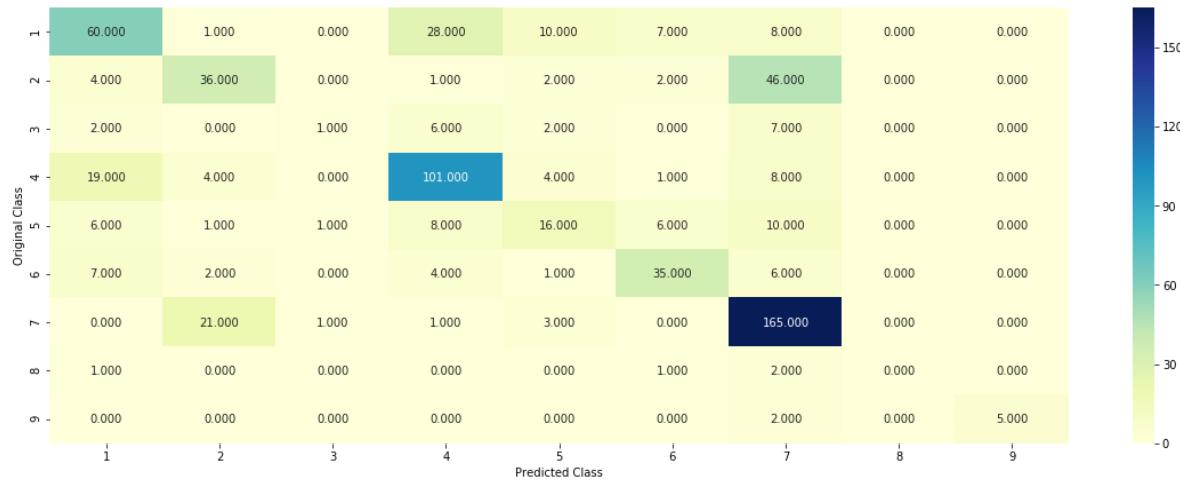
Log loss (train) on the stacking classifier : 0.6266161291069388

Log loss (CV) on the stacking classifier : 1.1071446569115515

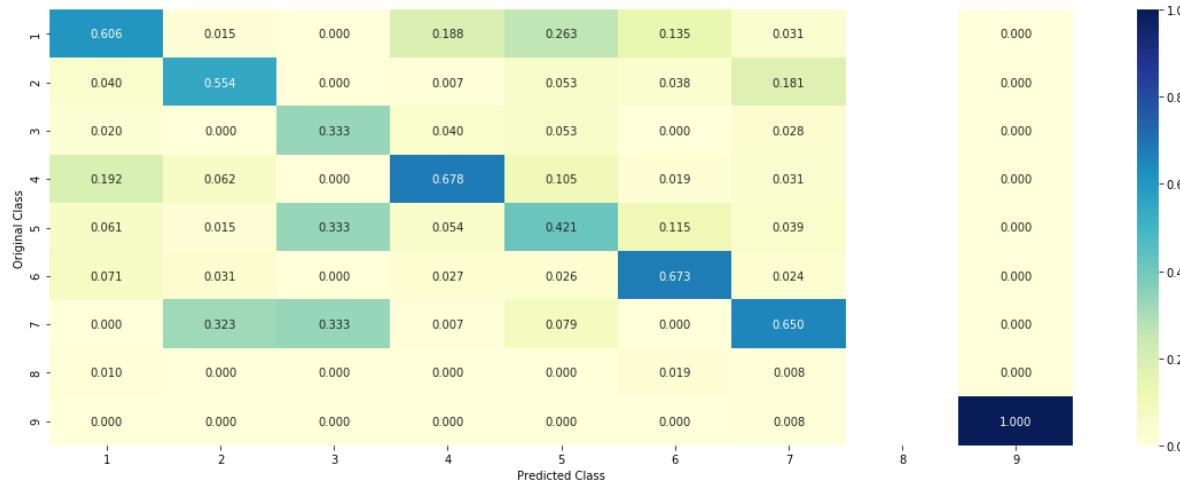
Log loss (test) on the stacking classifier : 1.1364278928245028

Number of missclassified point : 0.3699248120300752

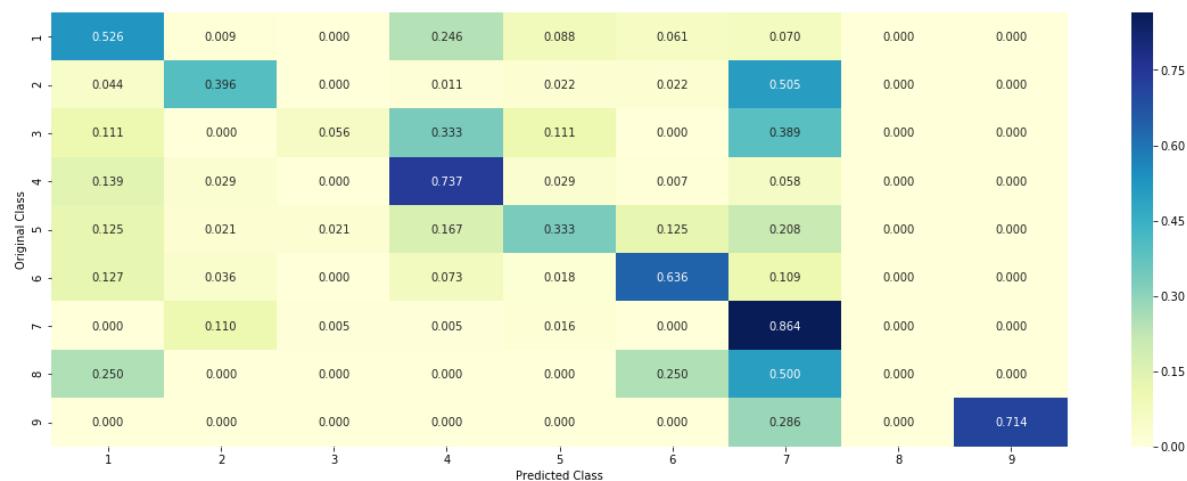
----- Confusion matrix -----



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



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



4.7.3 Maximum Voting classifier

In [290]:

```
#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)], 
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier : ", log_loss(train_y, vclf.predict_proba(tr
print("Log loss (CV) on the VotingClassifier : ", log_loss(cv_y, vclf.predict_proba(cv_x_one
print("Log loss (test) on the VotingClassifier : ", log_loss(test_y, vclf.predict_proba(test
print("Number of missclassified point : ", np.count_nonzero((vclf.predict(test_x_onehotCodir
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

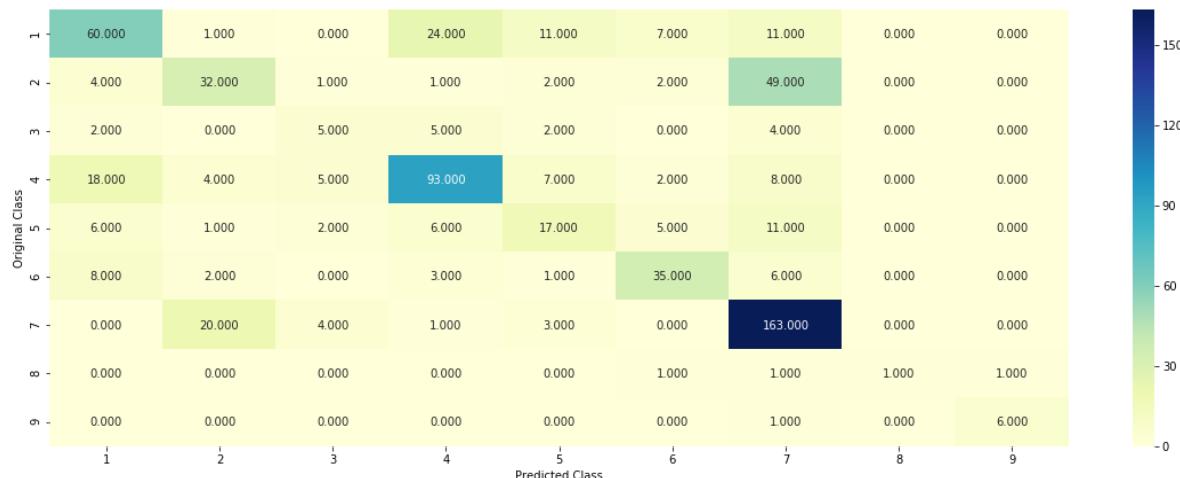
Log loss (train) on the VotingClassifier : 0.8435804441112051

Log loss (CV) on the VotingClassifier : 1.1045785609319974

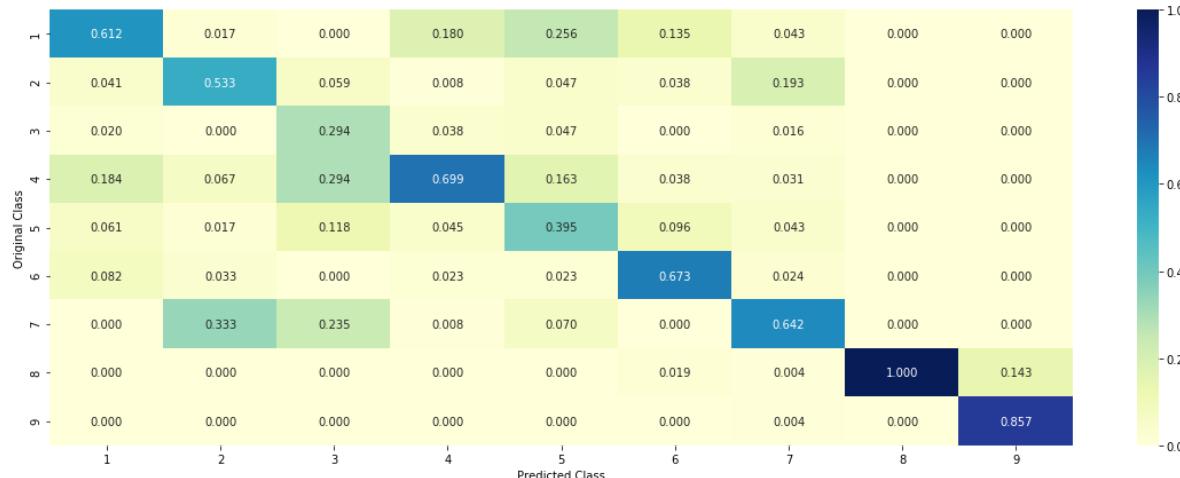
Log loss (test) on the VotingClassifier : 1.1416767089237367

Number of missclassified point : 0.3804511278195489

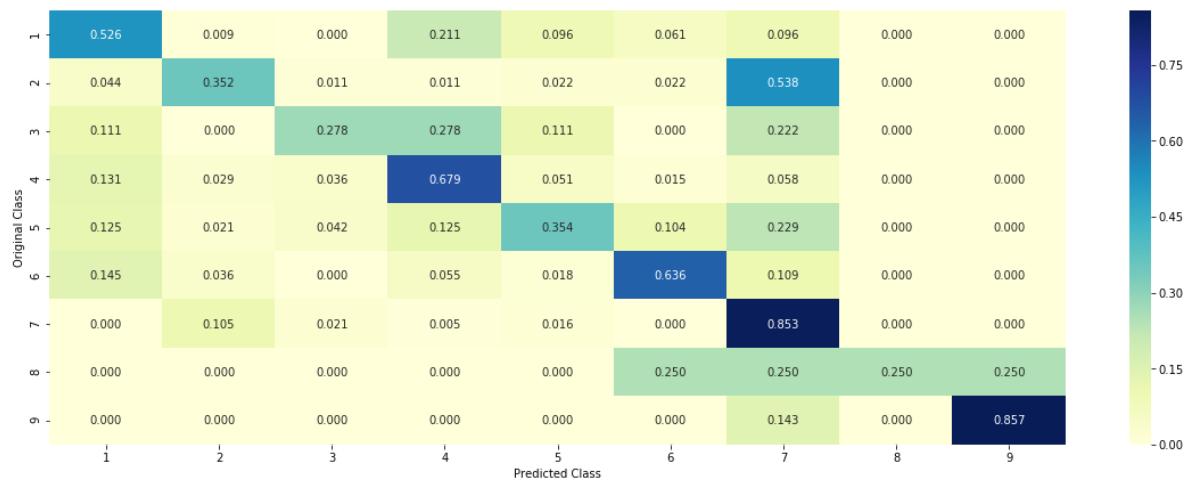
----- Confusion matrix -----



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



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



CONCLUSION

(a). Procedure Followed :-

STEP 1:- Replace CountVectorizer() by TfidfVectorizer() in all the one hot encoding section of gene , variation and text features

In [90]:

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

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

# Training Loss
train_loss = [0.91968163, 0.631220535,0.5727270346,0.557245418,0.68713377,0.638505713,0.066

# Cross Validation loss
cv_loss = [1.1284635,1.04444527,1.0084458008,1.01008217,1.0984432,1.1252161,1.1631729,1.107

# Test loss
test_loss = [1.1807045,1.0135835,1.08468508,1.10286512,1.1834357,1.12912,1.197920364468122,1.1631729,1.107

# Percentage Misclassified points
misclassified = [0.3947368,0.353383,0.3477443,0.34398496,0.359022, 0.390977,0.41729323,0.36

numbering = [1,2,3,4,5,6,7,8,9]
alpha=[1000,11,.001,.001,.01,1000,50,.1,'NaN']
# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
loss	Test_loss	Misclassified(%)		
84635	Naive Bayes 1.1807045	1000 0.3947368	0.91968163	1.12
444527	K-Nearest Neighbour 1.0135835	11 0.353383	0.631220535	1.04
4458008	LR With Class Balancing 1.08468508	0.001 0.3477443	0.5727270346	1.008
008217	LR Without Class Balancing 1.10286512	0.001 0.34398496	0.557245418	1.01
84432	Linear SVM 1.1834357	0.01 0.359022	0.68713377	1.09
52161	RF With One hot Encoding 1.12912	1000 50	0.638505713	1.12
31729	RF With Response Coding 1.197920364468122	0.390977 0.41729323	0.066769368	1.16
8	Stacking Classifier	0.1	0.626616	1.1

07144	1.13642	0.3699248			
9	Maximum Voting Classifier	NaN	0.84358044	1.10	
457856	1.1416767	0.38045			

-----TASK2-----

Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values

In [103]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features)
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 occurred
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 [104]:

```
dict_list = []
# dict_list =[] contains 9 dictionaries 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 [293]:

```
#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 [294]:

```
# 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_
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_re_
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_respon
```

In [105]:

```
# 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 [106]:

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

In [107]:

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

Counter({160.69752091417246: 1, 111.66779690222992: 1, 85.73673777202926: 1, 82.1518763563256: 1, 78.56624289956231: 1, 74.98940181349134: 1, 73.2600346578311: 1, 73.20978425569753: 1, 69.64941923631267: 1, 69.2062004086183: 1, 67.88796346785955: 1, 57.98160797366396: 1, 57.84398907203985: 1, 57.60762069756587: 1, 56.15805740940844: 1, 51.11759503127621: 1, 50.76484045263077: 1, 50.480561507066525: 1, 50.167231525465006: 1, 50.092324840116866: 1, 48.09558887824598: 1, 46.33788052904103: 1, 43.607937554383476: 1, 43.48839507465794: 1, 43.39482784961246: 1, 42.464698872419646: 1, 42.285314977425585: 1, 41.89918871322554: 1, 41.15288537059065: 1, 40.456533650696166: 1, 40.40325621493705: 1, 39.87559501958993: 1, 38.476245042641004: 1, 37.929146737379064: 1, 37.679395404596555: 1, 37.561584777192415: 1, 36.29601996796759: 1, 35.72139782506888: 1, 34.24967297692906: 1, 33.773335170005126: 1, 33.14412984208479: 1, 32.53448735353627: 1, 32.24501067856765: 1, 32.03619035815953: 1, 31.2810501792445: 1, 30.552413259139485: 1, 29.5495548790496: 1, 29.39100580837973: 1, 28.753845105823803: 1, 28.087611634399966: 1, 27.903343261646437: 1, 27.417577657996016: 1, 27.348266468007314: 1, 27.20881639280238: 1, 27.069332853393508: 1, 26.858237790713613: 1, 26.839330439689395: 1, 26.73292988971963: 1, 26.491628959810868: 1, 26.3489667992871: 1, 26.32727505745386: 1, 26.150026888907444: 1, 26.061382752576215: 1, 25.811206246241373: 1, 25.6056202886042263: 1, 25.5612552022622})

In [108]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_textfea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_textfea.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 [109]:

```
# 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,ye
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]" .format(wc
    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,ye

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

In [300]:

```
# Train a Logistic regression+Calibration model using text features which are on-hot encode
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,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

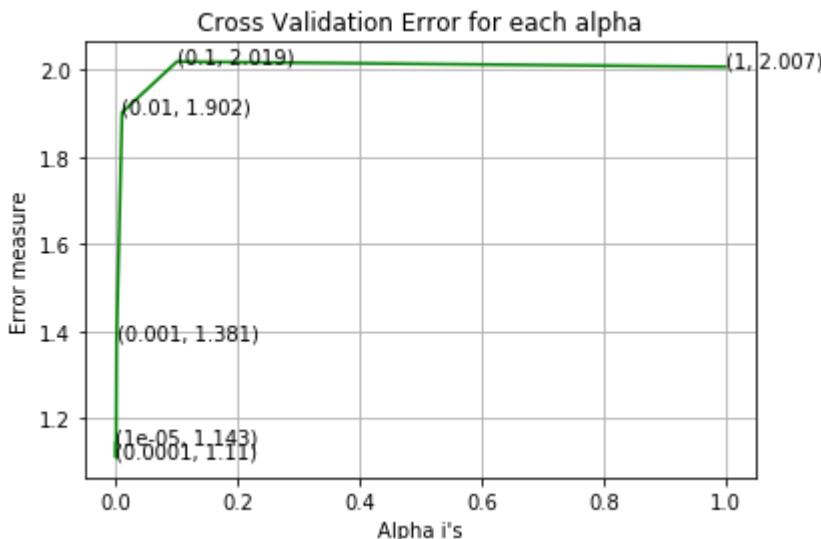
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
 ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
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_cv, predict_y))

For values of alpha = 1e-05 The log loss is: 1.1427920832332985
For values of alpha = 0.0001 The log loss is: 1.1095740275678843
For values of alpha = 0.001 The log loss is: 1.3806565458954045
For values of alpha = 0.01 The log loss is: 1.9021220692415444

For values of alpha = 0.1 The log loss is: 2.0186784949605423
 For values of alpha = 1 The log loss is: 2.0067537257824486



For values of best alpha = 0.0001 The train log loss is: 0.8252443509688081
 For values of best alpha = 0.0001 The cross validation log loss is: 1.1095740275678843
 For values of best alpha = 0.0001 The test log loss is: 1.0929255189849807

In [301]:

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

94.0 % of word of test data appeared in train data
 94.1 % of word of Cross Validation appeared in train data

Stacking the three types of features

In [302]:

```
# 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))
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding))
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 [303]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape[1])
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape[1])
print("(number of data points * number of features) in cross validation data = ", cv_x_onehotCoding.shape[1])
```

One hot encoding features :

```
(number of data points * number of features) in train data = (2124, 3186)
(number of data points * number of features) in test data = (665, 3186)
(number of data points * number of features) in cross validation data = (53, 3186)
```

In [304]:

```
print("Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape[1])
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape[1])
print("(number of data points * number of features) in cross validation data = ", cv_x_responseCoding.shape[1])
```

Response encoding features :

```
(number of data points * number of features) in train data = (2124, 27)
(number of data points * number of features) in test data = (665, 27)
(number of data points * number of features) in cross validation data = (53, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [305]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----


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 probabilités we use Log-probability estimat
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_)

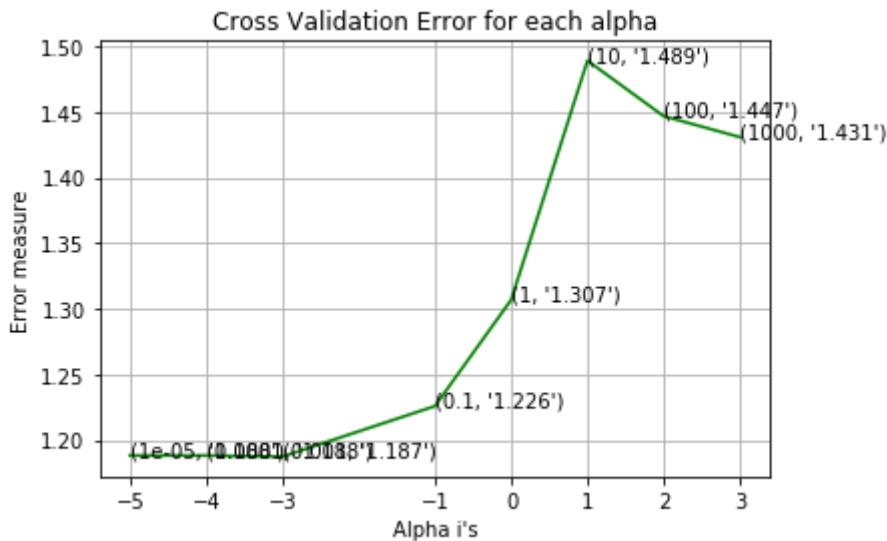
# Variables that will be used in the end to make comparison table of all models
nb_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_
nb_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1
nb_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding),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

```

```

for alpha = 1e-05
Log Loss : 1.1881136351981785
for alpha = 0.0001
Log Loss : 1.1881089730116194
for alpha = 0.001
Log Loss : 1.1872495687422477
for alpha = 0.1
Log Loss : 1.2257461540404357
for alpha = 1
Log Loss : 1.3068371515264645
for alpha = 10
Log Loss : 1.489242967454851
for alpha = 100
Log Loss : 1.4468729530575677
for alpha = 1000
Log Loss : 1.430904045903737

```



```

For values of best alpha = 0.001 The train log loss is: 0.507287200078767
For values of best alpha = 0.001 The cross validation log loss is: 1.187249
5687422477
For values of best alpha = 0.001 The test log loss is: 1.1608803949276791

```

4.1.1.2. Testing the model with best hyper parameters

In [306]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----
```



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----
```



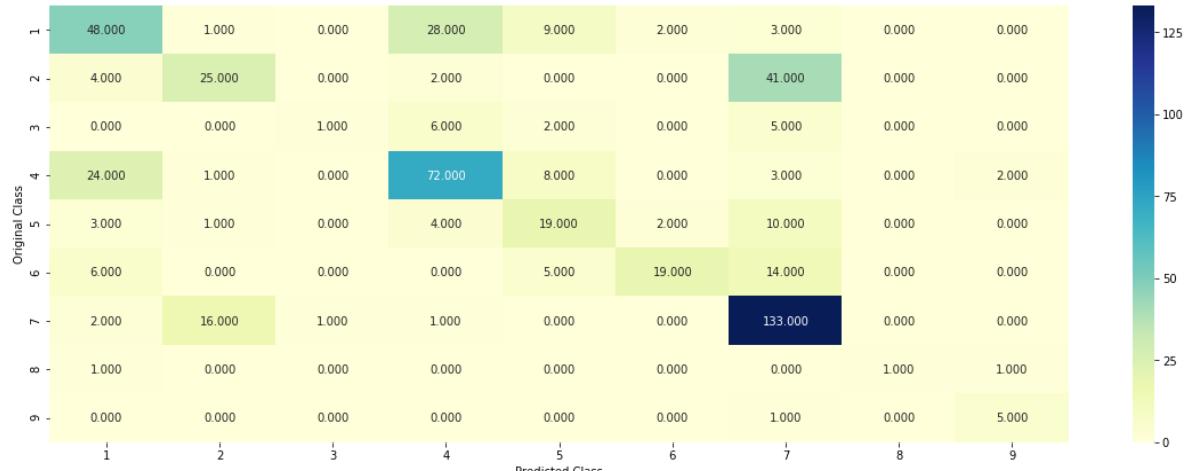
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites 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)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))

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

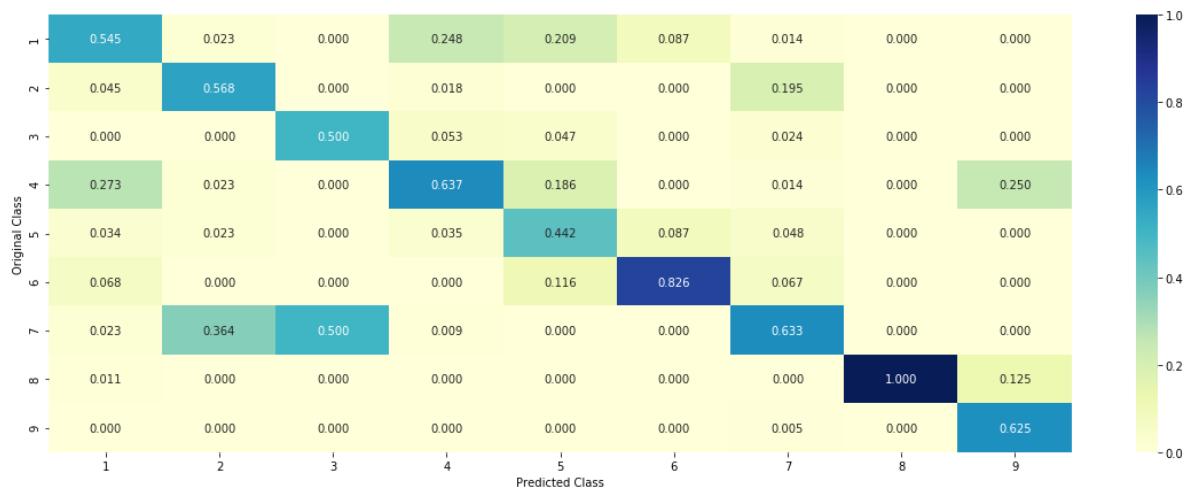
Log Loss : 1.1872495687422477

Number of missclassified point : 0.39285714285714285

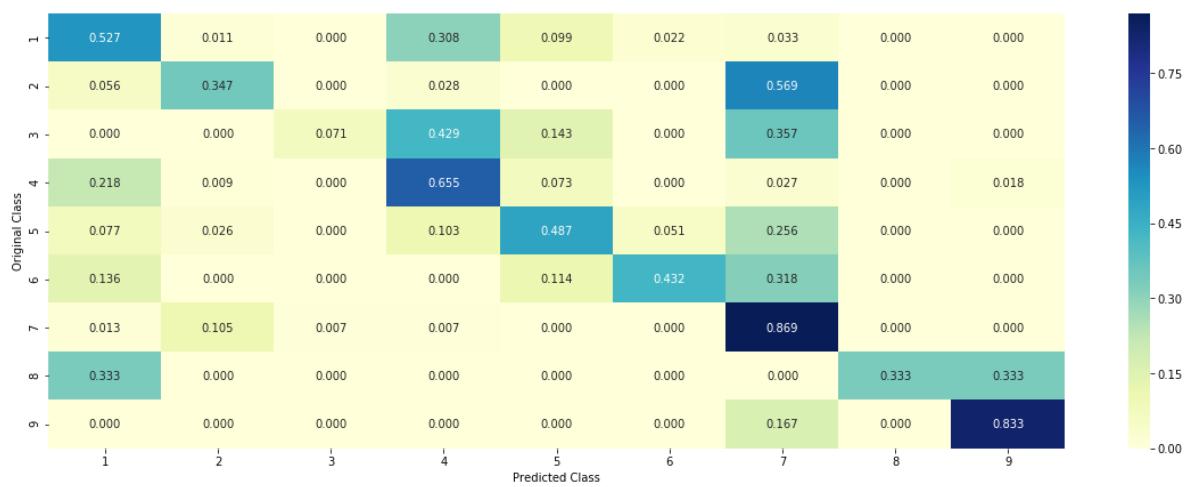
----- Confusion matrix -----



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [307]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index])
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6235 0.043  0.0136 0.1235 0.0473 0.0449
0.0979 0.0029 0.0033]]
Actual Class : 6
```

```
-----  
9 Text feature [one] present in test data point [True]  
12 Text feature [function] present in test data point [True]  
13 Text feature [results] present in test data point [True]  
14 Text feature [protein] present in test data point [True]  
15 Text feature [role] present in test data point [True]  
16 Text feature [loss] present in test data point [True]  
17 Text feature [two] present in test data point [True]  
18 Text feature [also] present in test data point [True]  
19 Text feature [table] present in test data point [True]  
20 Text feature [type] present in test data point [True]  
21 Text feature [region] present in test data point [True]  
22 Text feature [possible] present in test data point [True]  
24 Text feature [however] present in test data point [True]  
25 Text feature [may] present in test data point [True]  
26 Text feature [gene] present in test data point [True]  
27 Text feature [human] present in test data point [True]  
30 Text feature [determined] present in test data point [True]  
31 Text feature [functions] present in test data point [True]  
33 Text feature [binding] present in test data point [True]  
34 Text feature [likely] present in test data point [True]  
35 Text feature [using] present in test data point [True]  
36 Text feature [three] present in test data point [True]  
37 Text feature [including] present in test data point [True]  
38 Text feature [either] present in test data point [True]  
40 Text feature [analysis] present in test data point [True]  
41 Text feature [25] present in test data point [True]  
42 Text feature [whether] present in test data point [True]  
43 Text feature [cancer] present in test data point [True]  
44 Text feature [affect] present in test data point [True]  
45 Text feature [suggest] present in test data point [True]  
46 Text feature [wild] present in test data point [True]  
48 Text feature [well] present in test data point [True]  
49 Text feature [30] present in test data point [True]  
50 Text feature [although] present in test data point [True]  
52 Text feature [important] present in test data point [True]  
54 Text feature [defined] present in test data point [True]  
56 Text feature [discussion] present in test data point [True]  
57 Text feature [least] present in test data point [True]  
58 Text feature [data] present in test data point [True]  
59 Text feature [four] present in test data point [True]  
60 Text feature [large] present in test data point [True]  
61 Text feature [used] present in test data point [True]  
63 Text feature [specific] present in test data point [True]
```

```
64 Text feature [based] present in test data point [True]
65 Text feature [another] present in test data point [True]
66 Text feature [shown] present in test data point [True]
67 Text feature [expression] present in test data point [True]
68 Text feature [observed] present in test data point [True]
69 Text feature [identify] present in test data point [True]
70 Text feature [studies] present in test data point [True]
71 Text feature [performed] present in test data point [True]
72 Text feature [many] present in test data point [True]
73 Text feature [mutations] present in test data point [True]
74 Text feature [previous] present in test data point [True]
75 Text feature [different] present in test data point [True]
76 Text feature [remaining] present in test data point [True]
77 Text feature [dna] present in test data point [True]
78 Text feature [domains] present in test data point [True]
79 Text feature [containing] present in test data point [True]
80 Text feature [first] present in test data point [True]
81 Text feature [genes] present in test data point [True]
82 Text feature [involved] present in test data point [True]
83 Text feature [multiple] present in test data point [True]
85 Text feature [15] present in test data point [True]
86 Text feature [sequence] present in test data point [True]
87 Text feature [whereas] present in test data point [True]
88 Text feature [effect] present in test data point [True]
89 Text feature [directly] present in test data point [True]
90 Text feature [majority] present in test data point [True]
91 Text feature [similar] present in test data point [True]
92 Text feature [essential] present in test data point [True]
93 Text feature [addition] present in test data point [True]
94 Text feature [transcriptional] present in test data point [True]
95 Text feature [several] present in test data point [True]
96 Text feature [highly] present in test data point [True]
97 Text feature [cell] present in test data point [True]
98 Text feature [furthermore] present in test data point [True]
99 Text feature [described] present in test data point [True]
```

Out of the top 100 features 78 are present in query point

4.1.1.4. Feature Importance, Incorrectly classified point

In [309]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0576 0.1227 0.0127 0.0632 0.0386 0.0368
0.6626 0.0027 0.0032]]
Actual Class : 5
```

```
-----
16 Text feature [activation] present in test data point [True]
18 Text feature [activated] present in test data point [True]
19 Text feature [kinase] present in test data point [True]
20 Text feature [cells] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [downstream] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [signaling] present in test data point [True]
26 Text feature [presence] present in test data point [True]
27 Text feature [contrast] present in test data point [True]
28 Text feature [independent] present in test data point [True]
29 Text feature [growth] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [factor] present in test data point [True]
32 Text feature [10] present in test data point [True]
33 Text feature [addition] present in test data point [True]
36 Text feature [treatment] present in test data point [True]
37 Text feature [compared] present in test data point [True]
38 Text feature [however] present in test data point [True]
39 Text feature [sensitive] present in test data point [True]
40 Text feature [shown] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [constitutive] present in test data point [True]
44 Text feature [well] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [increased] present in test data point [True]
47 Text feature [cell] present in test data point [True]
48 Text feature [inhibitors] present in test data point [True]
49 Text feature [mutations] present in test data point [True]
50 Text feature [higher] present in test data point [True]
51 Text feature [activating] present in test data point [True]
52 Text feature [inhibition] present in test data point [True]
53 Text feature [suggest] present in test data point [True]
54 Text feature [tyrosine] present in test data point [True]
55 Text feature [total] present in test data point [True]
56 Text feature [potential] present in test data point [True]
57 Text feature [recently] present in test data point [True]
58 Text feature [activate] present in test data point [True]
59 Text feature [showed] present in test data point [True]
60 Text feature [phosphorylation] present in test data point [True]
61 Text feature [pathways] present in test data point [True]
64 Text feature [found] present in test data point [True]
```

```
65 Text feature [mutant] present in test data point [True]
66 Text feature [may] present in test data point [True]
67 Text feature [absence] present in test data point [True]
68 Text feature [oncogenic] present in test data point [True]
69 Text feature [obtained] present in test data point [True]
70 Text feature [fig] present in test data point [True]
71 Text feature [using] present in test data point [True]
72 Text feature [described] present in test data point [True]
73 Text feature [proliferation] present in test data point [True]
74 Text feature [without] present in test data point [True]
75 Text feature [although] present in test data point [True]
76 Text feature [20] present in test data point [True]
78 Text feature [12] present in test data point [True]
79 Text feature [increase] present in test data point [True]
81 Text feature [results] present in test data point [True]
82 Text feature [3b] present in test data point [True]
83 Text feature [effective] present in test data point [True]
84 Text feature [therapeutic] present in test data point [True]
85 Text feature [mutation] present in test data point [True]
86 Text feature [observed] present in test data point [True]
89 Text feature [including] present in test data point [True]
90 Text feature [phospho] present in test data point [True]
91 Text feature [study] present in test data point [True]
92 Text feature [3a] present in test data point [True]
93 Text feature [identified] present in test data point [True]
94 Text feature [reported] present in test data point [True]
96 Text feature [studies] present in test data point [True]
97 Text feature [expressed] present in test data point [True]
98 Text feature [pathway] present in test data point [True]
99 Text feature [consistent] present in test data point [True]
```

Out of the top 100 features 73 are present in query point

4.2. K Nearest Neighbour Classification

In [310]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/general
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', Leaf_size=30, p=
# 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-nea
#-----

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

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

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

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

```

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_

```

Variables that will be used in the end to make comparison table of all models

```

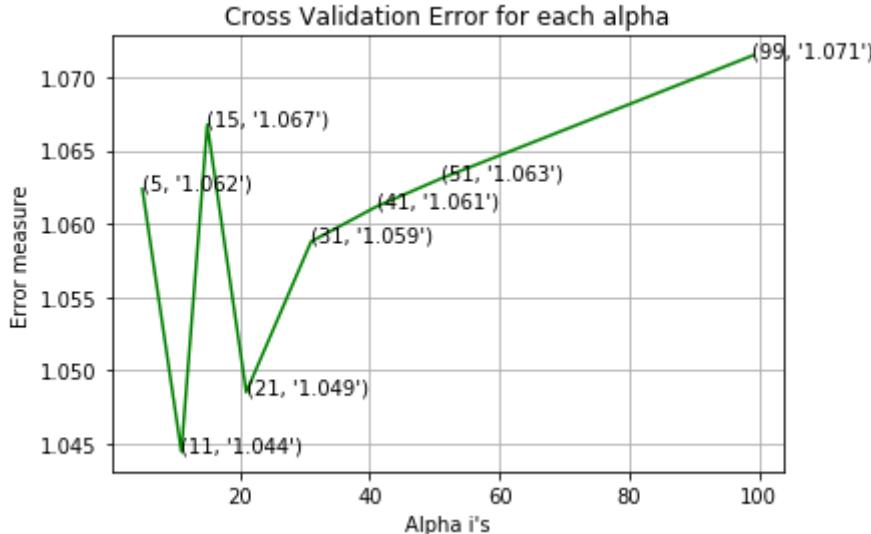
knn_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labels=clf.cla
knn_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.classes_, ep
knn_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=clf.classe

```

```

for alpha = 5
Log Loss : 1.0623805615343493
for alpha = 11
Log Loss : 1.0444452750986575
for alpha = 15
Log Loss : 1.0667705679076356
for alpha = 21
Log Loss : 1.0485047777307632
for alpha = 31
Log Loss : 1.0587909760517968
for alpha = 41
Log Loss : 1.0611629992233091
for alpha = 51
Log Loss : 1.0630889689067622
for alpha = 99
Log Loss : 1.071484468392315

```



```

For values of best alpha = 11 The train log loss is: 0.6312205358693138
For values of best alpha = 11 The cross validation log loss is: 1.044445275
0986575
For values of best alpha = 11 The test log loss is: 1.0135835455369469

```

4.2.2. Testing the model with best hyper parameters

In [311]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/general
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', Leaf_size=30, p=
# 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-nea
# -----
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

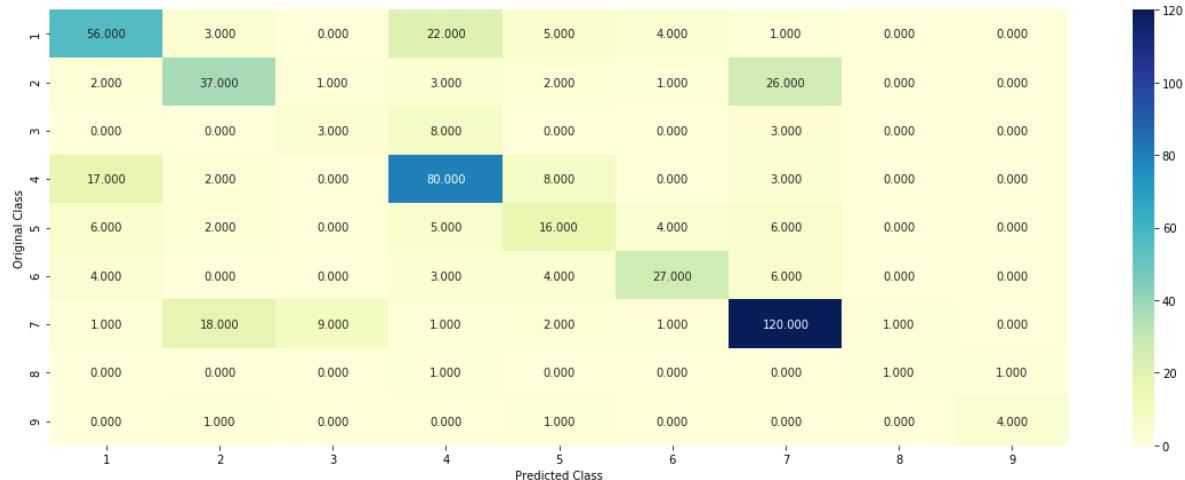
# Variables that will be used in the end to make comparison table of models
knn_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y).sign()))

```

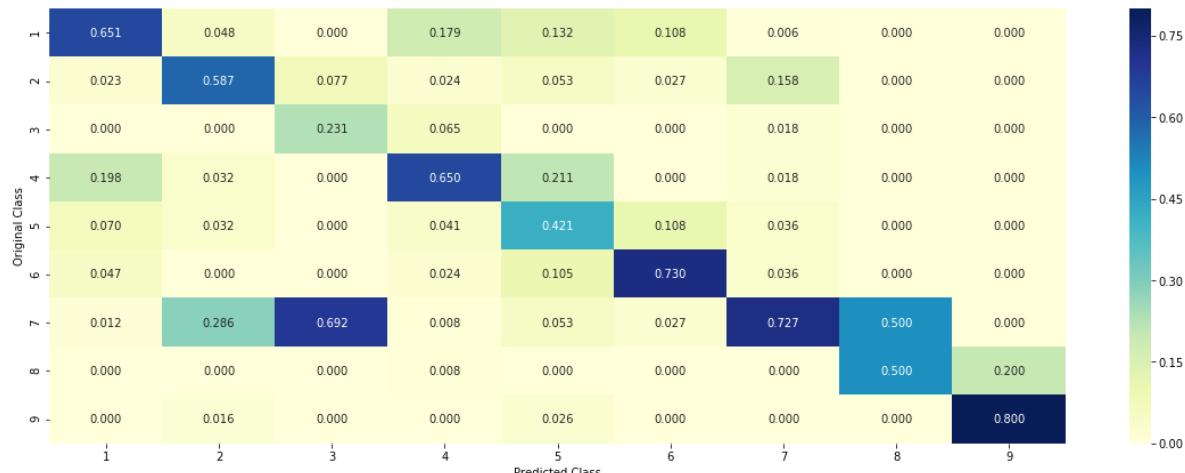
Log loss : 1.0444452750986575

Number of mis-classified points : 0.3533834586466165

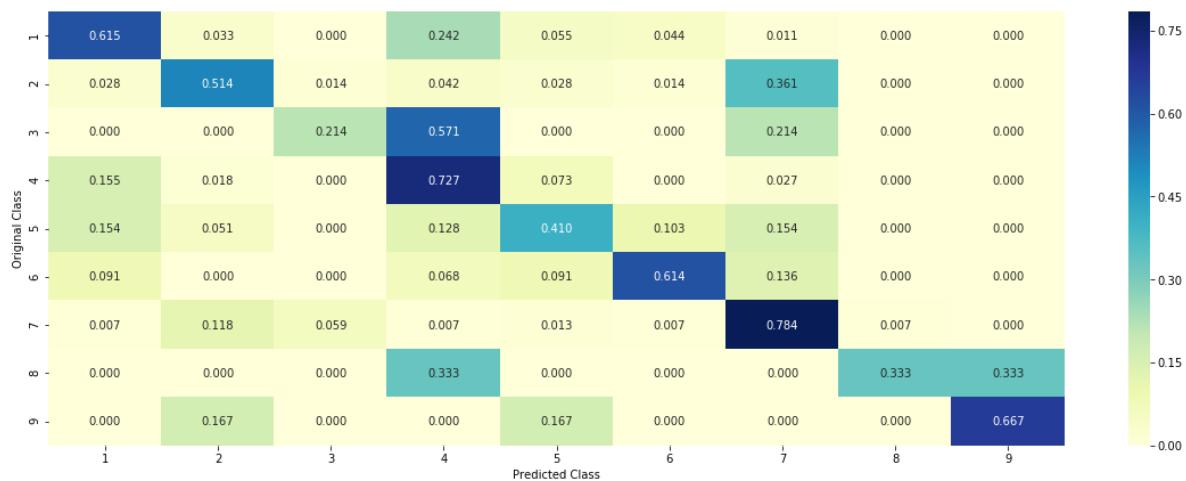
----- Confusion matrix -----



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



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



4.2.3. Sample Query point -1

In [312]:

```
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[be
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes", )
print("Frequency of nearest points : ",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 6
The 11 nearest neighbours of the test points belongs to classes [1 6 6 5 1
1 1 5 1 1 1]
Frequency of nearest points : Counter({1: 7, 6: 2, 5: 2})
```

4.2.4. Sample Query Point-2

In [313]:

```
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[be
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test pc
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]])))
```

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

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper parameter tuning

In [314]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/gen
# -----
# default parameters
# 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_
    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 estimat
    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='l
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_

```

Variables that will be used in the end to make comparison table of all models

```

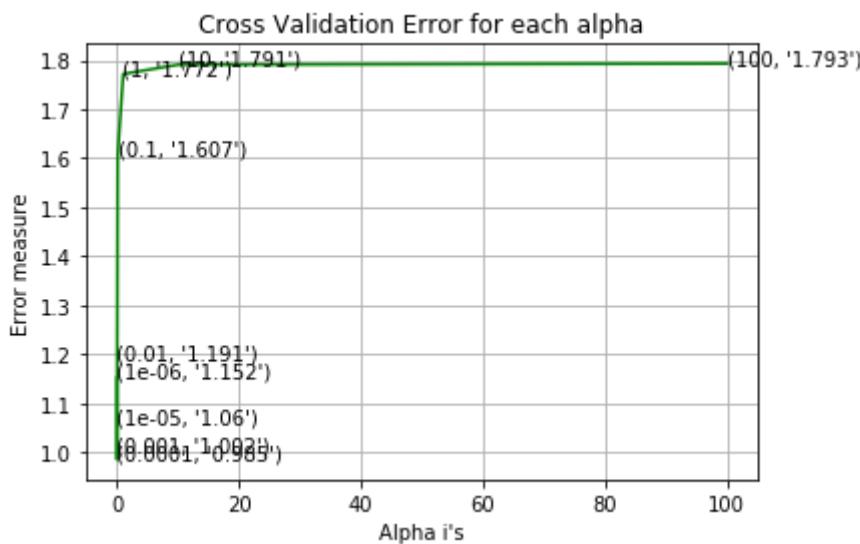
lr_balance_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.
lr_balance_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_
lr_balance_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.cla

```

```

for alpha = 1e-06
Log Loss : 1.152433630179926
for alpha = 1e-05
Log Loss : 1.0596541833098543
for alpha = 0.0001
Log Loss : 0.9853486903200103
for alpha = 0.001
Log Loss : 1.002366657267458
for alpha = 0.01
Log Loss : 1.1912926322909771
for alpha = 0.1
Log Loss : 1.6065582785894972
for alpha = 1
Log Loss : 1.7718690918030942
for alpha = 10
Log Loss : 1.7908288910423278
for alpha = 100
Log Loss : 1.7930810461607123

```



```

For values of best alpha =  0.0001 The train log loss is: 0.4349849887612012
3
For values of best alpha =  0.0001 The cross validation log loss is: 0.98534
86903200103
For values of best alpha =  0.0001 The test log loss is: 0.9833393189312338

```

In [315]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c

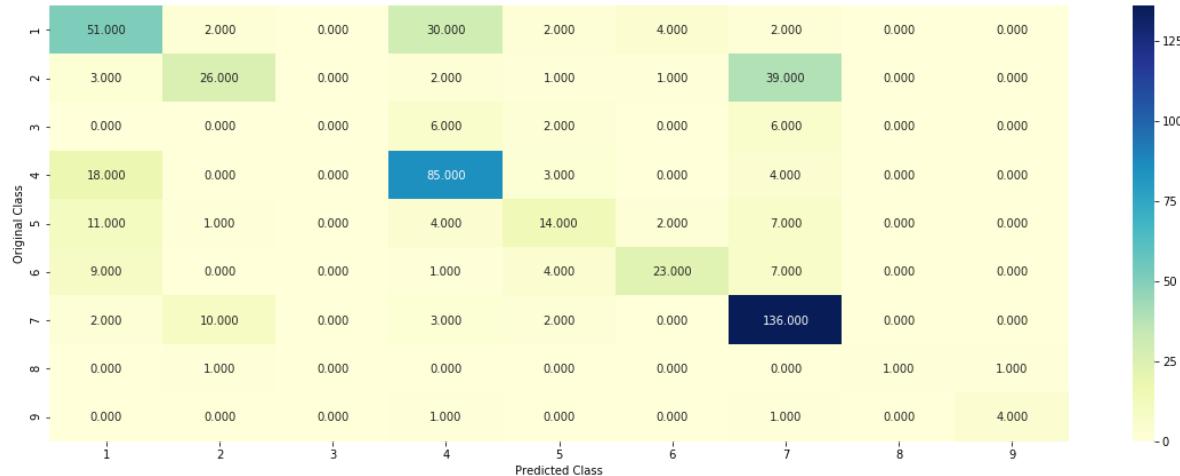
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

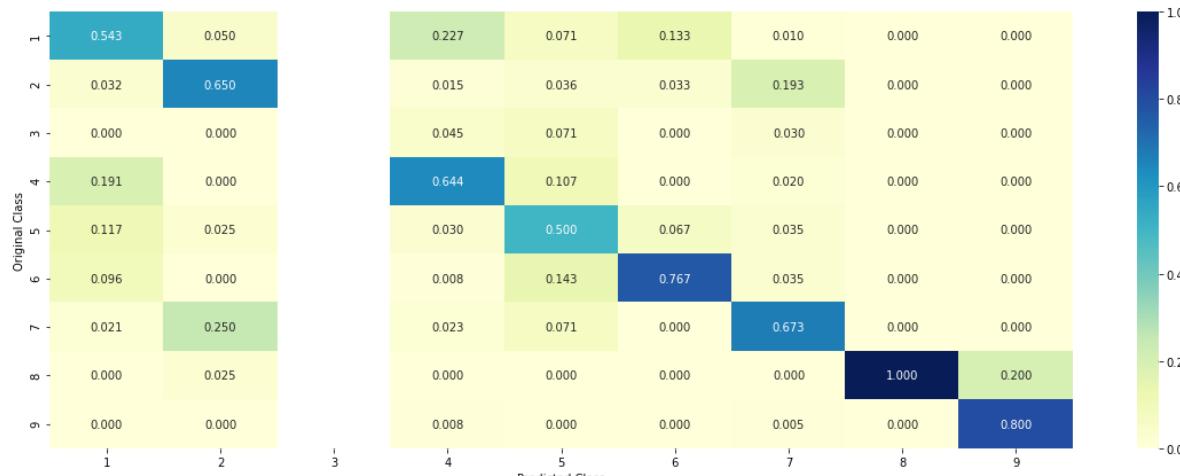
Log loss : 0.9853486903200103

Number of mis-classified points : 0.3609022556390977

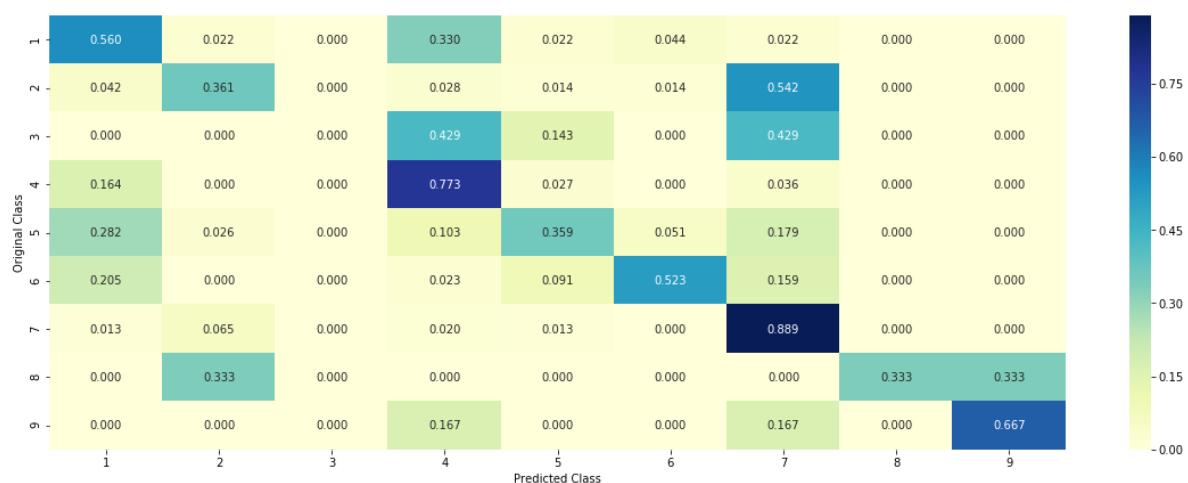
----- Confusion matrix -----



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



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



4.3.1.3. Feature Importance

4.3.1.3.1. Correctly Classified point

In [316]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='')
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[0])
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5314 0.0148 0.0115 0.1014 0.1341 0.1663
0.0315 0.0042 0.0048]]
Actual Class : 6
```

```
-----[REDACTED]-----
39 Text feature [surface] present in test data point [True]
83 Text feature [panel] present in test data point [True]
113 Text feature [hydrophobic] present in test data point [True]
158 Text feature [nucleus] present in test data point [True]
213 Text feature [defined] present in test data point [True]
217 Text feature [region] present in test data point [True]
227 Text feature [mutational] present in test data point [True]
235 Text feature [identify] present in test data point [True]
237 Text feature [function] present in test data point [True]
255 Text feature [21] present in test data point [True]
257 Text feature [structure] present in test data point [True]
275 Text feature [transcriptional] present in test data point [True]
288 Text feature [peptide] present in test data point [True]
289 Text feature [fraction] present in test data point [True]
292 Text feature [individual] present in test data point [True]
294 Text feature [position] present in test data point [True]
296 Text feature [calculated] present in test data point [True]
297 Text feature [molecules] present in test data point [True]
298 Text feature [essential] present in test data point [True]
299 Text feature [page] present in test data point [True]
301 Text feature [possible] present in test data point [True]
304 Text feature [corresponding] present in test data point [True]
306 Text feature [side] present in test data point [True]
310 Text feature [structural] present in test data point [True]
312 Text feature [conserved] present in test data point [True]
320 Text feature [domains] present in test data point [True]
325 Text feature [driven] present in test data point [True]
327 Text feature [functions] present in test data point [True]
333 Text feature [sequencing] present in test data point [True]
334 Text feature [pocket] present in test data point [True]
338 Text feature [construct] present in test data point [True]
340 Text feature [previous] present in test data point [True]
343 Text feature [one] present in test data point [True]
352 Text feature [1997] present in test data point [True]
353 Text feature [greater] present in test data point [True]
354 Text feature [complexes] present in test data point [True]
357 Text feature [colorectal] present in test data point [True]
361 Text feature [affect] present in test data point [True]
362 Text feature [smad3] present in test data point [True]
363 Text feature [gel] present in test data point [True]
```

368 Text feature [crystal] present in test data point [True]
370 Text feature [moreover] present in test data point [True]
373 Text feature [loss] present in test data point [True]
375 Text feature [2001] present in test data point [True]
377 Text feature [table] present in test data point [True]
392 Text feature [revealed] present in test data point [True]
396 Text feature [general] present in test data point [True]
399 Text feature [residues] present in test data point [True]
401 Text feature [breast] present in test data point [True]
403 Text feature [yet] present in test data point [True]
409 Text feature [gst] present in test data point [True]
411 Text feature [hr] present in test data point [True]
413 Text feature [analyzed] present in test data point [True]
415 Text feature [staining] present in test data point [True]
417 Text feature [upon] present in test data point [True]
419 Text feature [signal] present in test data point [True]
422 Text feature [change] present in test data point [True]
423 Text feature [chain] present in test data point [True]
424 Text feature [furthermore] present in test data point [True]
427 Text feature [interestingly] present in test data point [True]
428 Text feature [negative] present in test data point [True]
430 Text feature [another] present in test data point [True]
438 Text feature [remaining] present in test data point [True]
441 Text feature [17] present in test data point [True]
443 Text feature [interactions] present in test data point [True]
445 Text feature [binding] present in test data point [True]
446 Text feature [transcription] present in test data point [True]
448 Text feature [confirmed] present in test data point [True]
449 Text feature [ability] present in test data point [True]
456 Text feature [related] present in test data point [True]
461 Text feature [sequences] present in test data point [True]
465 Text feature [key] present in test data point [True]
468 Text feature [regulatory] present in test data point [True]
470 Text feature [1b] present in test data point [True]
471 Text feature [molecule] present in test data point [True]
473 Text feature [values] present in test data point [True]
474 Text feature [five] present in test data point [True]
477 Text feature [unknown] present in test data point [True]
478 Text feature [assess] present in test data point [True]
483 Text feature [protein] present in test data point [True]
484 Text feature [obtained] present in test data point [True]
485 Text feature [reported] present in test data point [True]
487 Text feature [set] present in test data point [True]
489 Text feature [strongly] present in test data point [True]
490 Text feature [cell] present in test data point [True]
495 Text feature [end] present in test data point [True]
497 Text feature [complex] present in test data point [True]
498 Text feature [criteria] present in test data point [True]

Out of the top 500 features 88 are present in query point

4.3.1.3.2. Incorrectly Classified point

In [318]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[4.490e-02 9.860e-02 1.600e-03 4.620e-02 6.0
40e-02 1.590e-02 7.301e-01
1.900e-03 5.000e-04]]
Actual Class : 5
```

```
-----
10 Text feature [activation] present in test data point [True]
19 Text feature [constitutive] present in test data point [True]
23 Text feature [transformed] present in test data point [True]
26 Text feature [activated] present in test data point [True]
29 Text feature [activate] present in test data point [True]
32 Text feature [oncogene] present in test data point [True]
34 Text feature [downstream] present in test data point [True]
43 Text feature [pathways] present in test data point [True]
50 Text feature [insertion] present in test data point [True]
53 Text feature [codon] present in test data point [True]
54 Text feature [3b] present in test data point [True]
65 Text feature [colony] present in test data point [True]
68 Text feature [presence] present in test data point [True]
81 Text feature [raf] present in test data point [True]
82 Text feature [signaling] present in test data point [True]
86 Text feature [expressing] present in test data point [True]
89 Text feature [derived] present in test data point [True]
98 Text feature [phospho] present in test data point [True]
104 Text feature [factor] present in test data point [True]
114 Text feature [lesions] present in test data point [True]
138 Text feature [s3] present in test data point [True]
146 Text feature [transformation] present in test data point [True]
157 Text feature [leading] present in test data point [True]
161 Text feature [combination] present in test data point [True]
163 Text feature [transforming] present in test data point [True]
165 Text feature [inhibited] present in test data point [True]
168 Text feature [2a] present in test data point [True]
174 Text feature [sensitive] present in test data point [True]
177 Text feature [2b] present in test data point [True]
187 Text feature [gefitinib] present in test data point [True]
190 Text feature [leukemia] present in test data point [True]
210 Text feature [conditions] present in test data point [True]
213 Text feature [effective] present in test data point [True]
229 Text feature [positive] present in test data point [True]
237 Text feature [observations] present in test data point [True]
240 Text feature [advanced] present in test data point [True]
258 Text feature [000] present in test data point [True]
271 Text feature [per] present in test data point [True]
273 Text feature [versus] present in test data point [True]
279 Text feature [erlotinib] present in test data point [True]
282 Text feature [increase] present in test data point [True]
284 Text feature [product] present in test data point [True]
```

```
288 Text feature [promote] present in test data point [True]
304 Text feature [increased] present in test data point [True]
308 Text feature [factors] present in test data point [True]
313 Text feature [3t3] present in test data point [True]
322 Text feature [approximately] present in test data point [True]
326 Text feature [high] present in test data point [True]
333 Text feature [ba] present in test data point [True]
344 Text feature [lead] present in test data point [True]
345 Text feature [her2] present in test data point [True]
348 Text feature [exhibited] present in test data point [True]
349 Text feature [f3] present in test data point [True]
351 Text feature [3a] present in test data point [True]
369 Text feature [tyrosine] present in test data point [True]
388 Text feature [tissues] present in test data point [True]
393 Text feature [fig] present in test data point [True]
400 Text feature [adenocarcinoma] present in test data point [True]
403 Text feature [cyclin] present in test data point [True]
409 Text feature [cdk4] present in test data point [True]
410 Text feature [lung] present in test data point [True]
423 Text feature [oncogenic] present in test data point [True]
425 Text feature [ras] present in test data point [True]
453 Text feature [fold] present in test data point [True]
454 Text feature [promoter] present in test data point [True]
461 Text feature [occur] present in test data point [True]
462 Text feature [total] present in test data point [True]
466 Text feature [standard] present in test data point [True]
470 Text feature [akt] present in test data point [True]
475 Text feature [inhibitor] present in test data point [True]
484 Text feature [addition] present in test data point [True]
490 Text feature [provided] present in test data point [True]
```

Out of the top 500 features 72 are present in query point

4.3.2. Without Class balancing

4.3.2.1. Hyper parameter tuning

In [319]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



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



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

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



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

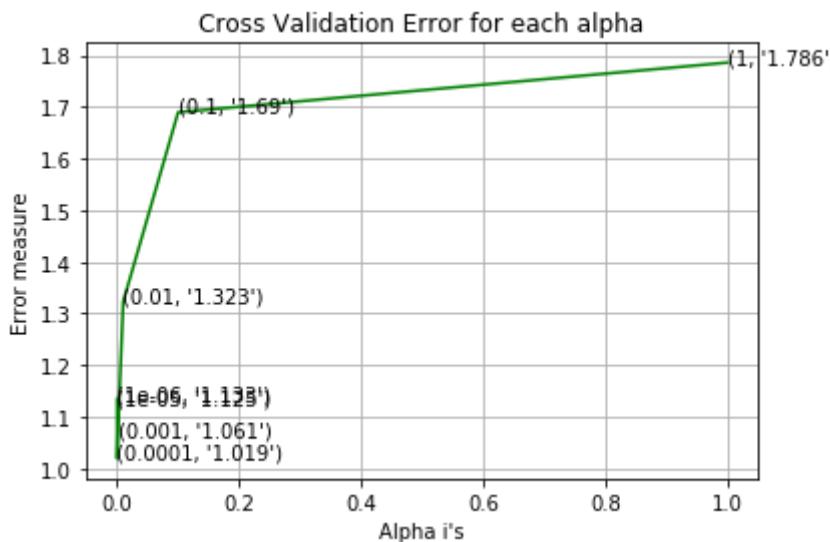
```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 1e-06
Log Loss : 1.1333765814986874
for alpha = 1e-05
Log Loss : 1.125334580204084
for alpha = 0.0001
Log Loss : 1.0185430453197877
for alpha = 0.001
Log Loss : 1.061076983874229
for alpha = 0.01
Log Loss : 1.322970521049101
for alpha = 0.1
Log Loss : 1.6897383787086464
for alpha = 1
Log Loss : 1.786066451634542

```



For values of best alpha = 0.0001 The train log loss is: 0.4284463704490417
 For values of best alpha = 0.0001 The cross validation log loss is: 1.0185430453197877
 For values of best alpha = 0.0001 The test log loss is: 1.0086883905938648

4.3.2.2. Testing model with best hyper parameters

In [320]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# 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, c

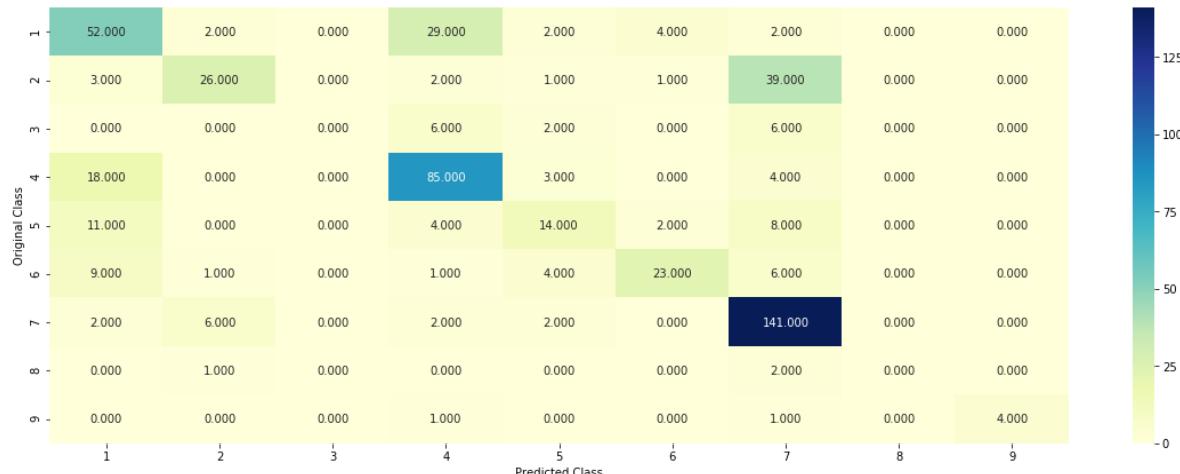
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

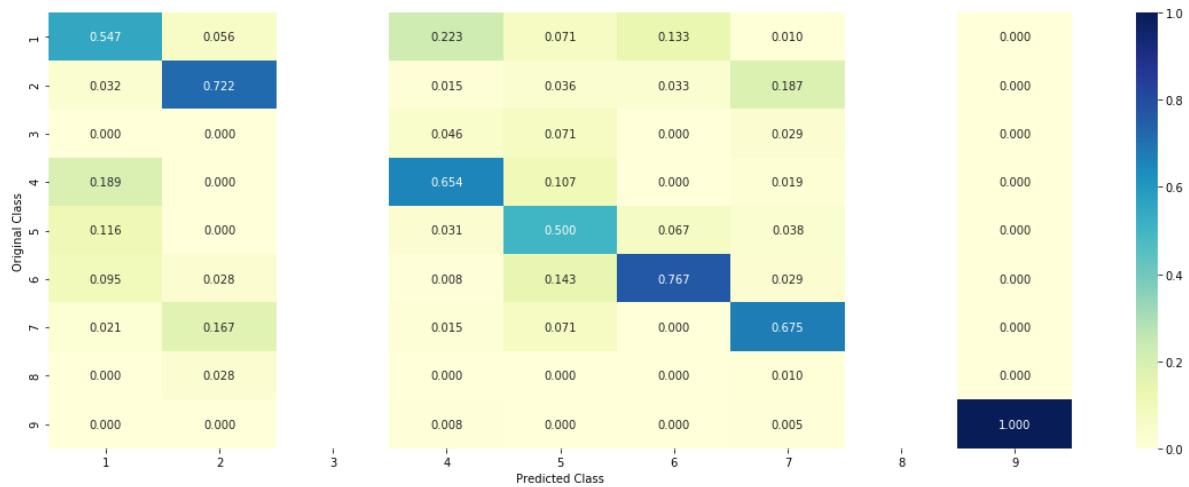
Log loss : 1.0185430453197877

Number of mis-classified points : 0.35150375939849626

----- Confusion matrix -----



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [321]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5416 0.015  0.0089 0.1117 0.1246 0.1569
0.0364 0.0028 0.0021]]
Actual Class : 6
```

```
-----  
56 Text feature [surface] present in test data point [True]  
80 Text feature [panel] present in test data point [True]  
116 Text feature [hydrophobic] present in test data point [True]  
170 Text feature [nucleus] present in test data point [True]  
213 Text feature [21] present in test data point [True]  
220 Text feature [defined] present in test data point [True]  
222 Text feature [region] present in test data point [True]  
242 Text feature [function] present in test data point [True]  
243 Text feature [mutational] present in test data point [True]  
249 Text feature [identify] present in test data point [True]  
270 Text feature [structure] present in test data point [True]  
283 Text feature [fraction] present in test data point [True]  
286 Text feature [molecules] present in test data point [True]  
290 Text feature [peptide] present in test data point [True]  
291 Text feature [transcriptional] present in test data point [True]  
295 Text feature [essential] present in test data point [True]  
296 Text feature [position] present in test data point [True]  
298 Text feature [individual] present in test data point [True]  
299 Text feature [possible] present in test data point [True]  
300 Text feature [corresponding] present in test data point [True]  
302 Text feature [page] present in test data point [True]  
303 Text feature [side] present in test data point [True]  
307 Text feature [calculated] present in test data point [True]  
309 Text feature [conserved] present in test data point [True]  
315 Text feature [previous] present in test data point [True]  
321 Text feature [sequencing] present in test data point [True]  
322 Text feature [functions] present in test data point [True]  
325 Text feature [domains] present in test data point [True]  
329 Text feature [1997] present in test data point [True]  
331 Text feature [greater] present in test data point [True]  
333 Text feature [one] present in test data point [True]  
335 Text feature [driven] present in test data point [True]  
337 Text feature [structural] present in test data point [True]  
339 Text feature [pocket] present in test data point [True]  
342 Text feature [construct] present in test data point [True]  
347 Text feature [colorectal] present in test data point [True]  
351 Text feature [complexes] present in test data point [True]  
355 Text feature [smad3] present in test data point [True]  
359 Text feature [gel] present in test data point [True]  
360 Text feature [2001] present in test data point [True]  
363 Text feature [affect] present in test data point [True]
```

368 Text feature [moreover] present in test data point [True]
372 Text feature [crystal] present in test data point [True]
373 Text feature [table] present in test data point [True]
376 Text feature [loss] present in test data point [True]
378 Text feature [general] present in test data point [True]
379 Text feature [residues] present in test data point [True]
380 Text feature [yet] present in test data point [True]
384 Text feature [interestingly] present in test data point [True]
386 Text feature [revealed] present in test data point [True]
393 Text feature [breast] present in test data point [True]
396 Text feature [analyzed] present in test data point [True]
397 Text feature [signal] present in test data point [True]
403 Text feature [gst] present in test data point [True]
407 Text feature [upon] present in test data point [True]
408 Text feature [staining] present in test data point [True]
409 Text feature [another] present in test data point [True]
410 Text feature [chain] present in test data point [True]
416 Text feature [furthermore] present in test data point [True]
419 Text feature [1b] present in test data point [True]
422 Text feature [related] present in test data point [True]
426 Text feature [hr] present in test data point [True]
429 Text feature [confirmed] present in test data point [True]
436 Text feature [change] present in test data point [True]
438 Text feature [negative] present in test data point [True]
445 Text feature [interactions] present in test data point [True]
448 Text feature [17] present in test data point [True]
453 Text feature [key] present in test data point [True]
455 Text feature [ability] present in test data point [True]
462 Text feature [values] present in test data point [True]
463 Text feature [regulatory] present in test data point [True]
466 Text feature [cell] present in test data point [True]
467 Text feature [binding] present in test data point [True]
471 Text feature [end] present in test data point [True]
472 Text feature [sequences] present in test data point [True]
473 Text feature [effect] present in test data point [True]
474 Text feature [molecule] present in test data point [True]
478 Text feature [transcription] present in test data point [True]
479 Text feature [reported] present in test data point [True]
480 Text feature [assess] present in test data point [True]
481 Text feature [remaining] present in test data point [True]
484 Text feature [obtained] present in test data point [True]
486 Text feature [unknown] present in test data point [True]
492 Text feature [set] present in test data point [True]
494 Text feature [vitro] present in test data point [True]
496 Text feature [strongly] present in test data point [True]
497 Text feature [involved] present in test data point [True]

Out of the top 500 features 87 are present in query point

4.3.2.4. Feature Importance, Inorrectly Classified point

In [322]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[4.200e-02 7.600e-02 8.000e-04 4.310e-02 3.6
30e-02 1.100e-02 7.906e-01
1.000e-04 0.000e+00]]
Actual Class : 5
```

```
-----
18 Text feature [activation] present in test data point [True]
38 Text feature [activate] present in test data point [True]
42 Text feature [activated] present in test data point [True]
43 Text feature [transformed] present in test data point [True]
44 Text feature [downstream] present in test data point [True]
49 Text feature [constitutive] present in test data point [True]
50 Text feature [3b] present in test data point [True]
55 Text feature [codon] present in test data point [True]
60 Text feature [insertion] present in test data point [True]
63 Text feature [oncogene] present in test data point [True]
65 Text feature [pathways] present in test data point [True]
82 Text feature [derived] present in test data point [True]
93 Text feature [presence] present in test data point [True]
101 Text feature [raf] present in test data point [True]
104 Text feature [colony] present in test data point [True]
107 Text feature [factor] present in test data point [True]
129 Text feature [phospho] present in test data point [True]
133 Text feature [signaling] present in test data point [True]
135 Text feature [s3] present in test data point [True]
163 Text feature [expressing] present in test data point [True]
165 Text feature [lesions] present in test data point [True]
206 Text feature [2b] present in test data point [True]
211 Text feature [sensitive] present in test data point [True]
213 Text feature [leading] present in test data point [True]
218 Text feature [inhibited] present in test data point [True]
224 Text feature [transformation] present in test data point [True]
227 Text feature [observations] present in test data point [True]
230 Text feature [2a] present in test data point [True]
235 Text feature [gefitinib] present in test data point [True]
242 Text feature [approximately] present in test data point [True]
245 Text feature [effective] present in test data point [True]
253 Text feature [positive] present in test data point [True]
254 Text feature [combination] present in test data point [True]
255 Text feature [advanced] present in test data point [True]
266 Text feature [leukemia] present in test data point [True]
273 Text feature [conditions] present in test data point [True]
284 Text feature [transforming] present in test data point [True]
291 Text feature [000] present in test data point [True]
301 Text feature [product] present in test data point [True]
309 Text feature [her2] present in test data point [True]
317 Text feature [high] present in test data point [True]
319 Text feature [versus] present in test data point [True]
```

```
326 Text feature [lead] present in test data point [True]
332 Text feature [factors] present in test data point [True]
334 Text feature [increased] present in test data point [True]
336 Text feature [increase] present in test data point [True]
347 Text feature [3a] present in test data point [True]
358 Text feature [fig] present in test data point [True]
364 Text feature [ba] present in test data point [True]
371 Text feature [exhibited] present in test data point [True]
373 Text feature [per] present in test data point [True]
378 Text feature [erlotinib] present in test data point [True]
379 Text feature [lung] present in test data point [True]
384 Text feature [f3] present in test data point [True]
387 Text feature [total] present in test data point [True]
404 Text feature [standard] present in test data point [True]
426 Text feature [cdk4] present in test data point [True]
431 Text feature [adenocarcinoma] present in test data point [True]
449 Text feature [ras] present in test data point [True]
452 Text feature [cyclin] present in test data point [True]
459 Text feature [tissues] present in test data point [True]
465 Text feature [addition] present in test data point [True]
472 Text feature [promoter] present in test data point [True]
478 Text feature [provided] present in test data point [True]
482 Text feature [akt] present in test data point [True]
485 Text feature [occur] present in test data point [True]
486 Text feature [3t3] present in test data point [True]
489 Text feature [fold] present in test data point [True]
491 Text feature [made] present in test data point [True]
493 Text feature [primer] present in test data point [True]
499 Text feature [promote] present in test data point [True]
Out of the top 500 features 71 are present in query point
```

Linear Support Vector Machines

Hyper parameter tuning

In [323]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



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



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

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



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

```

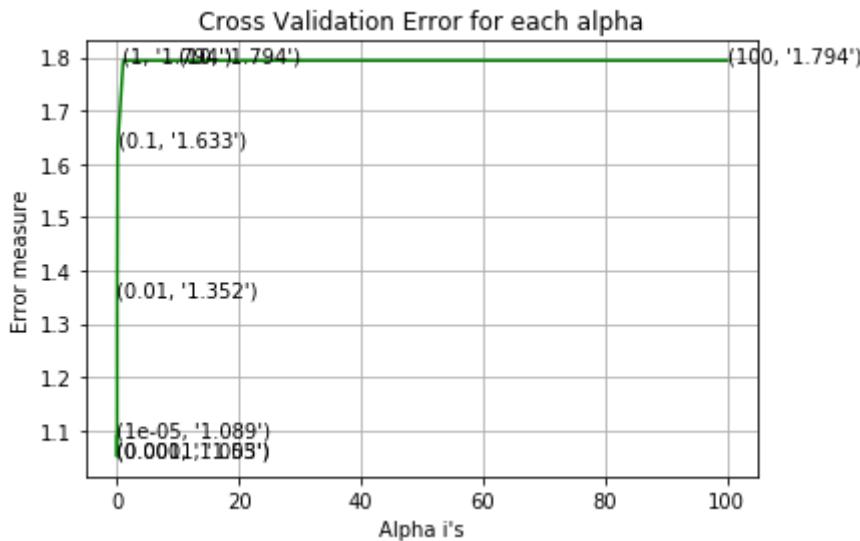
sig_clf.fit(train_x_onehotCoding, train_y)

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_)

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

for C = 1e-05
Log Loss : 1.0887857620080226
for C = 0.0001
Log Loss : 1.0502723113549248
for C = 0.001
Log Loss : 1.05277620601716
for C = 0.01
Log Loss : 1.3522973522117345
for C = 0.1
Log Loss : 1.6332199970062817
for C = 1
Log Loss : 1.793749394883818
for C = 10
Log Loss : 1.7937493064220016
for C = 100
Log Loss : 1.7937495369994643

```



```

For values of best alpha =  0.0001 The train log loss is: 0.4640753668889802
7
For values of best alpha =  0.0001 The cross validation log loss is: 1.05027
23113549248
For values of best alpha =  0.0001 The test log loss is: 1.0456982101442074

```

Testing model with best hyper parameters

In [324]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr')

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



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

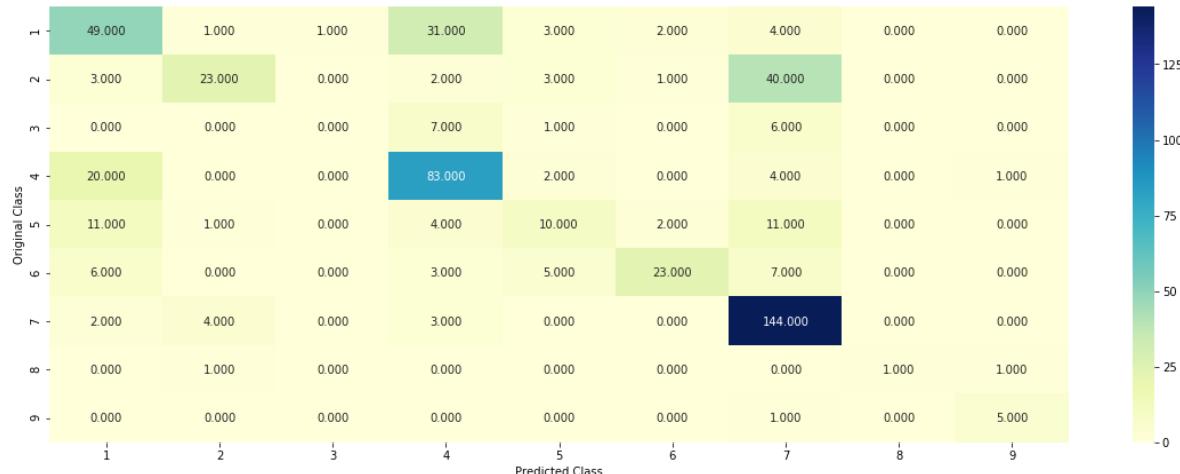
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

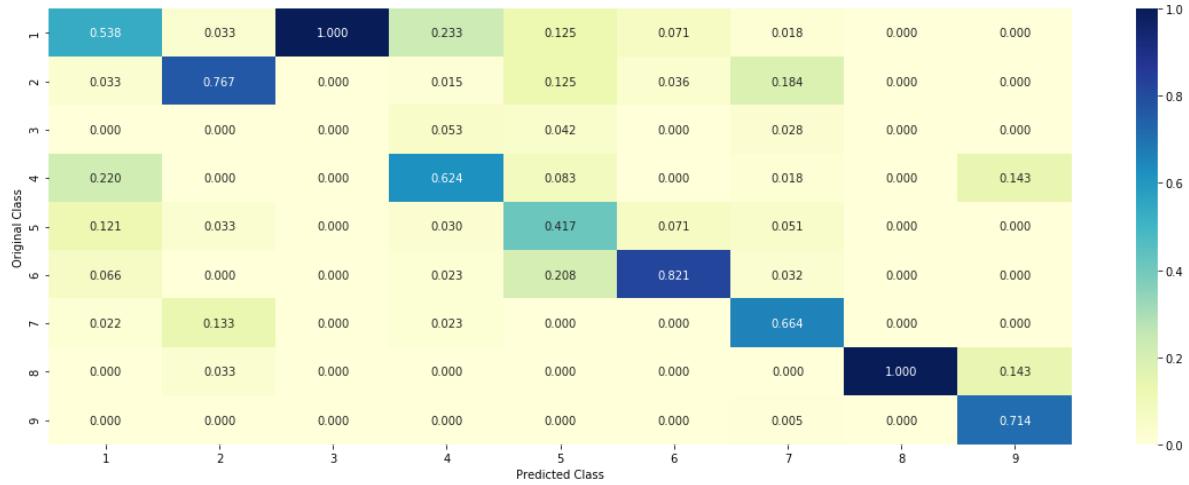
Log loss : 1.0502723113549248

Number of mis-classified points : 0.36466165413533835

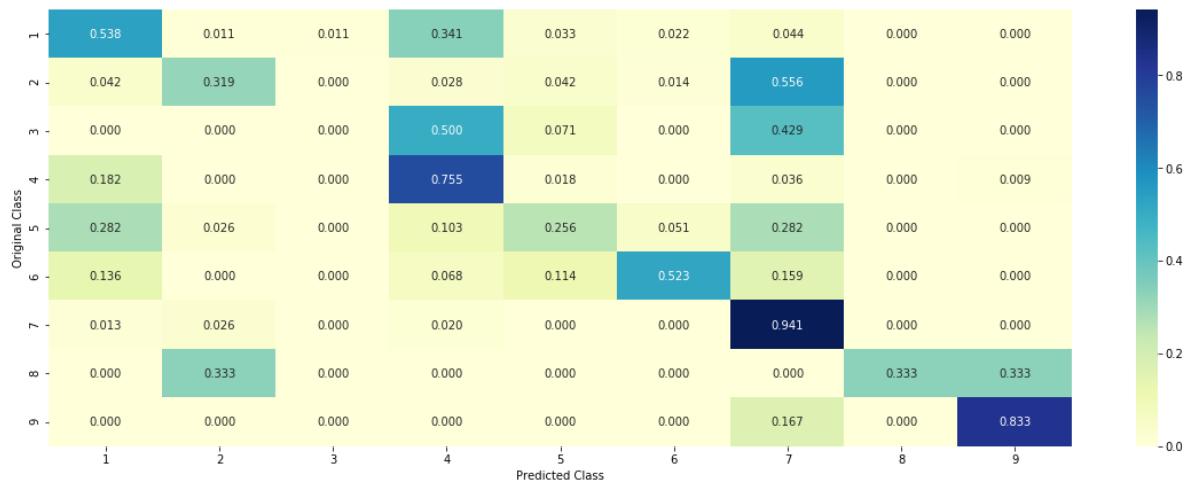
----- Confusion matrix -----



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [325]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4568 0.0758 0.0447 0.0414 0.1238 0.1681
0.0767 0.0049 0.0077]]
Actual Class : 6
```

```
-----[210 Text feature [21] present in test data point [True]
211 Text feature [hydrophobic] present in test data point [True]
213 Text feature [surface] present in test data point [True]
214 Text feature [panel] present in test data point [True]
222 Text feature [mutational] present in test data point [True]
223 Text feature [defined] present in test data point [True]
225 Text feature [molecules] present in test data point [True]
226 Text feature [function] present in test data point [True]
228 Text feature [driven] present in test data point [True]
229 Text feature [identify] present in test data point [True]
232 Text feature [previous] present in test data point [True]
233 Text feature [nucleus] present in test data point [True]
235 Text feature [essential] present in test data point [True]
237 Text feature [position] present in test data point [True]
238 Text feature [individual] present in test data point [True]
242 Text feature [related] present in test data point [True]
244 Text feature [colorectal] present in test data point [True]
245 Text feature [page] present in test data point [True]
246 Text feature [functions] present in test data point [True]
247 Text feature [furthermore] present in test data point [True]
248 Text feature [1997] present in test data point [True]
252 Text feature [region] present in test data point [True]
255 Text feature [transcriptional] present in test data point [True]
258 Text feature [2001] present in test data point [True]
259 Text feature [calculated] present in test data point [True]
260 Text feature [corresponding] present in test data point [True]
360 Text feature [structure] present in test data point [True]
363 Text feature [yet] present in test data point [True]
364 Text feature [peptide] present in test data point [True]
366 Text feature [fraction] present in test data point [True]
367 Text feature [tumor] present in test data point [True]
368 Text feature [staining] present in test data point [True]
372 Text feature [17] present in test data point [True]
373 Text feature [side] present in test data point [True]
374 Text feature [construct] present in test data point [True]
375 Text feature [possible] present in test data point [True]
376 Text feature [structural] present in test data point [True]
377 Text feature [one] present in test data point [True]
380 Text feature [interactions] present in test data point [True]
383 Text feature [smad3] present in test data point [True]
```

386 Text feature [revealed] present in test data point [True]
389 Text feature [1b] present in test data point [True]
391 Text feature [residues] present in test data point [True]
393 Text feature [chain] present in test data point [True]
395 Text feature [moreover] present in test data point [True]
396 Text feature [interestingly] present in test data point [True]
400 Text feature [hr] present in test data point [True]
401 Text feature [table] present in test data point [True]
402 Text feature [indicated] present in test data point [True]
404 Text feature [confirmed] present in test data point [True]
405 Text feature [complexes] present in test data point [True]
407 Text feature [domains] present in test data point [True]
409 Text feature [negative] present in test data point [True]
413 Text feature [ability] present in test data point [True]
414 Text feature [signal] present in test data point [True]
418 Text feature [dependent] present in test data point [True]
421 Text feature [pocket] present in test data point [True]
424 Text feature [loss] present in test data point [True]
425 Text feature [gst] present in test data point [True]
427 Text feature [breast] present in test data point [True]
428 Text feature [showing] present in test data point [True]
430 Text feature [gel] present in test data point [True]
431 Text feature [crystal] present in test data point [True]
433 Text feature [reported] present in test data point [True]
436 Text feature [effect] present in test data point [True]
439 Text feature [upon] present in test data point [True]
444 Text feature [vitro] present in test data point [True]
445 Text feature [contact] present in test data point [True]
448 Text feature [general] present in test data point [True]
449 Text feature [sequences] present in test data point [True]
450 Text feature [interaction] present in test data point [True]
453 Text feature [values] present in test data point [True]
454 Text feature [performed] present in test data point [True]
456 Text feature [strong] present in test data point [True]
457 Text feature [sequencing] present in test data point [True]
464 Text feature [least] present in test data point [True]
465 Text feature [al] present in test data point [True]
466 Text feature [another] present in test data point [True]
467 Text feature [molecule] present in test data point [True]
468 Text feature [et] present in test data point [True]
470 Text feature [conserved] present in test data point [True]
473 Text feature [indeed] present in test data point [True]
475 Text feature [obtained] present in test data point [True]
476 Text feature [second] present in test data point [True]
477 Text feature [carcinomas] present in test data point [True]
479 Text feature [30] present in test data point [True]
480 Text feature [greater] present in test data point [True]
481 Text feature [lines] present in test data point [True]
484 Text feature [assess] present in test data point [True]
485 Text feature [end] present in test data point [True]
490 Text feature [five] present in test data point [True]
499 Text feature [complete] present in test data point [True]

Out of the top 500 features 92 are present in query point

4.3.3.2. For Incorrectly classified point

In [326]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.252e-01 6.320e-02 1.300e-03 8.660e-02 8.0
20e-02 9.900e-03 6.320e-01
6.000e-04 1.000e-03]]
Actual Class : 5
```

```
-----  
33 Text feature [codon] present in test data point [True]  
40 Text feature [3b] present in test data point [True]  
42 Text feature [activate] present in test data point [True]  
43 Text feature [presence] present in test data point [True]  
44 Text feature [derived] present in test data point [True]  
45 Text feature [oncogene] present in test data point [True]  
48 Text feature [activation] present in test data point [True]  
49 Text feature [downstream] present in test data point [True]  
50 Text feature [transformed] present in test data point [True]  
52 Text feature [activated] present in test data point [True]  
229 Text feature [pathways] present in test data point [True]  
231 Text feature [insertion] present in test data point [True]  
235 Text feature [000] present in test data point [True]  
236 Text feature [colony] present in test data point [True]  
237 Text feature [product] present in test data point [True]  
238 Text feature [s3] present in test data point [True]  
239 Text feature [conditions] present in test data point [True]  
240 Text feature [leading] present in test data point [True]  
241 Text feature [raf] present in test data point [True]  
242 Text feature [lead] present in test data point [True]  
243 Text feature [2a] present in test data point [True]  
244 Text feature [2b] present in test data point [True]  
253 Text feature [approximately] present in test data point [True]  
255 Text feature [advanced] present in test data point [True]  
260 Text feature [factor] present in test data point [True]  
265 Text feature [expressing] present in test data point [True]  
266 Text feature [made] present in test data point [True]  
267 Text feature [provided] present in test data point [True]  
268 Text feature [constitutive] present in test data point [True]  
271 Text feature [ras] present in test data point [True]  
274 Text feature [interface] present in test data point [True]  
275 Text feature [sensitive] present in test data point [True]  
277 Text feature [f3] present in test data point [True]  
279 Text feature [ba] present in test data point [True]  
287 Text feature [observations] present in test data point [True]  
289 Text feature [examined] present in test data point [True]  
292 Text feature [confirm] present in test data point [True]  
293 Text feature [total] present in test data point [True]  
294 Text feature [standard] present in test data point [True]  
295 Text feature [positive] present in test data point [True]  
296 Text feature [signaling] present in test data point [True]  
297 Text feature [24] present in test data point [True]
```

```
299 Text feature [mean] present in test data point [True]
301 Text feature [like] present in test data point [True]
303 Text feature [her2] present in test data point [True]
305 Text feature [genomic] present in test data point [True]
307 Text feature [effective] present in test data point [True]
308 Text feature [inhibited] present in test data point [True]
309 Text feature [high] present in test data point [True]
315 Text feature [fig] present in test data point [True]
317 Text feature [addition] present in test data point [True]
318 Text feature [use] present in test data point [True]
321 Text feature [phosphorylated] present in test data point [True]
322 Text feature [exhibited] present in test data point [True]
324 Text feature [promoter] present in test data point [True]
326 Text feature [adenocarcinoma] present in test data point [True]
327 Text feature [3a] present in test data point [True]
328 Text feature [erlotinib] present in test data point [True]
329 Text feature [transforming] present in test data point [True]
330 Text feature [lesions] present in test data point [True]
331 Text feature [factors] present in test data point [True]
332 Text feature [leukemia] present in test data point [True]
333 Text feature [25] present in test data point [True]
334 Text feature [gefitinib] present in test data point [True]
338 Text feature [lung] present in test data point [True]
339 Text feature [egfr] present in test data point [True]
340 Text feature [increase] present in test data point [True]
342 Text feature [recently] present in test data point [True]
345 Text feature [common] present in test data point [True]
347 Text feature [increased] present in test data point [True]
350 Text feature [manner] present in test data point [True]
351 Text feature [formed] present in test data point [True]
352 Text feature [note] present in test data point [True]
353 Text feature [p110] present in test data point [True]
354 Text feature [indicate] present in test data point [True]
355 Text feature [prior] present in test data point [True]
356 Text feature [cyclin] present in test data point [True]
357 Text feature [versus] present in test data point [True]
359 Text feature [led] present in test data point [True]
360 Text feature [43] present in test data point [True]
361 Text feature [consistent] present in test data point [True]
362 Text feature [showed] present in test data point [True]
363 Text feature [extracellular] present in test data point [True]
```

Out of the top 500 features 83 are present in query point

4.5 Random Forest Classifier

4.5.1. Hyper parameter tuning

In [327]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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-9))
        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_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```
'''  
  
best_alpha = np.argmin(cv_log_error_array)  
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c  
clf.fit(train_x_onehotCoding, train_y)  
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig_clf.fit(train_x_onehotCoding, train_y)  
  
predict_y = sig_clf.predict_proba(train_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:")  
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo  
predict_y = sig_clf.predict_proba(test_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",  
  
# Variables that will be used in the end to make comparison table of all models  
rf_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classe  
rf_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1  
rf_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_,  
  
for n_estimators = 100 and max depth = 5  
Log Loss : 1.1935172529257632  
for n_estimators = 100 and max depth = 10  
Log Loss : 1.1962832308278364  
for n_estimators = 200 and max depth = 5  
Log Loss : 1.1709389086511737  
for n_estimators = 200 and max depth = 10  
Log Loss : 1.1959309381890129  
for n_estimators = 500 and max depth = 5  
Log Loss : 1.1673193296531086  
for n_estimators = 500 and max depth = 10  
Log Loss : 1.1925516168401449  
for n_estimators = 1000 and max depth = 5  
Log Loss : 1.1606244843131066  
for n_estimators = 1000 and max depth = 10  
Log Loss : 1.1914953869653797  
for n_estimators = 2000 and max depth = 5  
Log Loss : 1.1588013139682916  
for n_estimators = 2000 and max depth = 10  
Log Loss : 1.1932900940654183  
For values of best estimator = 2000 The train log loss is: 0.84630843111277  
71  
For values of best estimator = 2000 The cross validation log loss is: 1.158  
8013139682916  
For values of best estimator = 2000 The test log loss is: 1.178100572929731  
1
```

4.5.2. Testing model with best hyper parameters

In [328]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----

```

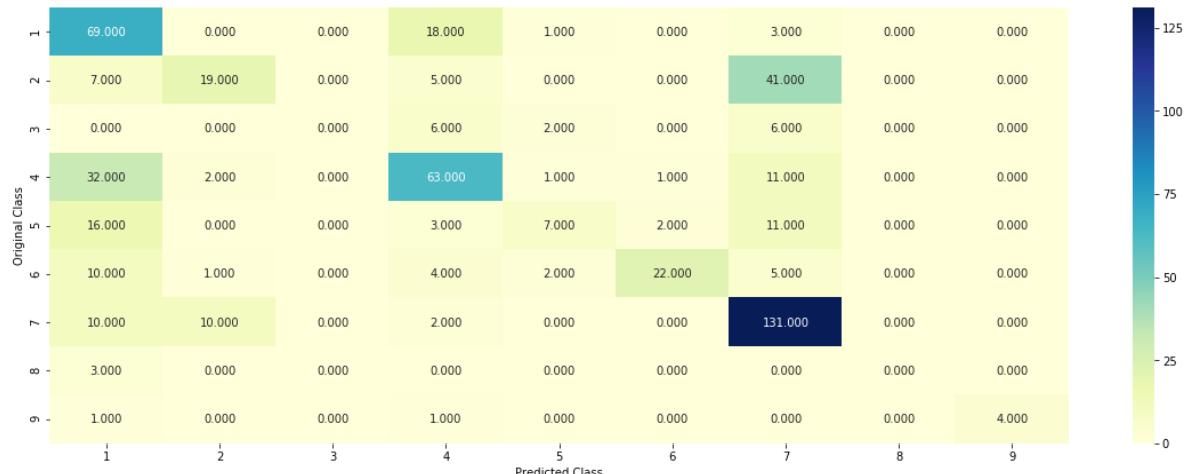
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

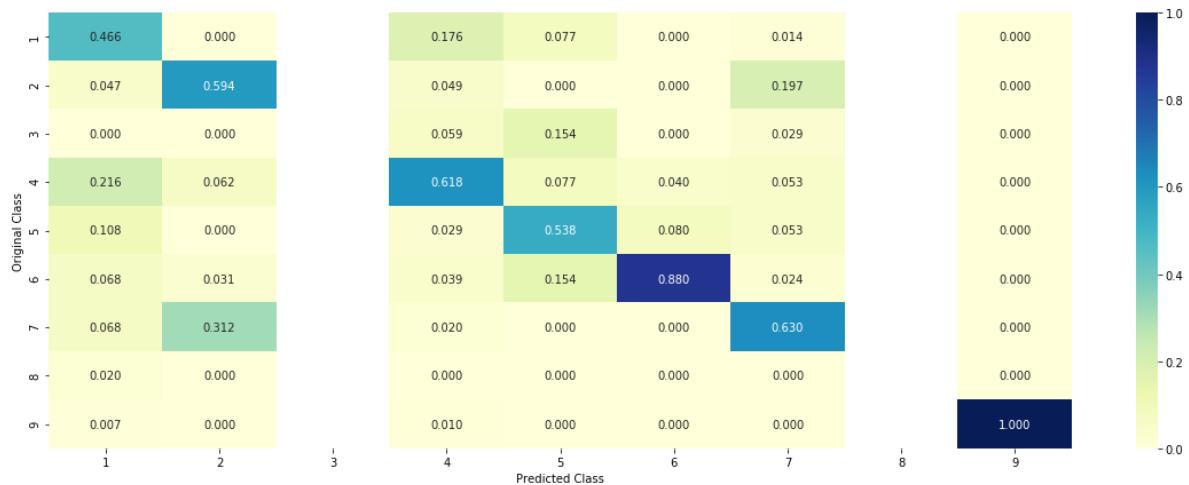
Log loss : 1.1588013139682916

Number of mis-classified points : 0.40789473684210525

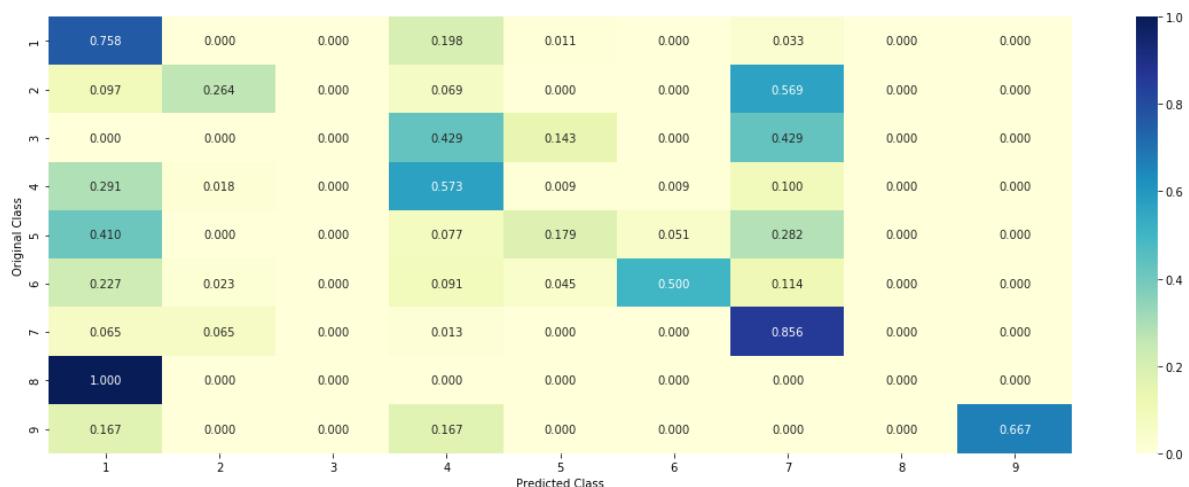
----- Confusion matrix -----



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [329]:

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

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3088 0.0622 0.0214 0.2837 0.0793 0.0979
0.1364 0.0063 0.0039]]
Actual Class : 6
```

```
-----
0 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [phosphorylation] present in test data point [True]
4 Text feature [suppressor] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
7 Text feature [activation] present in test data point [True]
8 Text feature [loss] present in test data point [True]
9 Text feature [activated] present in test data point [True]
11 Text feature [missense] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
15 Text feature [signaling] present in test data point [True]
17 Text feature [protein] present in test data point [True]
21 Text feature [functional] present in test data point [True]
25 Text feature [cell] present in test data point [True]
29 Text feature [cells] present in test data point [True]
31 Text feature [stability] present in test data point [True]
38 Text feature [activate] present in test data point [True]
42 Text feature [functions] present in test data point [True]
45 Text feature [receptor] present in test data point [True]
49 Text feature [inhibited] present in test data point [True]
51 Text feature [expression] present in test data point [True]
55 Text feature [kinases] present in test data point [True]
56 Text feature [proteins] present in test data point [True]
57 Text feature [inhibition] present in test data point [True]
58 Text feature [phospho] present in test data point [True]
60 Text feature [inactivation] present in test data point [True]
71 Text feature [expected] present in test data point [True]
81 Text feature [phosphatase] present in test data point [True]
82 Text feature [response] present in test data point [True]
84 Text feature [phosphorylated] present in test data point [True]
87 Text feature [downstream] present in test data point [True]
92 Text feature [predicted] present in test data point [True]
98 Text feature [conserved] present in test data point [True]
Out of the top 100 features 34 are present in query point
```

4.5.3.2. Incorrectly Classified point

In [330]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1474 0.246  0.0174 0.0417 0.0663 0.0394
0.4298 0.0073 0.0045]]
Actual Class : 5
```

```
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [phosphorylation] present in test data point [True]
4 Text feature [suppressor] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
7 Text feature [activation] present in test data point [True]
8 Text feature [loss] present in test data point [True]
9 Text feature [activated] present in test data point [True]
10 Text feature [inhibitor] present in test data point [True]
11 Text feature [missense] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
14 Text feature [oncogenic] present in test data point [True]
15 Text feature [signaling] present in test data point [True]
16 Text feature [therapy] present in test data point [True]
17 Text feature [protein] present in test data point [True]
18 Text feature [erk] present in test data point [True]
21 Text feature [functional] present in test data point [True]
22 Text feature [pten] present in test data point [True]
24 Text feature [treated] present in test data point [True]
25 Text feature [cell] present in test data point [True]
26 Text feature [akt] present in test data point [True]
27 Text feature [neutral] present in test data point [True]
29 Text feature [cells] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
33 Text feature [classified] present in test data point [True]
34 Text feature [transforming] present in test data point [True]
35 Text feature [ba] present in test data point [True]
37 Text feature [patients] present in test data point [True]
38 Text feature [activate] present in test data point [True]
39 Text feature [repair] present in test data point [True]
41 Text feature [f3] present in test data point [True]
43 Text feature [months] present in test data point [True]
44 Text feature [drug] present in test data point [True]
45 Text feature [receptor] present in test data point [True]
49 Text feature [inhibited] present in test data point [True]
50 Text feature [growth] present in test data point [True]
51 Text feature [expression] present in test data point [True]
52 Text feature [odds] present in test data point [True]
53 Text feature [ic50] present in test data point [True]
55 Text feature [kinases] present in test data point [True]
```

```
56 Text feature [proteins] present in test data point [True]
57 Text feature [inhibition] present in test data point [True]
58 Text feature [phospho] present in test data point [True]
60 Text feature [inactivation] present in test data point [True]
61 Text feature [mek] present in test data point [True]
62 Text feature [3t3] present in test data point [True]
63 Text feature [defective] present in test data point [True]
64 Text feature [sensitivity] present in test data point [True]
65 Text feature [extracellular] present in test data point [True]
66 Text feature [ovarian] present in test data point [True]
67 Text feature [expressing] present in test data point [True]
71 Text feature [expected] present in test data point [True]
73 Text feature [nuclear] present in test data point [True]
76 Text feature [resistance] present in test data point [True]
77 Text feature [use] present in test data point [True]
78 Text feature [damage] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
80 Text feature [advanced] present in test data point [True]
81 Text feature [phosphatase] present in test data point [True]
82 Text feature [response] present in test data point [True]
83 Text feature [clinical] present in test data point [True]
84 Text feature [phosphorylated] present in test data point [True]
85 Text feature [amplification] present in test data point [True]
86 Text feature [egfr] present in test data point [True]
87 Text feature [downstream] present in test data point [True]
89 Text feature [imatinib] present in test data point [True]
91 Text feature [affected] present in test data point [True]
92 Text feature [predicted] present in test data point [True]
94 Text feature [efficacy] present in test data point [True]
96 Text feature [assays] present in test data point [True]
97 Text feature [splice] present in test data point [True]
98 Text feature [conserved] present in test data point [True]
99 Text feature [ras] present in test data point [True]
Out of the top 100 features 76 are present in query point
```

4.5.3. Hyper parameter tuning (With Response Coding)

In [331]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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-9))
        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_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```
'''  
  
best_alpha = np.argmin(cv_log_error_array)  
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c  
clf.fit(train_x_responseCoding, train_y)  
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig_clf.fit(train_x_responseCoding, train_y)  
  
predict_y = sig_clf.predict_proba(train_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log  
predict_y = sig_clf.predict_proba(cv_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log lo  
predict_y = sig_clf.predict_proba(test_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_  
  
# Variables that will be used in the end to make comparison table of all models  
rf_response_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labels=  
rf_response_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.clas  
rf_response_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=cl  
  
for n_estimators = 10 and max depth = 2  
Log Loss : 2.1345591835423474  
for n_estimators = 10 and max depth = 3  
Log Loss : 1.6986595236413289  
for n_estimators = 10 and max depth = 5  
Log Loss : 1.3087805771055592  
for n_estimators = 10 and max depth = 10  
Log Loss : 1.7473129038487052  
for n_estimators = 50 and max depth = 2  
Log Loss : 1.713611972456188  
for n_estimators = 50 and max depth = 3  
Log Loss : 1.348759034314188  
for n_estimators = 50 and max depth = 5  
Log Loss : 1.1631729403859217  
for n_estimators = 50 and max depth = 10  
Log Loss : 1.6206307856002518  
for n_estimators = 100 and max depth = 2  
Log Loss : 1.5474997506240116  
for n_estimators = 100 and max depth = 3  
Log Loss : 1.3671756851043169  
for n_estimators = 100 and max depth = 5  
Log Loss : 1.2096820653557683  
for n_estimators = 100 and max depth = 10  
Log Loss : 1.5865651771717832  
for n_estimators = 200 and max depth = 2  
Log Loss : 1.5744674423408953  
for n_estimators = 200 and max depth = 3  
Log Loss : 1.3170980176753937  
for n_estimators = 200 and max depth = 5  
Log Loss : 1.2212622834758893  
for n_estimators = 200 and max depth = 10  
Log Loss : 1.5666903796057539  
for n_estimators = 500 and max depth = 2  
Log Loss : 1.5816470808142502  
for n_estimators = 500 and max depth = 3  
Log Loss : 1.4119029712913718  
for n_estimators = 500 and max depth = 5  
Log Loss : 1.2599288508644926  
for n_estimators = 500 and max depth = 10  
Log Loss : 1.6016271902270371
```

```
for n_estimators = 1000 and max depth = 2
Log Loss : 1.554638732072944
for n_estimators = 1000 and max depth = 3
Log Loss : 1.4105956670693958
for n_estimators = 1000 and max depth = 5
Log Loss : 1.260818064638381
for n_estimators = 1000 and max depth = 10
Log Loss : 1.5729734765100511
For values of best alpha = 50 The train log loss is: 0.06676936890896665
For values of best alpha = 50 The cross validation log loss is: 1.163172940
3859215
For values of best alpha = 50 The test log loss is: 1.197920364468122
```

Testing model with best hyper parameters (Response Coding)

In [332]:

```
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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/rando
# -----
```

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha%4)])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y)

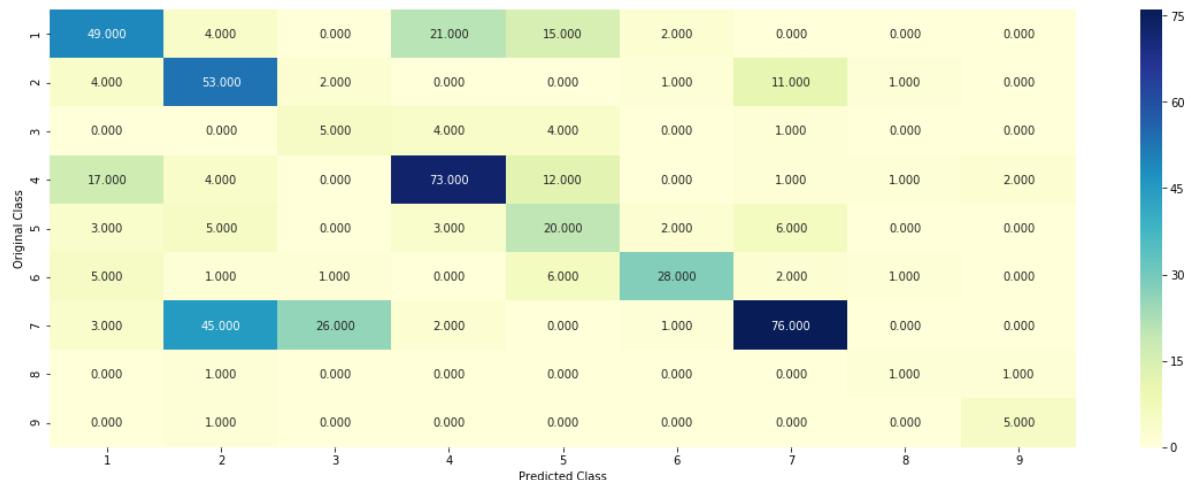
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

Variables that will be used in the end to make comparison table of models
rf_response_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding) - cv_y))

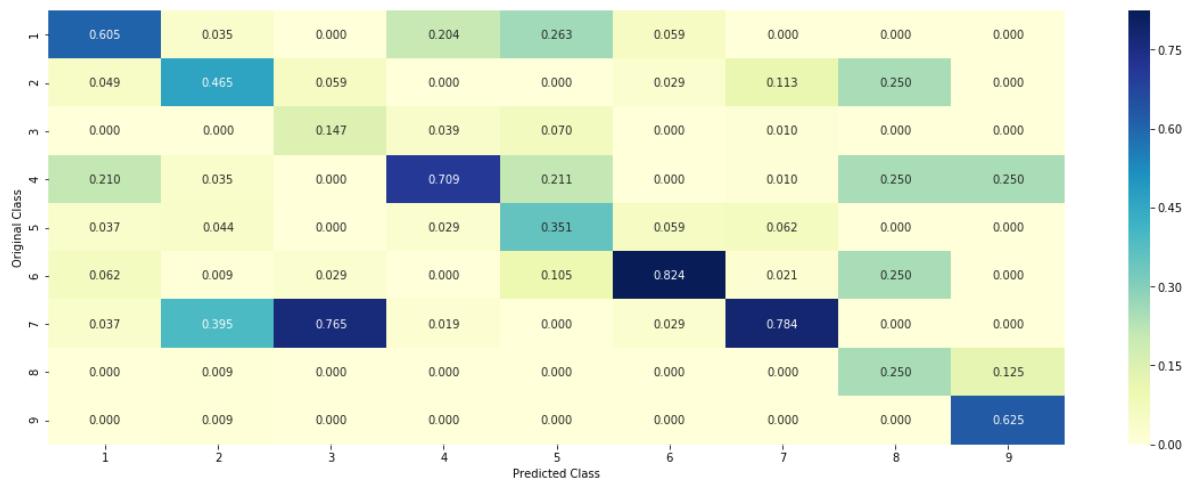
Log loss : 1.1631729403859217

Number of mis-classified points : 0.41729323308270677

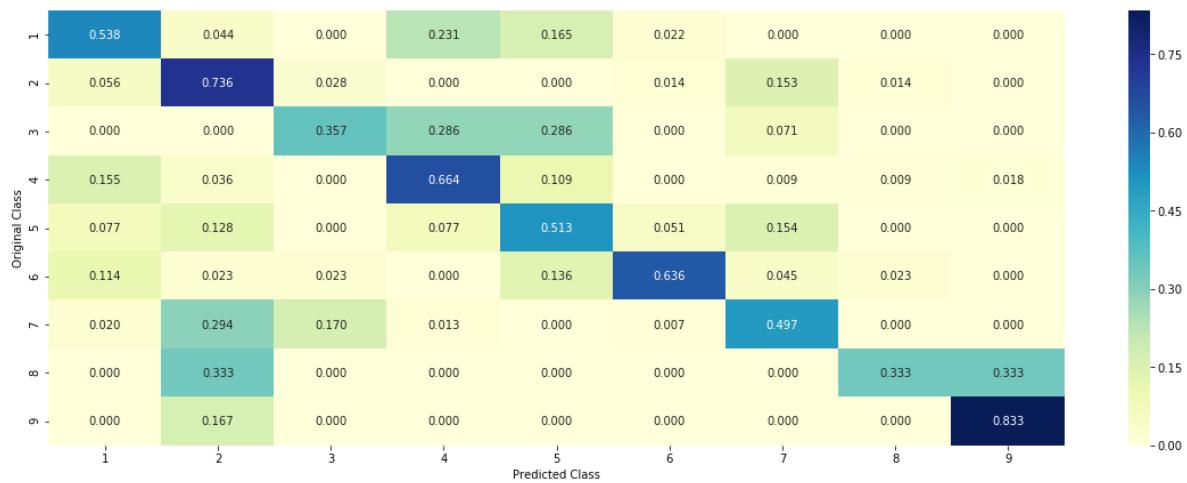
----- Confusion matrix -----



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [333]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCodir
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 : 1
Predicted Class Probabilities: [[0.3371 0.0094 0.0903 0.069  0.2148 0.2541
0.0043 0.0084 0.0126]]
Actual Class : 6
```

```
-----
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
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
```

4.5.5.2. Incorrectly Classified point

In [334]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0238 0.4701 0.0882 0.0295 0.0351 0.0694
0.2534 0.0198 0.0107]]
Actual Class : 5
-----
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
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
```

Stack the models

4.7.1 testing with hyper parameter tuning

In [335]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



```
# read more about support vector machines with Linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=F
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='o

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



```
# read more about support vector machines with Linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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
# -----
```



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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_s
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")
```

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

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

```
Logistic Regression : Log Loss: 1.00
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.19
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.028
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.493
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.169
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.390
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.879
```

4.7.2 testing the model with the best hyper parameters

In [336]:

```

lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, n_jobs=-1)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

# Variables that will be used in the end to make comparison table of all models
stack_train = log_error
stack_cv = log_error1
stack_test = log_error2
stack_missclassified = (np.count_nonzero((sclf.predict(test_x_onehotCoding) - test_y)) / test_y)

```

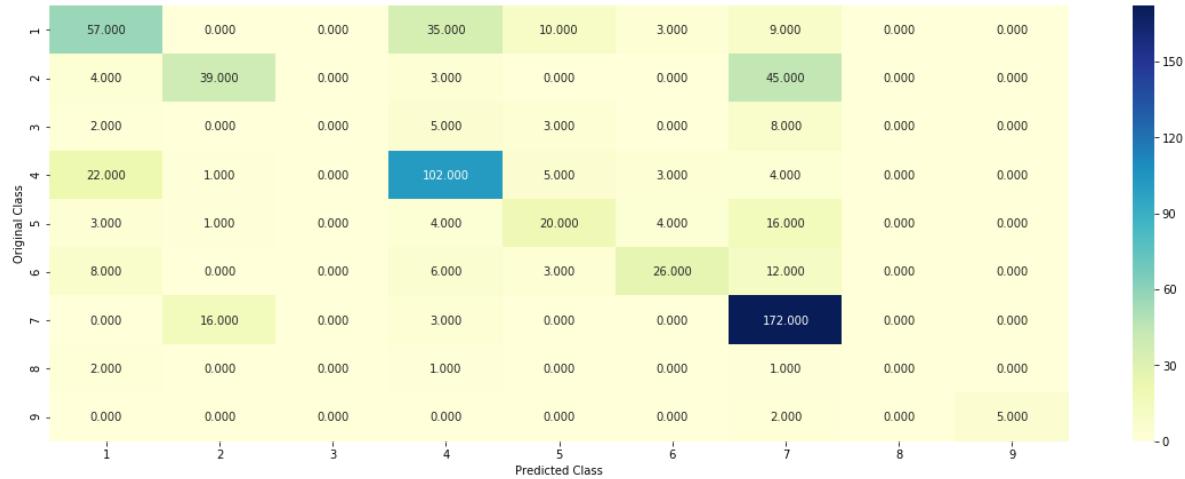
Log loss (train) on the stacking classifier : 0.5343871367603206

Log loss (CV) on the stacking classifier : 0.5343871367603206

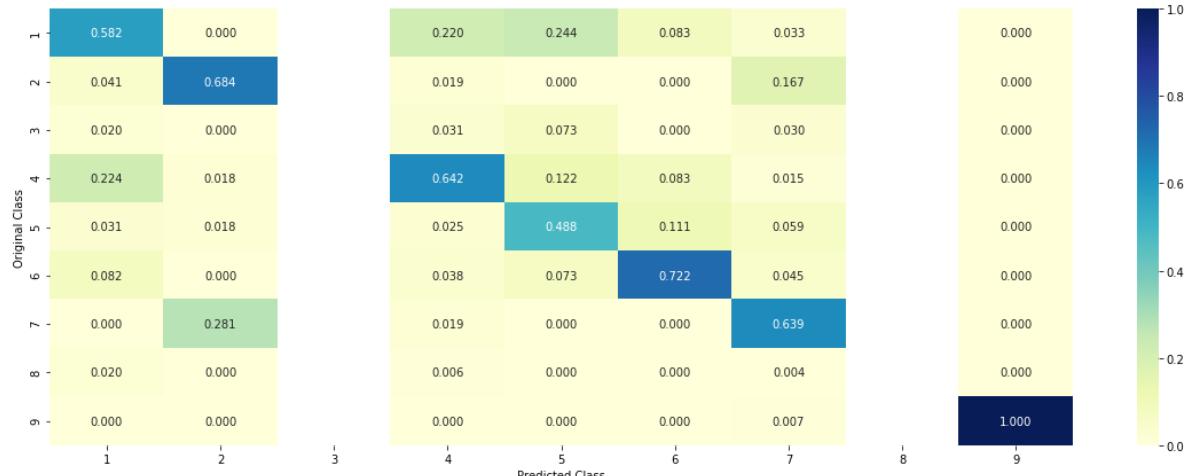
Log loss (test) on the stacking classifier : 0.5343871367603206

Number of missclassified point : 0.3669172932330827

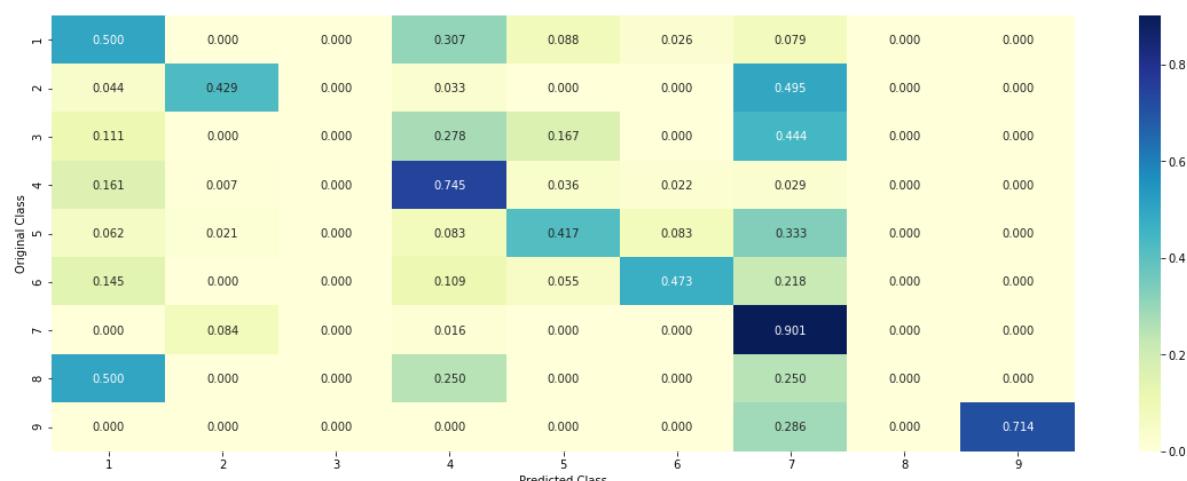
----- Confusion matrix -----



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



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



4.7.3 Maximum Voting classifier

In [337]:

```
#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)], 
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier : ", log_loss(train_y, vclf.predict_proba(tr
print("Log loss (CV) on the VotingClassifier : ", log_loss(cv_y, vclf.predict_proba(cv_x_one
print("Log loss (test) on the VotingClassifier : ", log_loss(test_y, vclf.predict_proba(test
print("Number of missclassified point : ", np.count_nonzero((vclf.predict(test_x_onehotCodir
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))

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

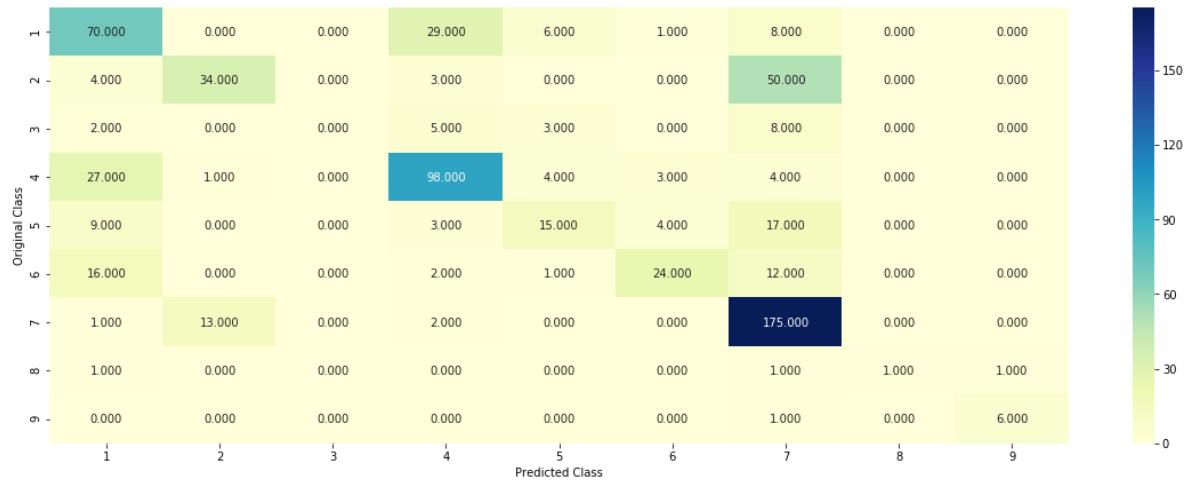
Log loss (train) on the VotingClassifier : 0.8230766457755163

Log loss (CV) on the VotingClassifier : 1.1683190460408739

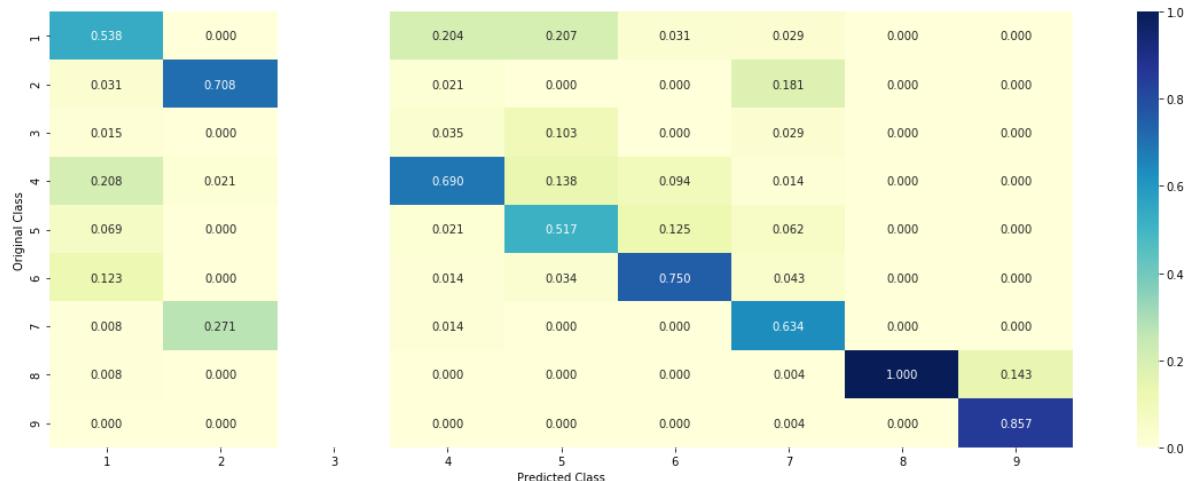
Log loss (test) on the VotingClassifier : 1.1679973985930663

Number of missclassified point : 0.36390977443609024

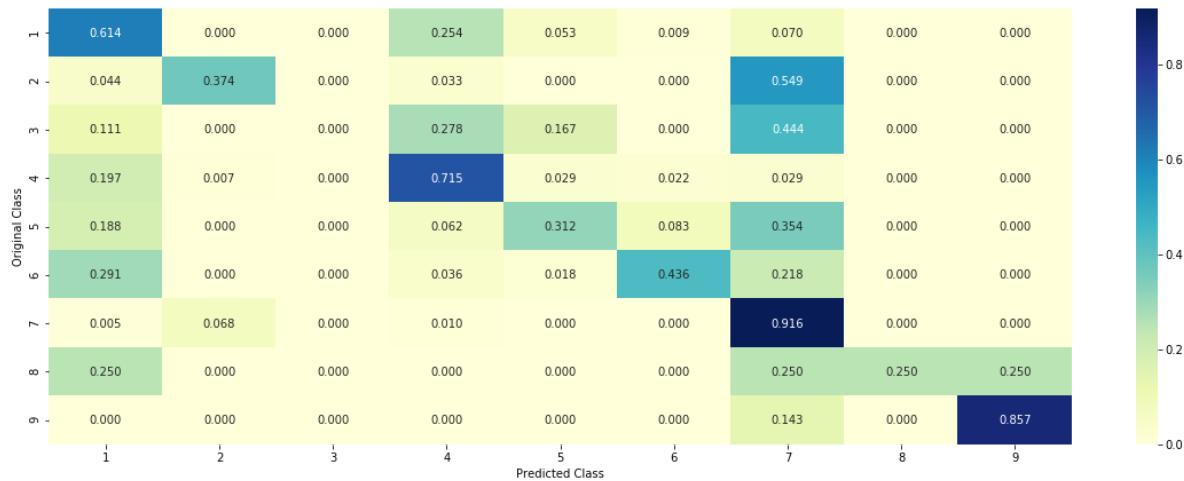
----- Confusion matrix -----



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



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



CONCLUSION

(a). Procedure Followed :-

STEP 1:- Replace `TfidfVectorizer()` by `TfidfVectorizer(max_features=1000)` in the one hot encoding section of text feature to get top 1000 words based of tf-idf values.

(b).Table (Model Performances):-

In [91]:

```
# Creating table using PrettyTable library

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

# Training Loss
train_loss = [0.507287, 0.6312205, 0.434984 ,0.42844637, 0.4640753, 0.846308, 0.06676936, 0.5343

# Cross Validation loss
cv_loss = [1.1872495, 1.04444527, 0.98534869, 1.0185430, 1.05027231, 1.1588013, 1.1631729, 0.53438
# Test Loss
test_loss = [1.1608803, 1.0135835, 0.983339, 1.0086883, 1.04569821, 1.17810057, 1.197920, 0.53438

alpha=[0.001,11,.001,.0001,2000,50,.1,'NaN']

# Percentage Misclassified points
misclassified = [0.3928571, 0.353383, 0.36090226, 0.351503759, 0.36466165, 0.40789473, 0.41729

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

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
s	Test_loss	Misclassified(%)		
95	Naive Bayes	0.001	0.507287	1.18724
527	K-Nearest Neighbour	11	0.6312205	1.04444
869	LR With Class Balancing	0.001	0.434984	0.98534
43	LR Without Class Balancing	0.0001	0.42844637	1.0185
231	Linear SVM	0.0001	0.4640753	1.05027
13	RF With One hot Encoding	2000	0.846308	1.15880
7	RF With Response Coding	50	0.06676936	1.16317

29		1.19792		0.417293				
	8		Stacking Classifier			0.1		0.5343871 0.53438
71		0.5343871		0.366917293233				
	9		Maximum Voting Classifier			NaN		0.8230766 1.16831
904		1.167997398		0.363909774				
<hr/>								

-----TASK3-----

Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [342]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
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 [343]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [344]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of words)
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 occurred
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 : 2363094

In [345]:

```

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

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

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

```

In [346]:

```

# 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 [347]:

```

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

```

In [348]:

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

In [341]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(ngram_range=(1,2))
    df_textfea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_textfea_counts = df_textfea.sum(axis=0).A1
    df_textfea_dict = dict(zip(list(df_text_features),df_textfea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [349]:

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

73.529 % of word of test data appeared in train data
74.447 % of word of Cross Validation appeared in train data

In [350]:

```

def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer(ngram_range=(1,2))
    var_count_vec = CountVectorizer(ngram_range=(1,2))
    text_count_vec = CountVectorizer(ngram_range=(1,2))

    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,ye
    elif (v < fea1_len+fea2_len):
        word = var_vec.get_feature_names()[v-(fea1_len)]
        yes_no = True if word == var else False
        if yes_no:
            word_present += 1
            print(i, "variation feature [{}] present in test data point [{}].format(wc
    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,ye

print("Out of the top ",no_features," features ", word_present, "are present in query p

```

Logistic Regression

. With Class balancing

Hyper parameter tuning

In [351]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/gen
# -----
# default parameters
# 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_
    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 estimat
    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='l
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_

```

Variables that will be used in the end to make comparison table of all models

```

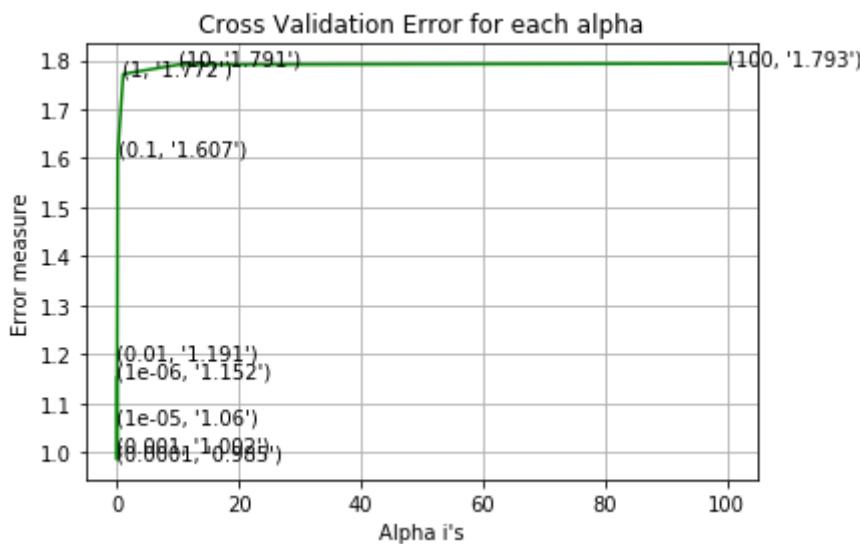
lr_balance_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.
lr_balance_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_
lr_balance_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.cla

```

```

for alpha = 1e-06
Log Loss : 1.152433630179926
for alpha = 1e-05
Log Loss : 1.0596541833098543
for alpha = 0.0001
Log Loss : 0.9853486903200103
for alpha = 0.001
Log Loss : 1.002366657267458
for alpha = 0.01
Log Loss : 1.1912926322909771
for alpha = 0.1
Log Loss : 1.6065582785894972
for alpha = 1
Log Loss : 1.7718690918030942
for alpha = 10
Log Loss : 1.7908288910423278
for alpha = 100
Log Loss : 1.7930810461607123

```



```

For values of best alpha =  0.0001 The train log loss is: 0.4349849887612012
3
For values of best alpha =  0.0001 The cross validation log loss is: 0.98534
86903200103
For values of best alpha =  0.0001 The test log loss is: 0.9833393189312338

```

4.1.1.2. Testing the model with best hyper parameters

In [352]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c

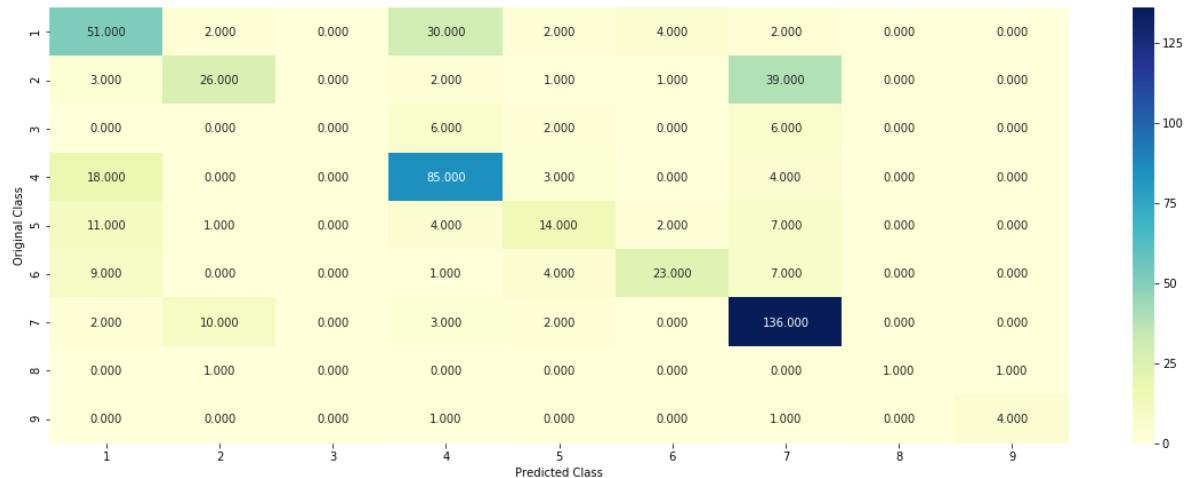
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

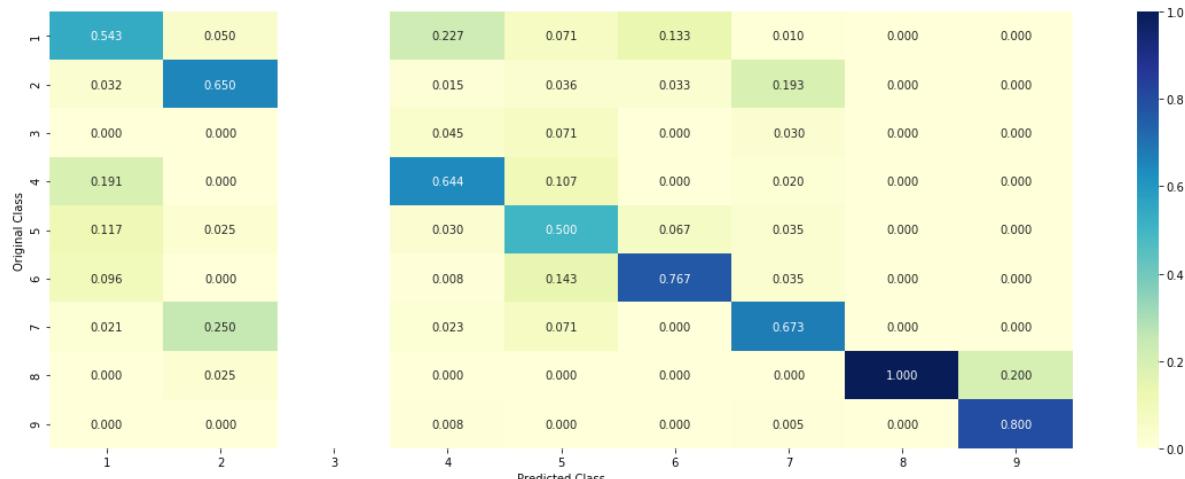
Log loss : 0.9853486903200103

Number of mis-classified points : 0.3609022556390977

----- Confusion matrix -----



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



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



4.1.2. Without Class balancing

4.1.2.1. Hyper parameter tuning

In [162]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



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



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

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



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

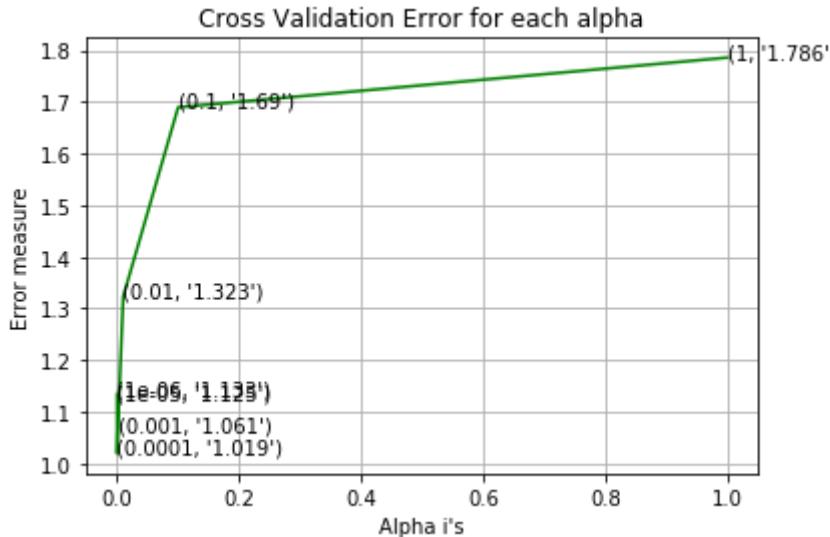
```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 1e-06
Log Loss : 1.1333765814986874
for alpha = 1e-05
Log Loss : 1.125334580204084
for alpha = 0.0001
Log Loss : 1.0185430453197877
for alpha = 0.001
Log Loss : 1.061076983874229
for alpha = 0.01
Log Loss : 1.322970521049101
for alpha = 0.1
Log Loss : 1.6897383787086464
for alpha = 1
Log Loss : 1.786066451634542

```



For values of best alpha = 0.0001 The train log loss is: 0.4284463704490417
 For values of best alpha = 0.0001 The cross validation log loss is: 1.01854
 30453197877
 For values of best alpha = 0.0001 The test log loss is: 1.0086883905938648

4.1.2.2. Testing model with best hyper parameters

In [353]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# 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, c

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

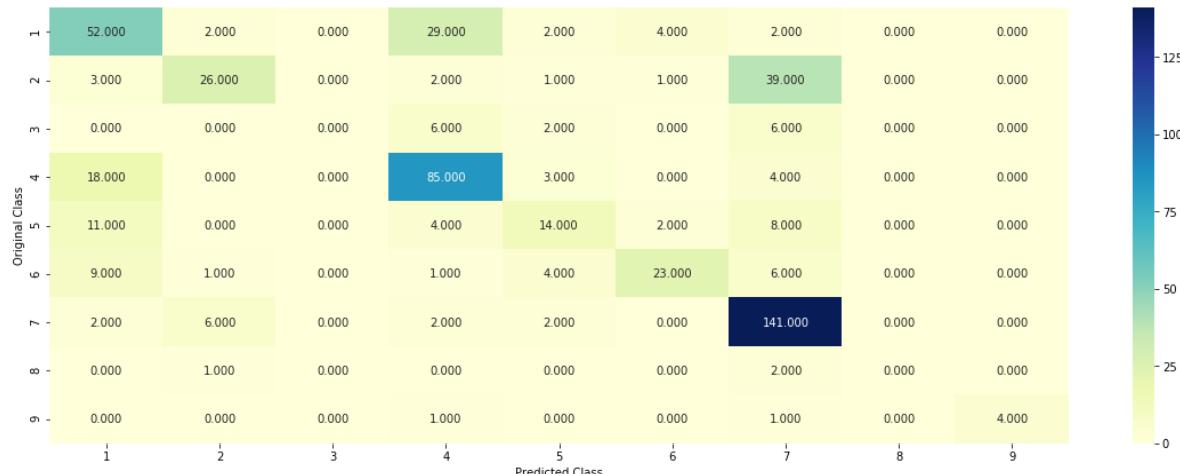
# Variables that will be used in the end to make comparison table of models
lr_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding) - cv_y))/cv_y.shape[0])

```

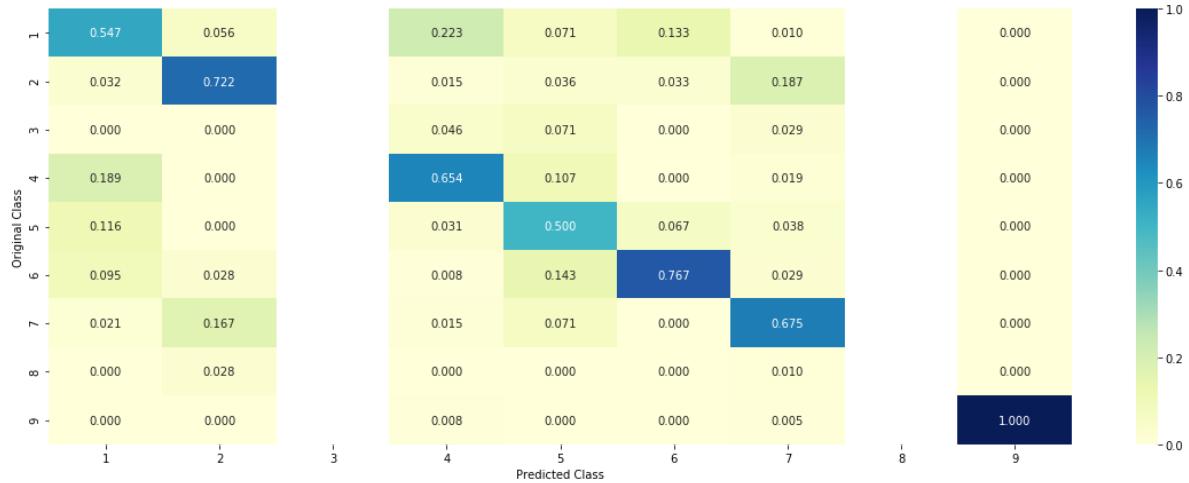
Log loss : 1.0185430453197877

Number of mis-classified points : 0.35150375939849626

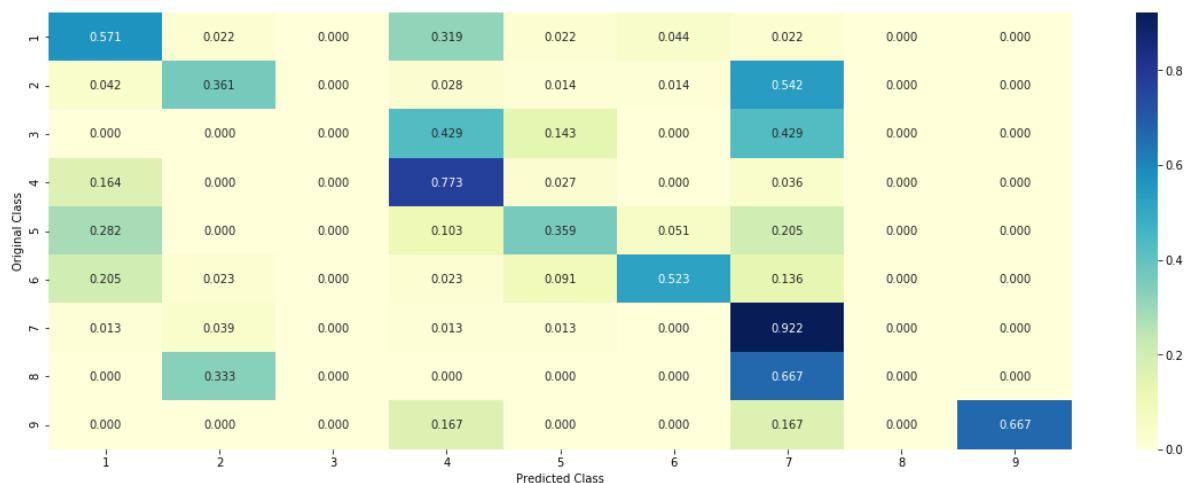
----- Confusion matrix -----



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



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



CONCLUSION

(a). Procedure Followed :- STEP 1:- Replace CountVectorizer() by CountVectorizer(ngram_range=(1,2)) in all the one hot encoding section of gene , variation and text features to get unigrams and bigrams .

STEP 2:- Then Apply Logistic Regression

(b).Table (Model Performances):-

In [92]:

```
# Names of models
names = ['LR With Class Balancing', 'LR Without Class Balancing']

# Training Loss
train_loss = [0.43498498, 0.4284463]

# Cross Validation Loss
cv_loss = [0.985348690, 1.018543045]

# Test Loss
test_loss = [0.98333931, 1.00868839]

# Percentage Misclassified points
misclassified = [0.360902255, 0.351503759]

alpha=[0.0001,11]
numbering = [1,2]

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
ss	Test_loss	Misclassified(%)		
1	LR With Class Balancing	0.0001	0.43498498	0.9853
2	LR Without Class Balancing	11	0.4284463	1.01854

TASK4

Trying different feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less as possible i.e 1 or near to 1

NOTE: HERE WE ARE TRYING DIFFERENT ENGINEERING HACKS:

1--> We also vectorise via TF-IDF with taken all words into consideration from gene, variation and text feature.

2--> We also vectorise our text via spaCy NLP en_core_web_sm and concat it with all above feature to improve our results.

-----vectorising our text via spaCy NLP en_core_web_sm-----

In [42]:

```
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.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
# extract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://Landinghub.visualstudio.com/visual-cpp-build-tools
import spacy

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

In [25]:

```
questions = list(train_df['TEXT'])
tfidf = TfidfVectorizer(lowercase=False, )
x=tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score

word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [96]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3)
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)
train_textfea_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 occurred
text_fea_dict = dict(zip(list(train_text_features),train_textfea_counts))

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

Total number of unique words in train data : 42861

In [17]:

```
from tqdm import tqdm
import spacy
```

In [34]:

```
import en_core_web_sm
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = en_core_web_sm.load()

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(train_df['TEXT'])):
    doc1 = nlp(qu1)
    # is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)
```

0%
| 0/2124 [00:00<?, ?it/s]

0%
| 1/2124 [00:54<32:20:54, 54.85s/it]

0%
| 2/2124 [00:56<22:50:20, 38.75s/it]

0%
| 3/2124 [01:26<21:23:10, 36.30s/it]

In [37]:

```
train_df['q1_feats_m'] = list(vecs1)
```

In [47]:

```
df3_q1 = pd.DataFrame(train_df.q1_feats_m.values.tolist(), index=train_df.index)
```

In [52]:

```
if not os.path.isfile('FEATUREtfidfw2v.csv'):
    df3_q1['ID']=train_df['ID']
    result = train_df.merge(df3_q1, on='ID', how='left')
    result.to_csv('FEATUREtfidfw2v.csv')
```

In [61]:

```
questions = list(test_df['TEXT'])
tfidf = TfidfVectorizer(lowercase=False, )
x=tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score

word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [62]:

```

import en_core_web_sm
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = en_core_web_sm.load()

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(test_df['TEXT'])):
    doc1 = nlp(qu1)
    # is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
test_df['q1_feats_m'] = list(vecs1)

```

0%|
| 0/665 [00:00<?, ?it/s]

0%|
| 1/665 [00:02<32:31, 2.94s/it]

0%||
| 2/665 [00:03<24:54, 2.25s/it]

0%||
| 3/665 [00:04<19:15, 1.75s/it]

In [63]:

```
df3_q1 = pd.DataFrame(test_df.q1_feats_m.values.tolist(), index=test_df.index)
```

In [64]:

```

if not os.path.isfile('FEATUREtfidfw2vTEST.csv'):
    df3_q1['ID']=test_df['ID']
    resultEST = test_df.merge(df3_q1, on='ID', how='left')
    resultEST.to_csv('FEATUREtfidfw2vTEST.csv')

```

CV

In [65]:

```
questions = list(cv_df['TEXT'])
tfidf = TfidfVectorizer(lowercase=False, )
x=tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score

word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [66]:

```
import en_core_web_sm
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = en_core_web_sm.load()

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(cv_df['TEXT'])):
    doc1 = nlp(qu1)
    # is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1),96])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
cv_df['q1_feats_m'] = list(vecs1)
```

0%
| 0/532 [00:00<?, ?it/s]

0%||
| 1/532 [00:04<37:14, 4.21s/it]

0%||
| 2/532 [00:05<28:45, 3.25s/it]

1%||
| 3/532 [00:28<1:21:19, 9.22s/it]

In [79]:

```
df3_q1 = pd.DataFrame(cv_df.q1_feats_m.values.tolist(), index=cv_df.index)
```

In [84]:

```
if not os.path.isfile('FEATUREtfidfw2vCV.csv'):
    df3_q1['ID']=cv_df['ID']
    resultCV = cv_df.merge(df3_q1, on='ID', how='left')
    resultCV.to_csv('FEATUREtfidfw2vCV.csv')
```

In [153]:

```
if not os.path.isfile('FEATUREtfidfw2vCV2.csv'):
    resultCV2 = df3_q1
    resultCV2.to_csv('FEATUREtfidfw2vCV2.csv')
```

NOTE:- We will use above file little further after other engineering hack (vectorising our text via spaCy NLP en_core_web_sm).

-----task 4.2-----

vectorising via TF-IDF with taken all words into consideration from gene, variation and text features.

In [48]:

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

In [49]:

```
len(corpus)
```

Out[49]:

9963

In [51]:

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

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

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

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

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

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

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

In [52]:

```
# merging gene, variance and text features

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

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

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

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

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

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

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

In [53]:

```
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, 209601)
(number of data points * number of features) in test data = (665, 209601)
(number of data points * number of features) in cross validation data = (53, 2, 209601)
```

In [54]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding)
print("(number of data points * number of features) in test data = ", test_x_responseCoding)
print("(number of data points * number of features) in cross validation data =", cv_x_respo
```

```
Response encoding features :
(number of data points * number of features) in train data = (2124, 27)
(number of data points * number of features) in test data = (665, 27)
(number of data points * number of features) in cross validation data = (53
2, 27)
```

Base Line Model

Naive Bayes

Hyper parameter tuning

In [249]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----


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 probabilités we use Log-probability estimat
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_)

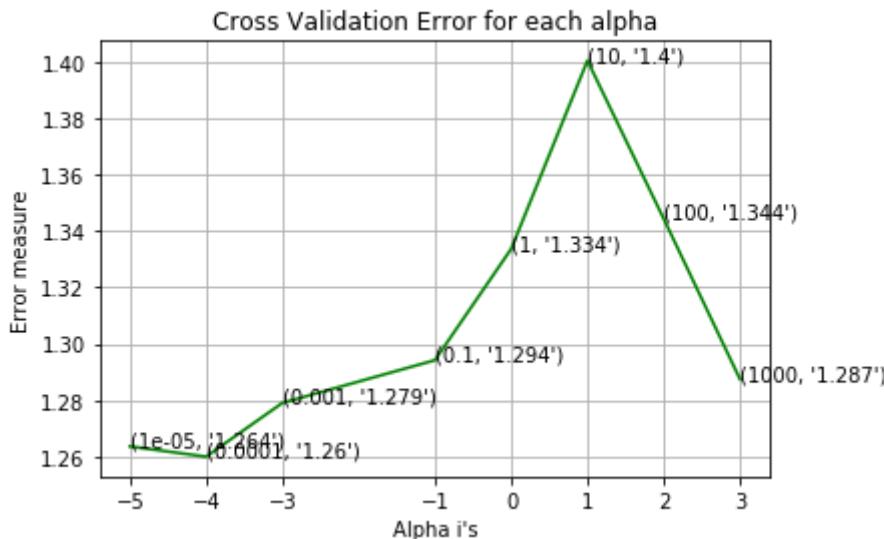
# Variables that will be used in the end to make comparison table of all models
nb_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_
nb_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1
nb_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding),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

```

```

for alpha = 1e-05
Log Loss : 1.2636175748288012
for alpha = 0.0001
Log Loss : 1.2599940390140119
for alpha = 0.001
Log Loss : 1.2790308557635353
for alpha = 0.01
Log Loss : 1.2941790828859339
for alpha = 0.1
Log Loss : 1.3336256681289844
for alpha = 1
Log Loss : 1.3443274215003693
for alpha = 10
Log Loss : 1.4003673863579484
for alpha = 100
Log Loss : 1.3443274215003693
for alpha = 1000
Log Loss : 1.2874832184297773

```



```

For values of best alpha = 0.0001 The train log loss is: 0.862966080684403
For values of best alpha = 0.0001 The cross validation log loss is: 1.25999
40390140119
For values of best alpha = 0.0001 The test log loss is: 1.293204403095999

```

Testing the model with best hyper parameters

In [250]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----
```



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----
```



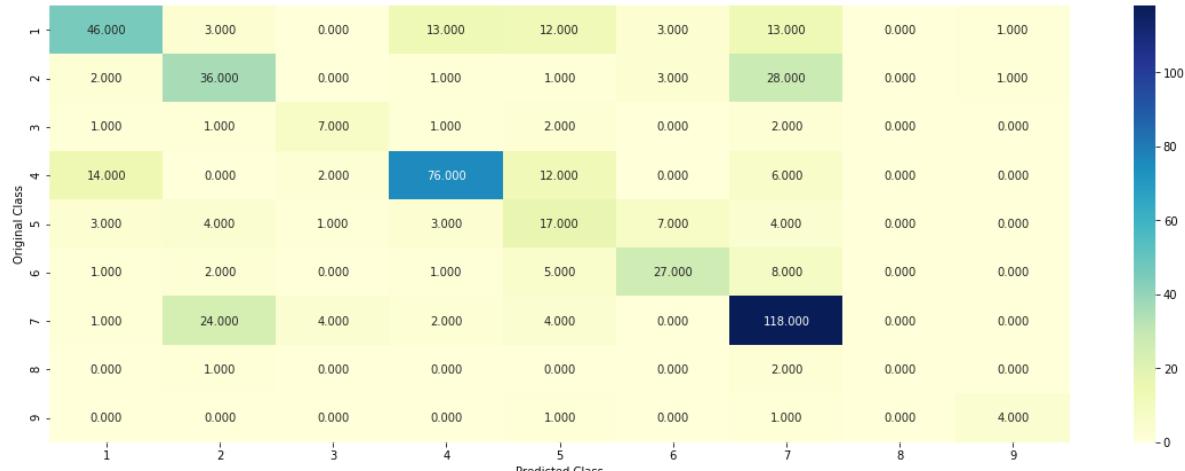
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites 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)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))

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

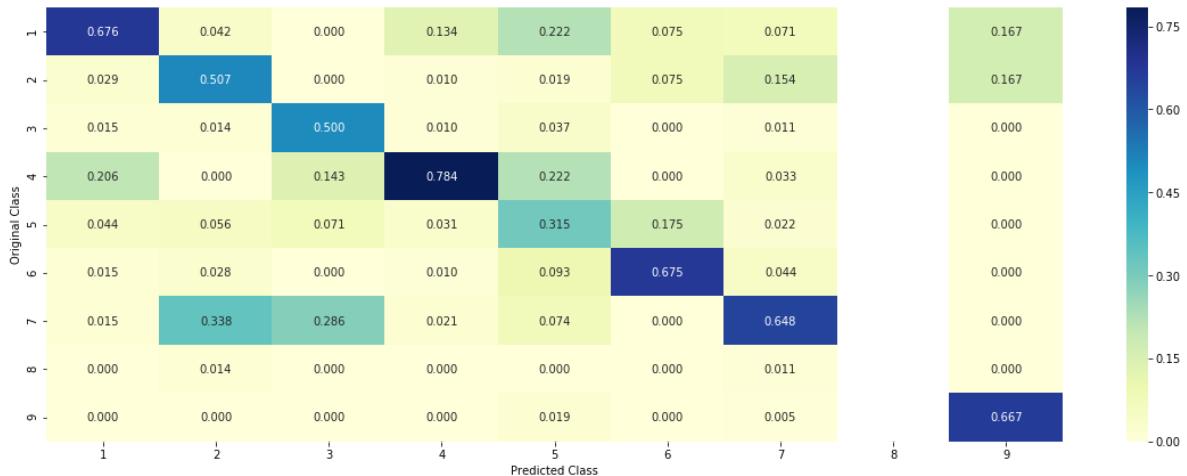
Log Loss : 1.2599940390140119

Number of missclassified point : 0.37781954887218044

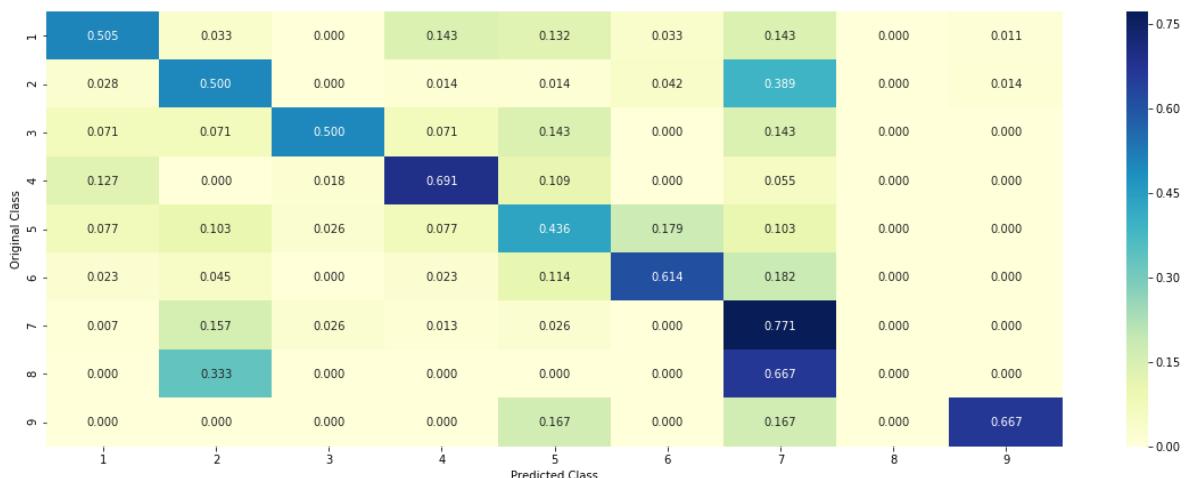
----- Confusion matrix -----



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



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



Feature Importance, Correctly classified point

In [251]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class : ", predicted_cls[0])
print("Predicted Class Probabilities: ", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class : ", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5661 0.0877 0.0123 0.1178 0.0374 0.0407
0.1303 0.0036 0.004 ]]
Actual Class : 4
```

Feature Importance, Incorrectly classified point

In [252]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1      0.0812  0.0112  0.1075  0.0344  0.0375
0.6212  0.0034  0.0036]]
Actual Class : 7
-----
```

K Nearest Neighbour Classification

Hyper parameter tuning

In [253]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/general
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', Leaf_size=30, p=
# 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-nea
#-----

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

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

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

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

```

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_

```

Variables that will be used in the end to make comparison table of all models

```

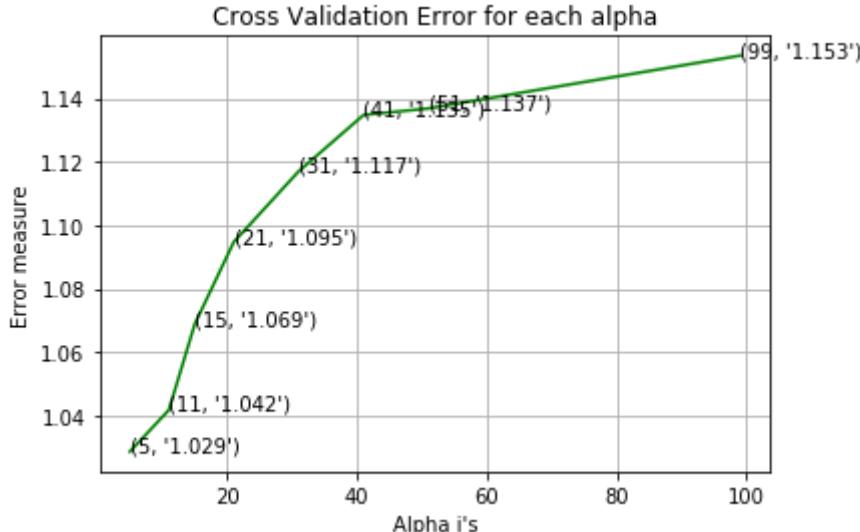
knn_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labels=clf.cla
knn_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.classes_, ep
knn_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=clf.classes_

```

```

for alpha = 5
Log Loss : 1.0286915438663768
for alpha = 11
Log Loss : 1.0417943000730487
for alpha = 15
Log Loss : 1.068867888844843
for alpha = 21
Log Loss : 1.0946906560944136
for alpha = 31
Log Loss : 1.1169587475472424
for alpha = 41
Log Loss : 1.1347140844351764
for alpha = 51
Log Loss : 1.1367712357363147
for alpha = 99
Log Loss : 1.1534130469659825

```



For values of best alpha = 5 The train log loss is: 0.4816834308635685

For values of best alpha = 5 The cross validation log loss is: 1.0286915438663768

For values of best alpha = 5 The test log loss is: 1.0828043626651087

Testing the model with best hyper parameters

In [254]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/general
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', Leaf_size=30, p=
# 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-nea
# -----
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

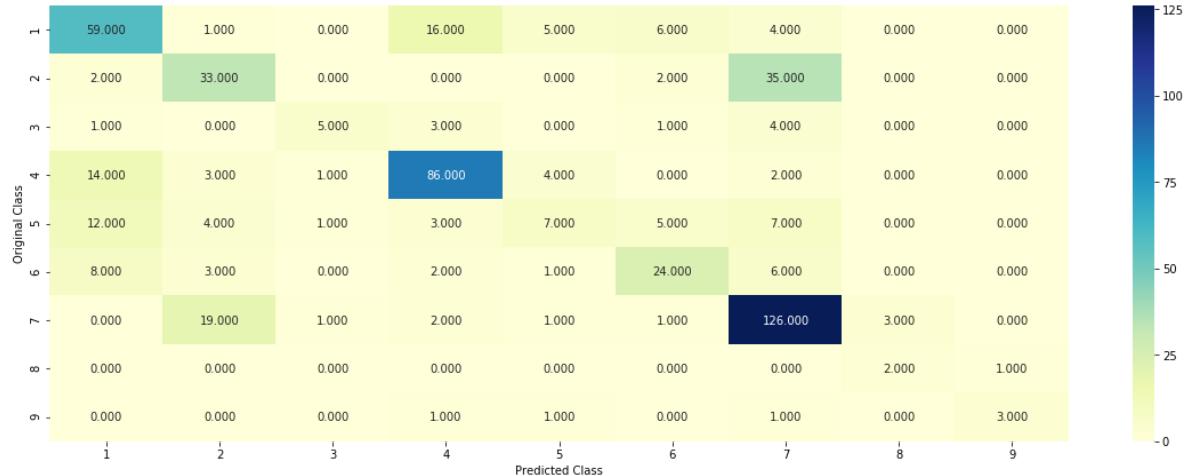
# Variables that will be used in the end to make comparison table of models
knn_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y).sign()))

```

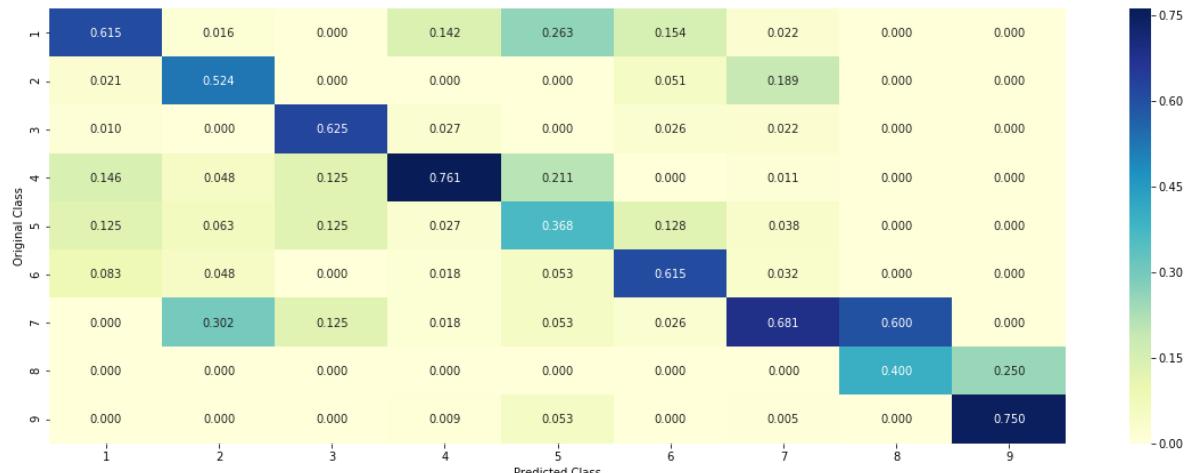
Log loss : 1.0286915438663768

Number of mis-classified points : 0.35150375939849626

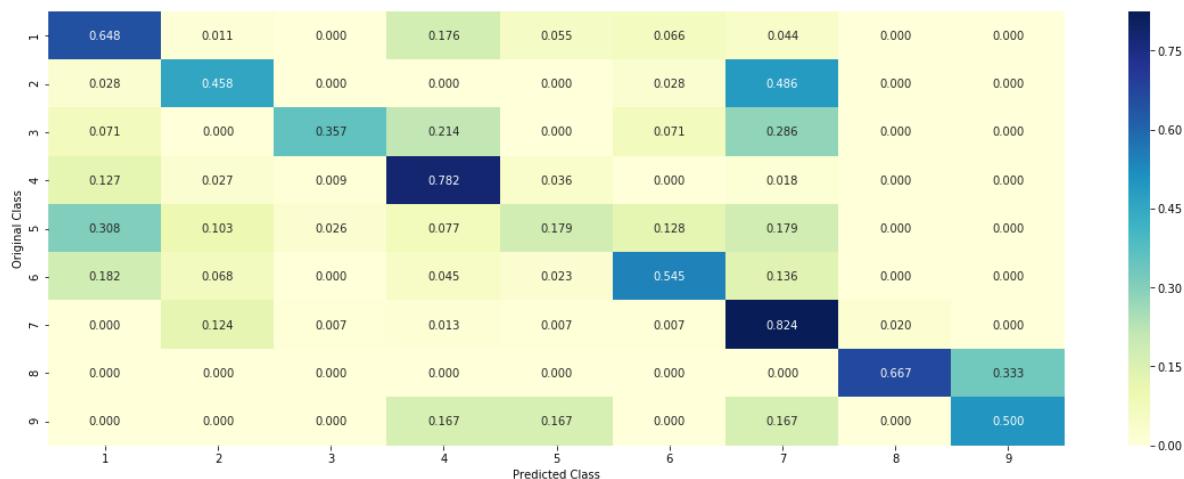
----- Confusion matrix -----



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



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



4.2.3. Sample Query point -1

In [255]:

```
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[be
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",
print("Frequency of nearest points : ",Counter(train_y[neighbors[1][0]])))
```

Predicted Class : 7

Actual Class : 4

The 5 nearest neighbours of the test points belongs to classes [1 4 4 4 4]

Frequency of nearest points : Counter({4: 4, 1: 1})

4.2.4. Sample Query Point-2

In [256]:

```
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[be
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test pc
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]])))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 5 and the nearest neighbours of the test points belon
gs to classes [7 7 7 7 7]
Frequency of nearest points : Counter({7: 5})
```

Logistic Regression

With Class balancing

Hyper parameter tuning

In [257]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/gen
# -----
# default parameters
# 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_
    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 estimat
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

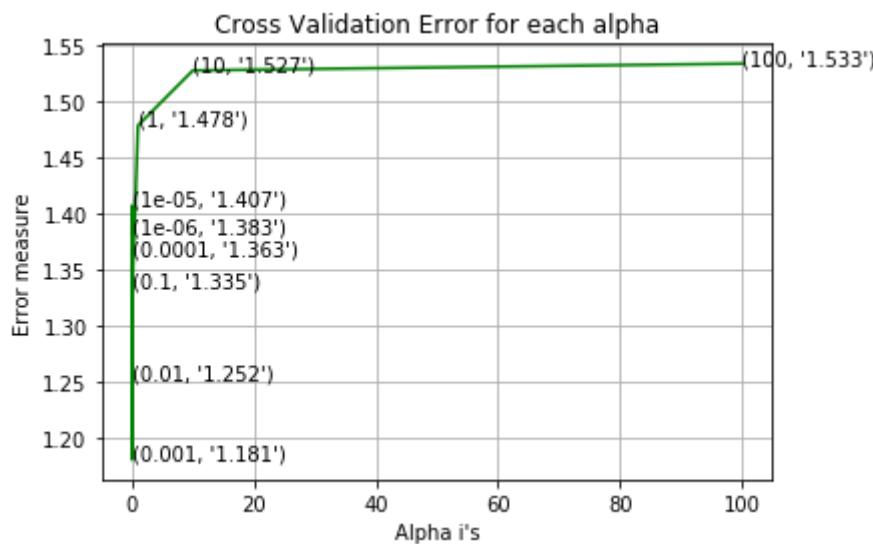
```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 1e-06
Log Loss : 1.382896329477155
for alpha = 1e-05
Log Loss : 1.406930679589009
for alpha = 0.0001
Log Loss : 1.3627067059703108
for alpha = 0.001
Log Loss : 1.181015192123009
for alpha = 0.01
Log Loss : 1.2521096163495384
for alpha = 0.1
Log Loss : 1.3351409303188035
for alpha = 1
Log Loss : 1.4781154491767066
for alpha = 10
Log Loss : 1.5270200067970783
for alpha = 100
Log Loss : 1.5331226242000724

```



For values of best alpha = 0.001 The train log loss is: 0.6791066310298077
 For values of best alpha = 0.001 The cross validation log loss is: 1.181015192123009
 For values of best alpha = 0.001 The test log loss is: 1.116954897770258

4.3.1.2. Testing the model with best hyper parameters

In [258]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c

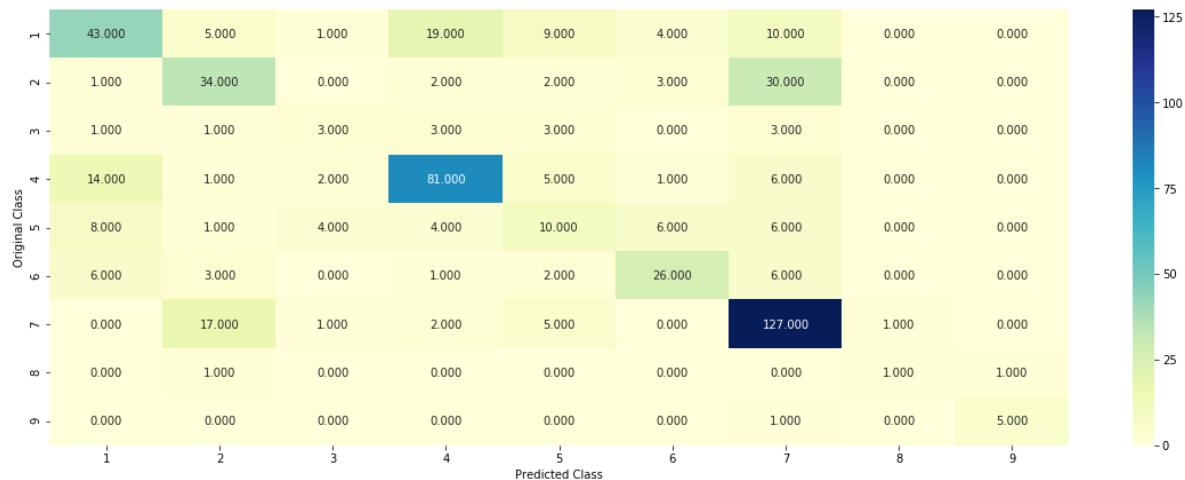
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

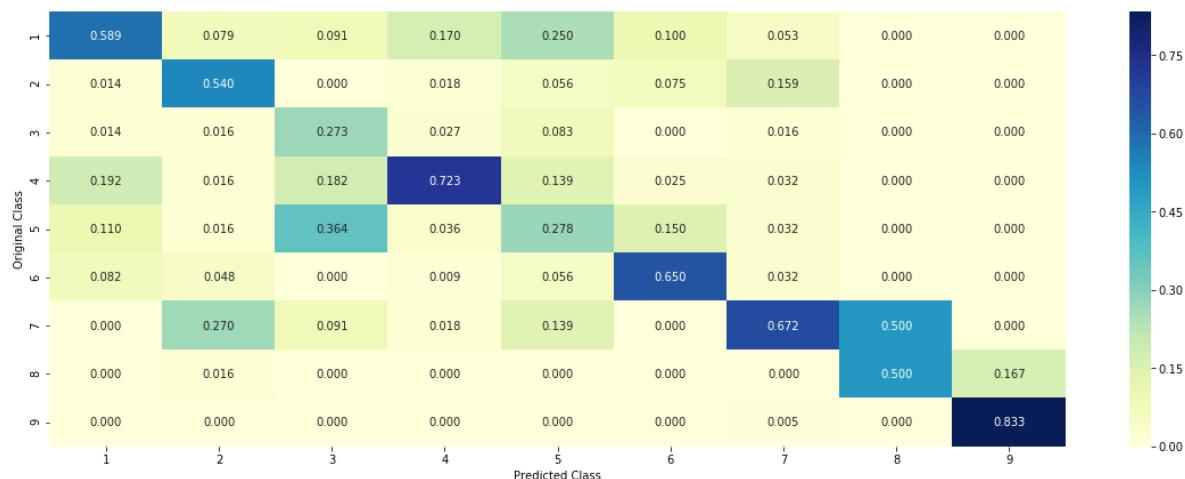
Log loss : 1.181015192123009

Number of mis-classified points : 0.37969924812030076

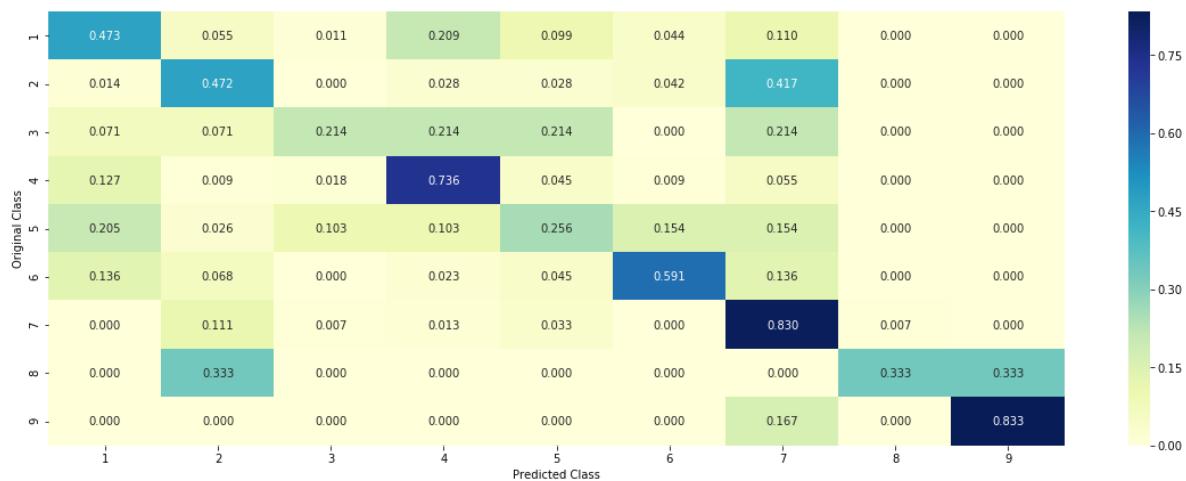
----- Confusion matrix -----



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



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



4.3.1.3. Feature Importance

In [259]:

```
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 [260]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='')
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2202 0.1398 0.0159 0.4649 0.0415 0.0203
0.0399 0.031 0.0265]]
Actual Class : 4
-----
```

Incorrectly Classified point

In [261]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.031 0.0861 0.0139 0.0485 0.0362 0.0076
0.7501 0.0158 0.0108]]
Actual Class : 7
-----
```

Without Class balancing

Hyperparameter tuning

In [262]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



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



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

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



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

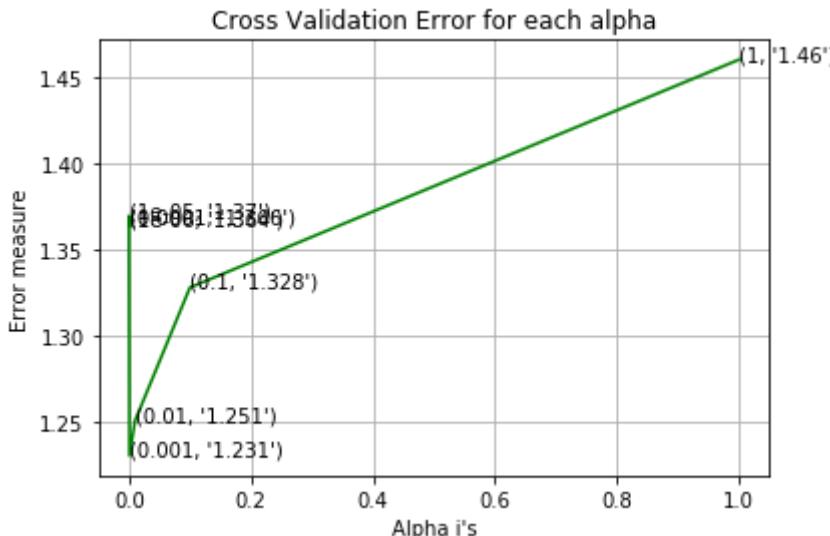
```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 1e-06
Log Loss : 1.3641803149507898
for alpha = 1e-05
Log Loss : 1.3697580339765207
for alpha = 0.0001
Log Loss : 1.3661726525075808
for alpha = 0.001
Log Loss : 1.230755439498845
for alpha = 0.01
Log Loss : 1.2509779193634274
for alpha = 0.1
Log Loss : 1.3281864080424566
for alpha = 1
Log Loss : 1.460260478875054

```



```

For values of best alpha = 0.001 The train log loss is: 0.6569506540671142
For values of best alpha = 0.001 The cross validation log loss is: 1.230755
439498845
For values of best alpha = 0.001 The test log loss is: 1.1397148544080722

```

Testing model with best hyper parameters

In [265]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# 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, c

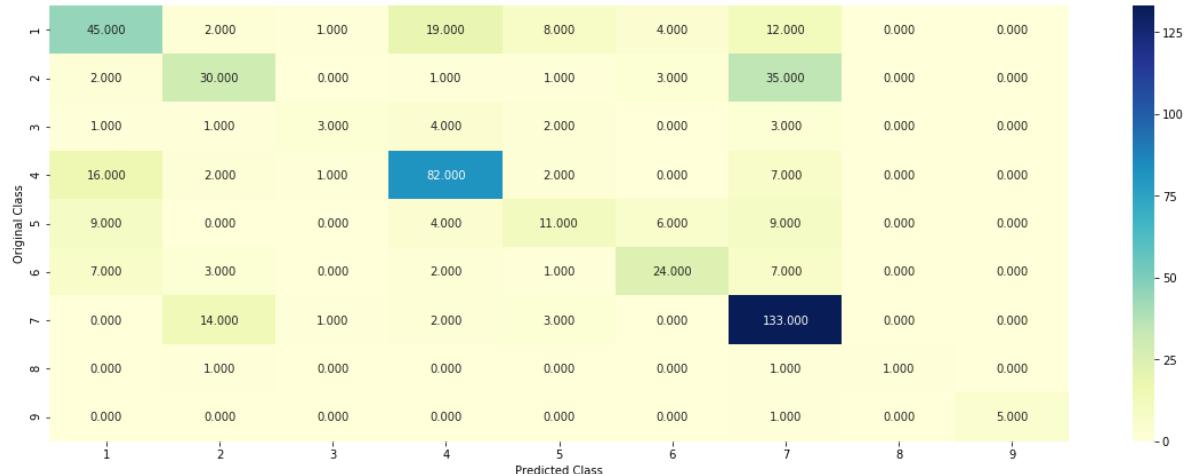
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

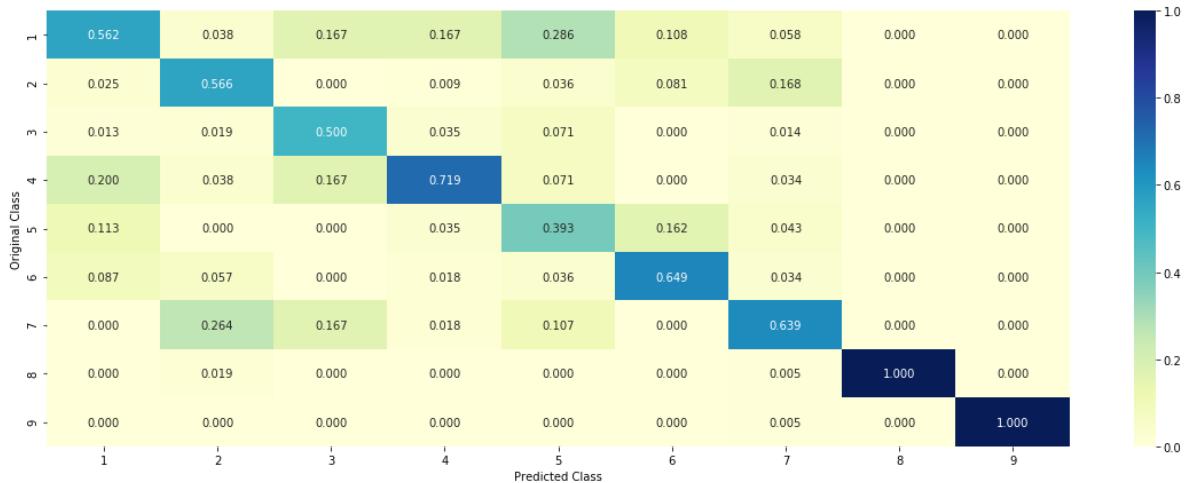
Log loss : 1.230755439498845

Number of mis-classified points : 0.37218045112781956

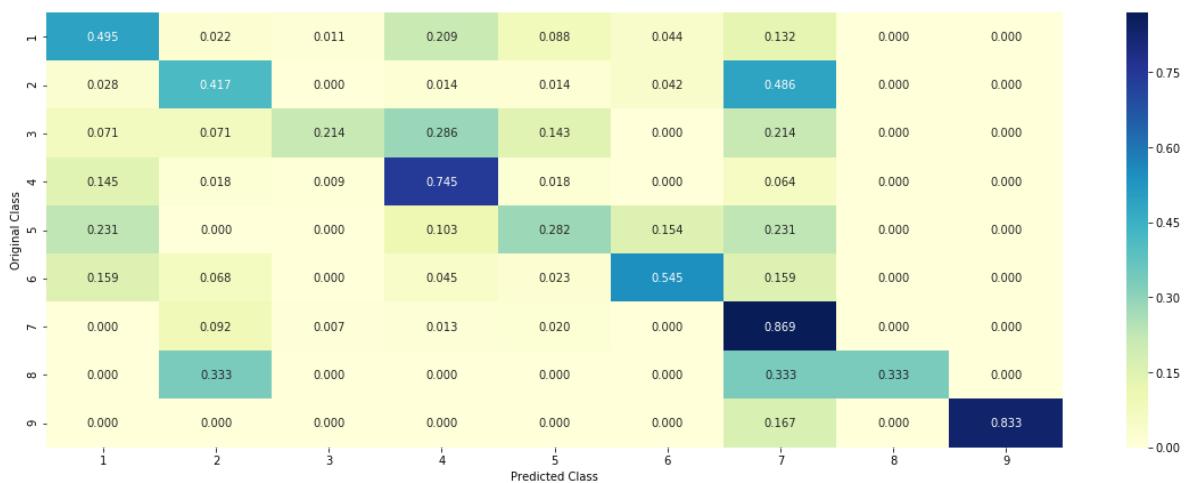
----- Confusion matrix -----



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



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



Feature Importance, Correctly Classified point

In [266]:

```

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

```

Predicted Class : 4
 Predicted Class Probabilities: [[0.2705 0.1406 0.0025 0.4053 0.0152 0.0107
 0.0842 0.0704 0.0006]]
 Actual Class : 4

Feature Importance, Incorrectly Classified point

In [267]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[3.470e-02 7.830e-02 3.200e-03 6.440e-02 1.5
50e-02 4.700e-03 7.808e-01
1.780e-02 5.000e-04]]
Actual Class : 7
-----
```

Linear Support Vector Machines

Hyper parameter tuning

In [268]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



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



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

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



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

```

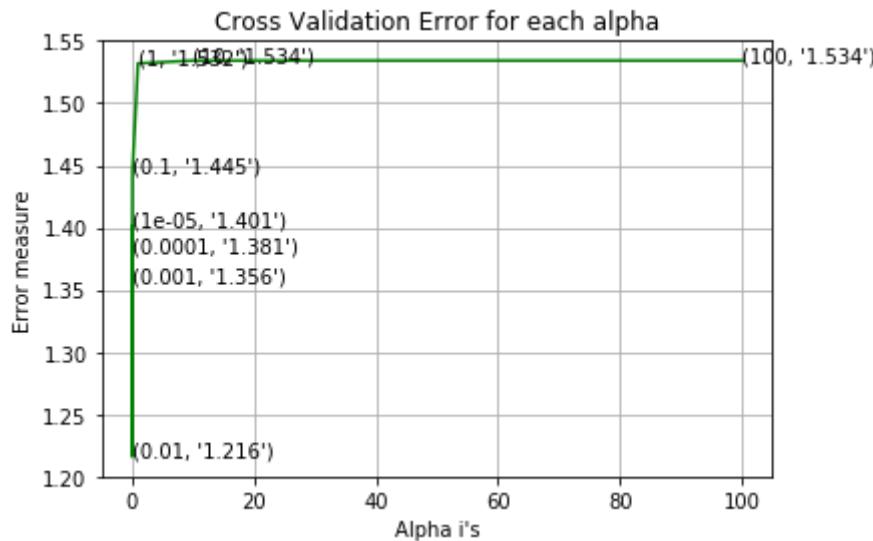
sig_clf.fit(train_x_onehotCoding, train_y)

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_)

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

for C = 1e-05
Log Loss : 1.4012909445702306
for C = 0.0001
Log Loss : 1.381131192603443
for C = 0.001
Log Loss : 1.355705566290905
for C = 0.01
Log Loss : 1.2159251469517538
for C = 0.1
Log Loss : 1.4448111787584679
for C = 1
Log Loss : 1.5318994856199861
for C = 10
Log Loss : 1.5341410144305523
for C = 100
Log Loss : 1.5341347709887743

```



```

For values of best alpha = 0.01 The train log loss is: 0.681009539603264
For values of best alpha = 0.01 The cross validation log loss is: 1.2159251
469517538
For values of best alpha = 0.01 The test log loss is: 1.160057432243769

```

Testing model with best hyper parameters

In [269]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video Link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM
# -----
```

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

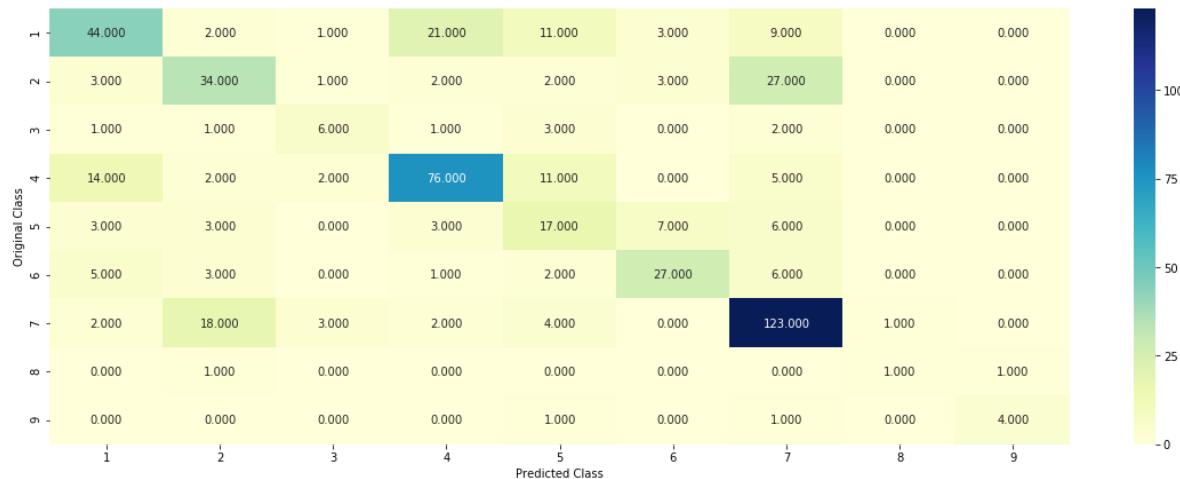
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

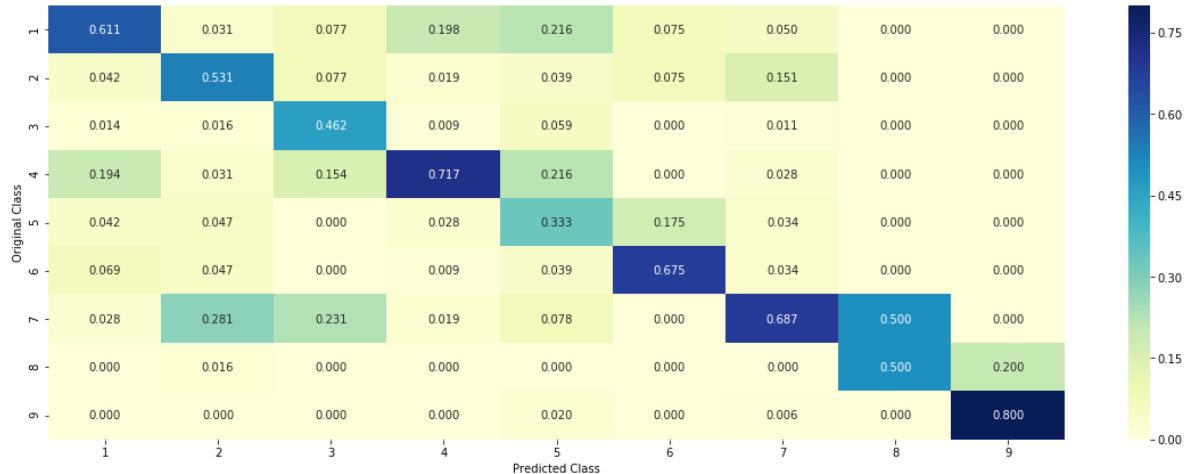
Log loss : 1.2159251469517538

Number of mis-classified points : 0.37593984962406013

----- Confusion matrix -----



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



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



Feature Importance

For Correctly classified point

In [270]:

```

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

```

Predicted Class : 4

Predicted Class Probabilities: [[0.2645 0.0554 0.0059 0.5541 0.0248 0.0111
0.0411 0.0291 0.014]]

Actual Class : 4

For Incorrectly classified point

In [271]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.024  0.0475  0.0055  0.051   0.0207  0.0044
0.8175  0.024   0.0055]]
Actual Class : 7
-----
```

Random Forest Classifier

Hyper parameter tuning (With One hot Encoding)

In [272]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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-9))
        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_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```
'''  
  
best_alpha = np.argmin(cv_log_error_array)  
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c  
clf.fit(train_x_onehotCoding, train_y)  
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig_clf.fit(train_x_onehotCoding, train_y)  
  
predict_y = sig_clf.predict_proba(train_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:")  
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo  
predict_y = sig_clf.predict_proba(test_x_onehotCoding)  
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",  
  
# Variables that will be used in the end to make comparison table of all models  
rf_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classe  
rf_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1  
rf_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_,  
  
for n_estimators = 100 and max depth = 5  
Log Loss : 1.2498296478319073  
for n_estimators = 100 and max depth = 10  
Log Loss : 1.1714012529782467  
for n_estimators = 200 and max depth = 5  
Log Loss : 1.2282298796652482  
for n_estimators = 200 and max depth = 10  
Log Loss : 1.157632090163359  
for n_estimators = 500 and max depth = 5  
Log Loss : 1.2182096294495122  
for n_estimators = 500 and max depth = 10  
Log Loss : 1.1494394592994939  
for n_estimators = 1000 and max depth = 5  
Log Loss : 1.2136127789930673  
for n_estimators = 1000 and max depth = 10  
Log Loss : 1.147359705716597  
for n_estimators = 2000 and max depth = 5  
Log Loss : 1.212976825449904  
for n_estimators = 2000 and max depth = 10  
Log Loss : 1.1479651593110227  
For values of best estimator = 1000 The train log loss is: 0.63463704506109  
51  
For values of best estimator = 1000 The cross validation log loss is: 1.147  
359705716597  
For values of best estimator = 1000 The test log loss is: 1.126528119685484  
8
```

Testing model with best hyper parameters (One Hot Encoding)

In [273]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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-classifier
# -----

```

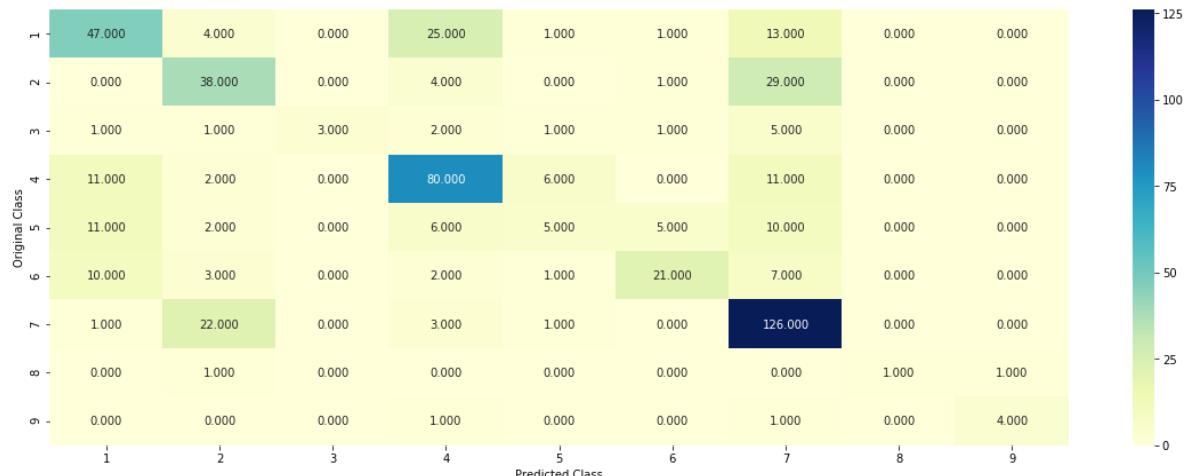
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_c
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

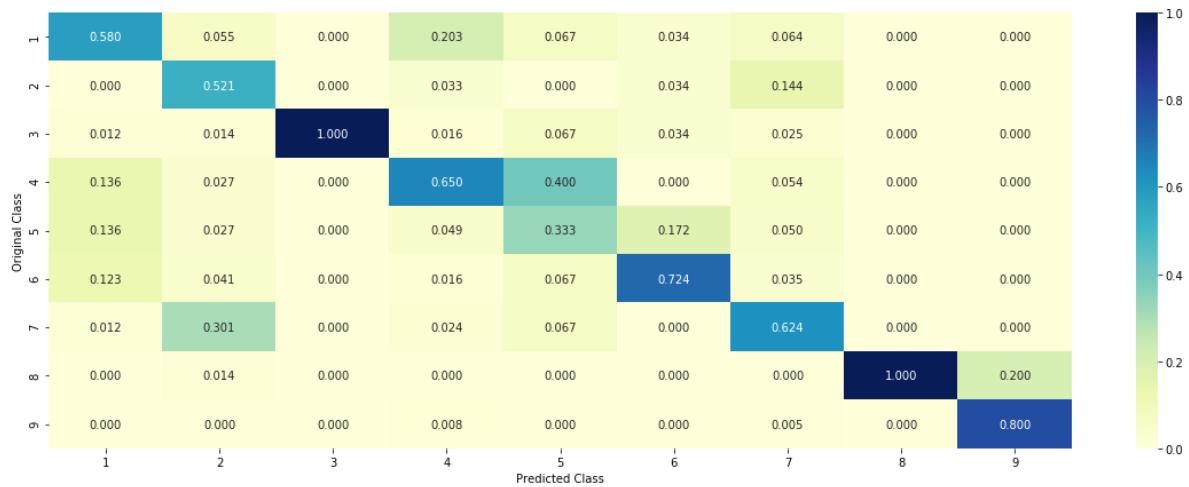
Log loss : 1.147359705716597

Number of mis-classified points : 0.3890977443609023

----- Confusion matrix -----



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



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



Feature Importance

Correctly Classified point

In [274]:

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

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2352 0.1229 0.0208 0.2871 0.0644 0.0662
0.1873 0.009 0.0072]]
Actual Class : 4
-----
```

4.5.3.2. Inorrectly Classified point

In [275]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0578 0.1993 0.0163 0.0576 0.0506 0.0419
0.5632 0.0098 0.0036]]
Actuall Class : 7
-----
```

+4.5.3. Hyper paramter tuning (With Response Coding)

In [276]:

```

# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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/rando
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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
        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_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```
'''  
  
best_alpha = np.argmin(cv_log_error_array)  
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c  
clf.fit(train_x_responseCoding, train_y)  
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
sig_clf.fit(train_x_responseCoding, train_y)  
  
predict_y = sig_clf.predict_proba(train_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log  
predict_y = sig_clf.predict_proba(cv_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log lo  
predict_y = sig_clf.predict_proba(test_x_responseCoding)  
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_
```

Variables that will be used in the end to make comparison table of all models

```
rf_response_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labels=  
rf_response_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.clas  
rf_response_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=cl
```

```
< └─>  
for n_estimators = 10 and max depth = 2  
Log Loss : 2.020210030510885  
for n_estimators = 10 and max depth = 3  
Log Loss : 1.6885996241440222  
for n_estimators = 10 and max depth = 5  
Log Loss : 1.6485804136423752  
for n_estimators = 10 and max depth = 10  
Log Loss : 1.9271970775455762  
for n_estimators = 50 and max depth = 2  
Log Loss : 1.7105735356095693  
for n_estimators = 50 and max depth = 3  
Log Loss : 1.4677803370319527  
for n_estimators = 50 and max depth = 5  
Log Loss : 1.3449341266878405  
for n_estimators = 50 and max depth = 10  
Log Loss : 1.7305357692341607  
for n_estimators = 100 and max depth = 2  
Log Loss : 1.6117759340687894  
for n_estimators = 100 and max depth = 3  
Log Loss : 1.5065418603384086  
for n_estimators = 100 and max depth = 5  
Log Loss : 1.2994398280626545  
for n_estimators = 100 and max depth = 10  
Log Loss : 1.7150358854036032  
for n_estimators = 200 and max depth = 2  
Log Loss : 1.7005967327460845  
for n_estimators = 200 and max depth = 3  
Log Loss : 1.5215058749845087  
for n_estimators = 200 and max depth = 5  
Log Loss : 1.3384797555008465  
for n_estimators = 200 and max depth = 10  
Log Loss : 1.7090753661946534  
for n_estimators = 500 and max depth = 2  
Log Loss : 1.7506253803638006  
for n_estimators = 500 and max depth = 3  
Log Loss : 1.5653913674965196  
for n_estimators = 500 and max depth = 5  
Log Loss : 1.361866008806591  
for n_estimators = 500 and max depth = 10  
Log Loss : 1.7037465052277823
```

```
for n_estimators = 1000 and max depth =  2
Log Loss : 1.716426514423557
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5409738899398664
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3399131377270832
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6885330925552782
For values of best alpha =  100 The train log loss is: 0.052462961217133106
For values of best alpha =  100 The cross validation log loss is: 1.29943982
80626542
For values of best alpha =  100 The test log loss is: 1.3014627532073675
```

Testing model with best hyper parameters (Response Coding)

In [277]:

```
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, ve
# 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/rando
# -----
```

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int

predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y,

clf.fit(train_x_responseCoding, train_y)

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

sig_clf.fit(train_x_responseCoding, train_y)

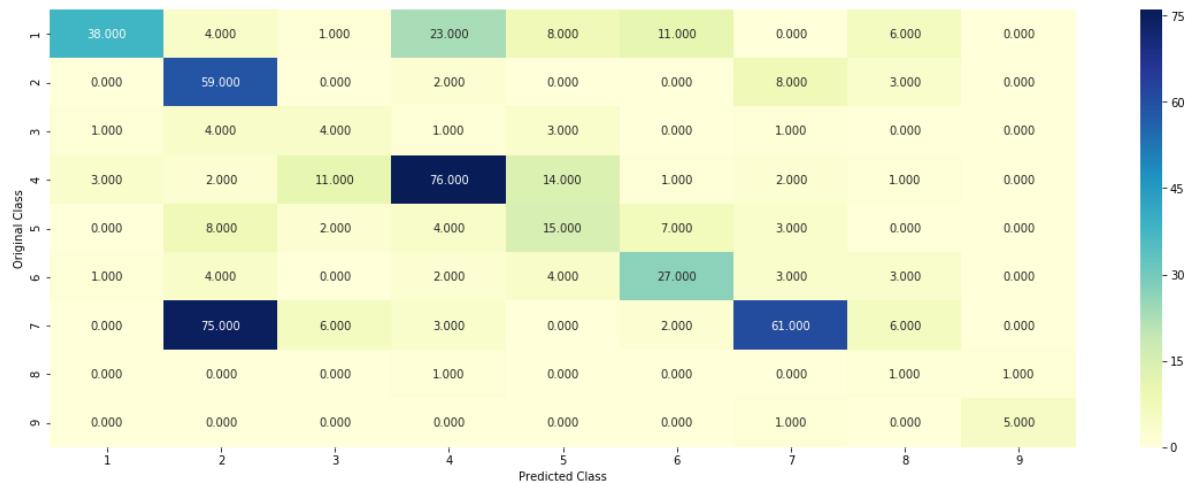
Variables that will be used in the end to make comparison table of models

rf_response_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding) - cv_y))

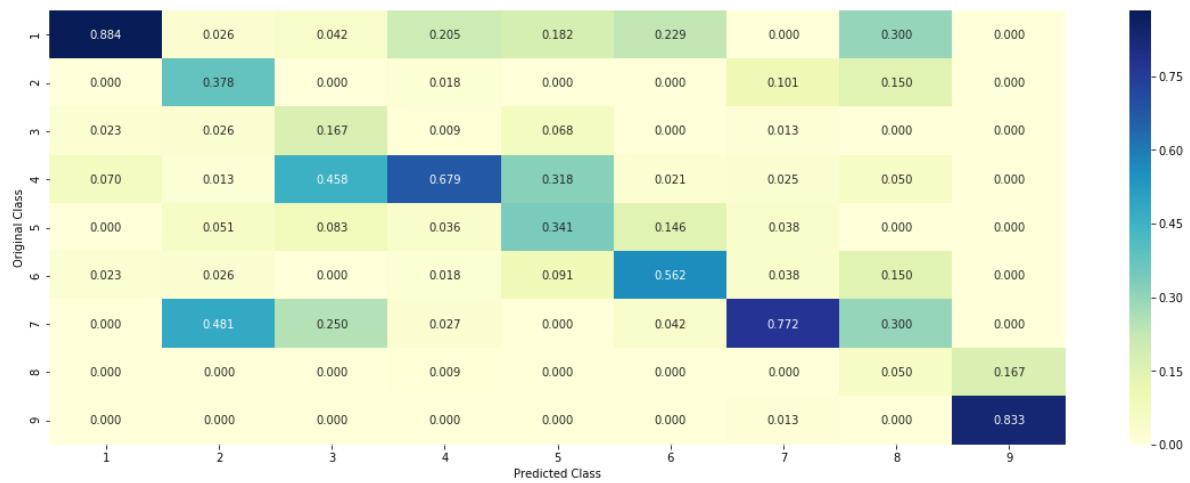
Log loss : 1.2994398280626545

Number of mis-classified points : 0.462406015037594

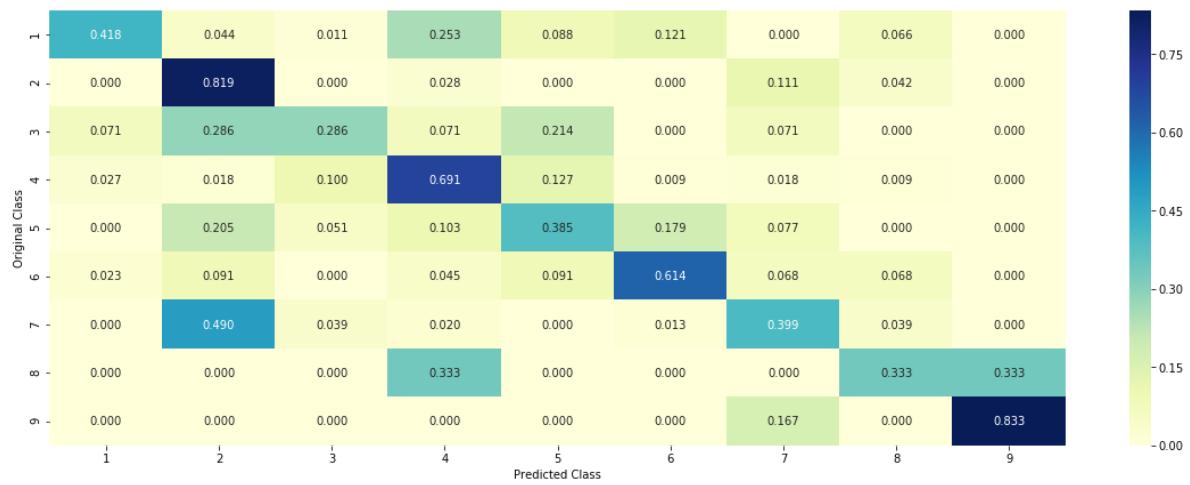
----- Confusion matrix -----



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



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



Feature Importance

Correctly Classified point

In [278]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCodir
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: [[0.1269 0.0154 0.1087 0.6363 0.0171 0.0409
0.0087 0.0209 0.025 ]]
Actual Class : 4
```

```
-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

Incorrectly Classified point

In [279]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0217 0.289  0.0626 0.0245 0.0362 0.0473
0.3718 0.1291 0.0178]]
Actual Class : 7
-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

Stack the models

testing with hyper parameter tuning

In [281]:

```
# 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,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# 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/geometry-of-linear-classifiers
# -----


# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')

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


# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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
# -----


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=42)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=42)
clf2.fit(train_x_onehotCoding, train_y)
```

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

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

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

```
Logistic Regression : Log Loss: 1.18
Support vector machines : Log Loss: 1.53
Naive Bayes : Log Loss: 1.28
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.046
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.542
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.146
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.221
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.449
```

testing the model with the best hyper parameters

In [282]:

```

lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, n_jobs=-1)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

# Variables that will be used in the end to make comparison table of all models
stack_train = log_error
stack_cv = log_error1
stack_test = log_error2
stack_missclassified = (np.count_nonzero((sclf.predict(test_x_onehotCoding) - test_y)) / test_y)

```

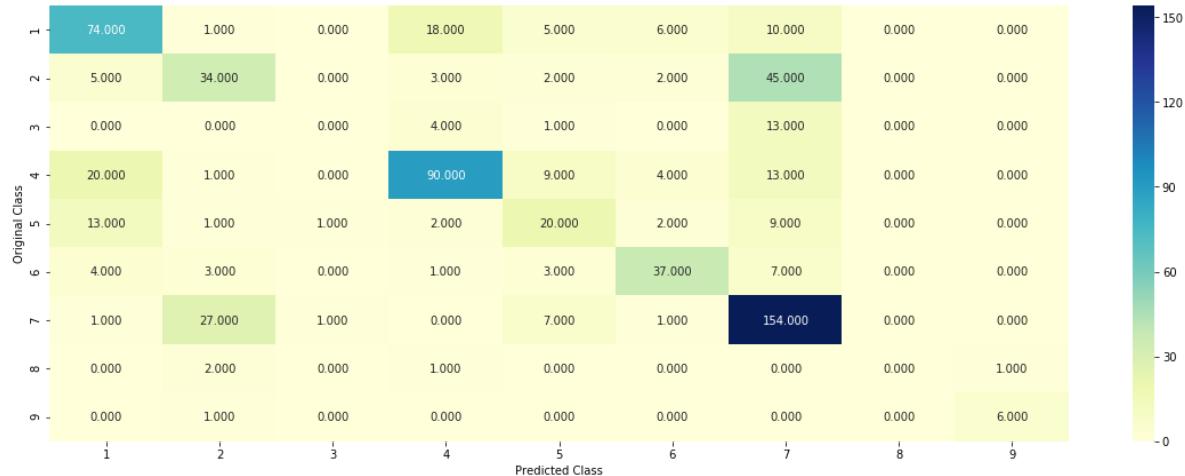
Log loss (train) on the stacking classifier : 0.6697751074173852

Log loss (CV) on the stacking classifier : 1.1463790492707695

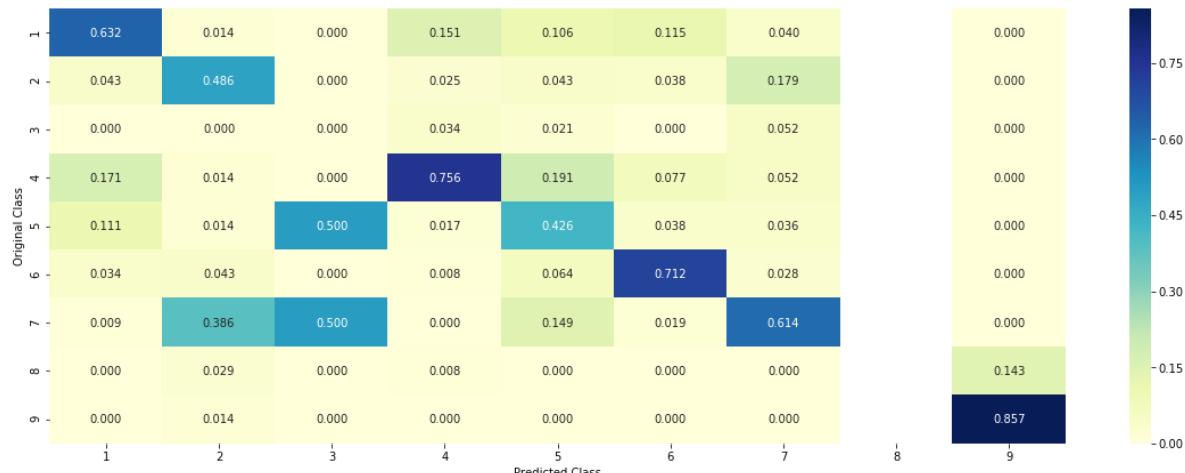
Log loss (test) on the stacking classifier : 1.175289256401481

Number of missclassified point : 0.37593984962406013

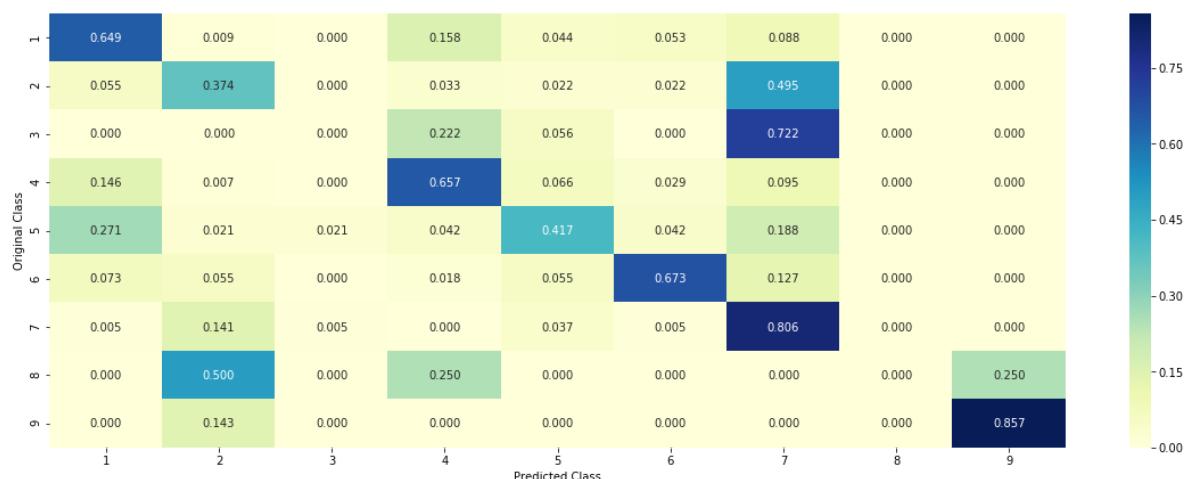
----- Confusion matrix -----



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



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

**Maximum Voting classifier**

In [283]:

```
#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)], 
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier : ", log_loss(train_y, vclf.predict_proba(tr
print("Log loss (CV) on the VotingClassifier : ", log_loss(cv_y, vclf.predict_proba(cv_x_one
print("Log loss (test) on the VotingClassifier : ", log_loss(test_y, vclf.predict_proba(test
print("Number of missclassified point : ", np.count_nonzero((vclf.predict(test_x_onehotCodir
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))

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

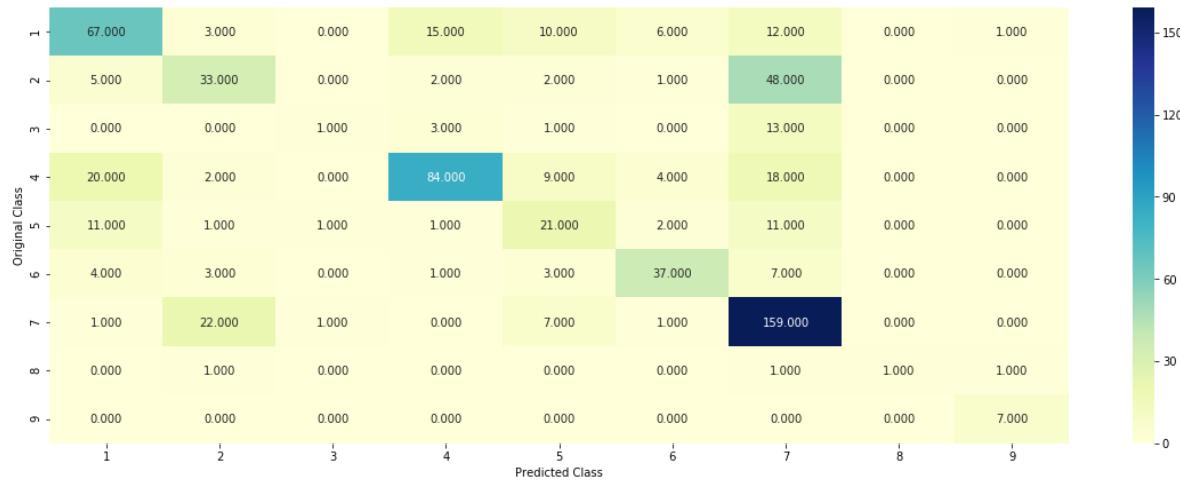
Log loss (train) on the VotingClassifier : 0.8992452713850164

Log loss (CV) on the VotingClassifier : 1.1958440607295253

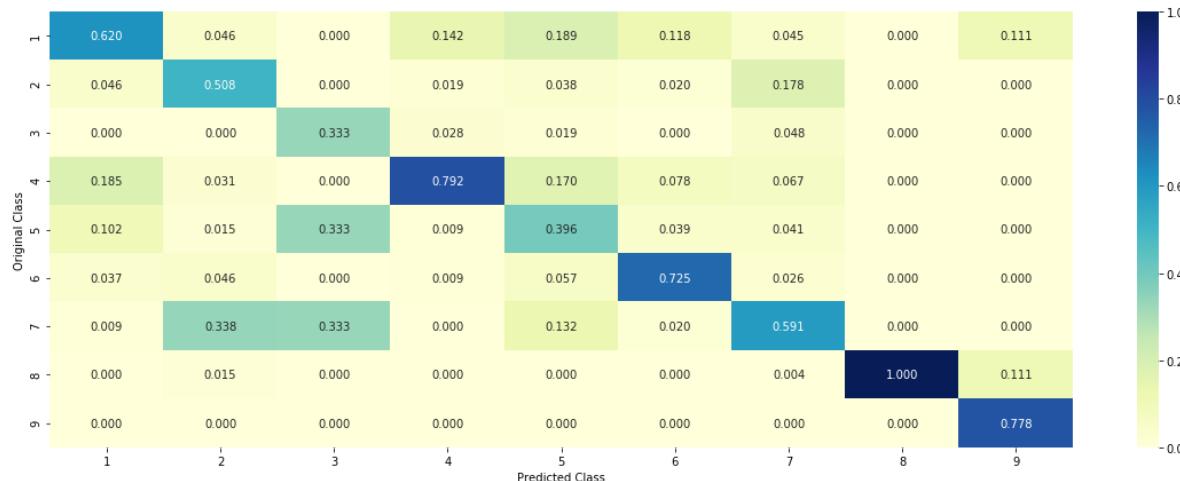
Log loss (test) on the VotingClassifier : 1.1921558409637527

Number of missclassified point : 0.38345864661654133

----- Confusion matrix -----

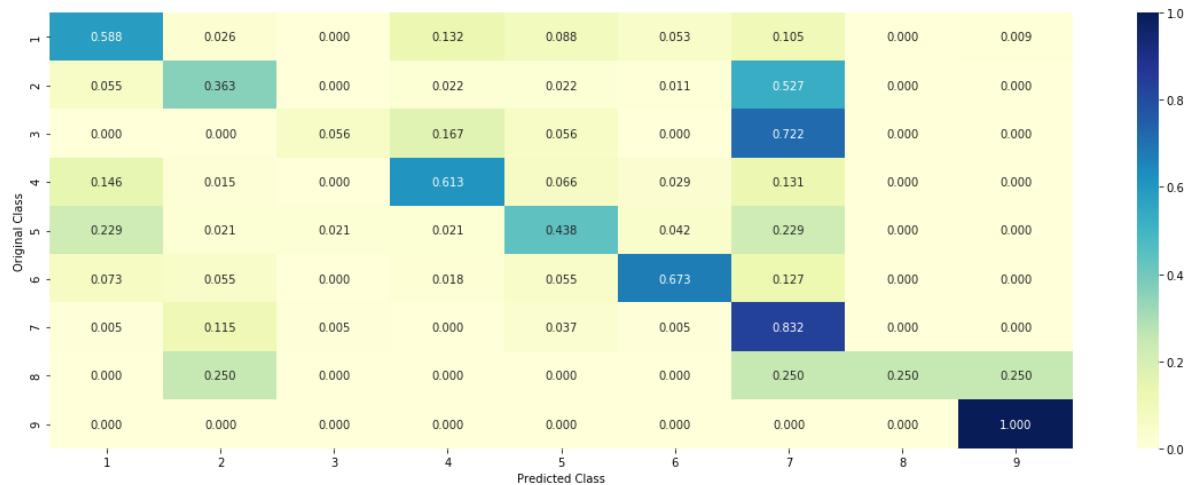


----- Precision matrix (Column Sum=1) -----



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

PersonalizedCancerDiagnosis



In [98]:

S.NO.	MODEL	hyperparameter	Train_loss	CV_lo
ss	Test_loss	log_loss	Misclassified(%)	
1	Naive Bayes	0.0001	0.86296	1.259
99	1.2932	1.259994	0.377819	
2	K-Nearest Neighbour	5	0.4816834	1.028
691	1.082804	1.028691	0.35150375	
3	LR With Class Balancing	0.001	0.679106	1.181
015	1.11695	1.18101519	0.37969924	
4	LR Without Class Balancing	0.001	0.65695065	1.23075
5439	1.13971	1.23075	0.372180451	
5	Linear SVM	0.01	0.6810095	1.2159
251	1.1600574	1.21592514	0.3759398	
6	RF With One hot Encoding	1000	0.634637	1.147

359		1.1265281		1.147359		0.3890977			
	7		RF With Response Coding		100		0.05246296		1.2994
3982		1.3014627		1.2994398		0.46240601			
	8		Stacking Classifier		0.1		0.669775		1.146
37		1.1752892		1.1752892		0.3759398			
	9		Maximum Voting Classifier		NaN		0.8992452		1.195
844		1.1921558		1.1921558		0.3834586			
<hr/>									

TASK 4.2-----

In [55]:

```
FEATUREtfidfw2v=pd.read_csv("FEATUREtfidfw2v.csv",encoding='latin-1')
```

In [56]:

```
FEATUREtfidfw2vTEST=pd.read_csv("FEATUREtfidfw2vTEST.csv",encoding='latin-1')
```

In [58]:

```
FEATUREtfidfw2vCV=pd.read_csv("FEATUREtfidfw2vCV.csv",encoding='latin-1')
```

In [59]:

```
df1 = FEATUREtfidfw2v.drop(['q1_feats_m','ID','Gene','Variation','Class','TEXT'],axis=1)
df2 = FEATUREtfidfw2vTEST.drop(['q1_feats_m','ID','Gene','Variation','Class','TEXT'],axis=1)
df3 = FEATUREtfidfw2vCV.drop(['q1_feats_m','ID','Gene','Variation','Class','TEXT'],axis=1)
```

In [314]:

```
train_x_onehotCoding = df1
test_x_onehotCoding = df2
cv_x_onehotCoding = df3
```

In [61]:

```
df4 = FEATUREtfidfw2v['q1_feats_m']
```

In [62]:

```
type(df4)
```

Out[62]:

```
pandas.core.series.Series
```

In [63]:

```
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
```

In [346]:

```
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [347]:

```
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 [348]:

```
train_text_feature_onehotCoding
train_x_onehotCoding = hstack((train_text_feature_onehotCoding, df1, train_variation_feature_onehotCoding))

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_x_onehotCoding = hstack((test_text_feature_onehotCoding, df2, test_variation_feature_onehotCoding))

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_x_onehotCoding = hstack((cv_text_feature_onehotCoding, df3, cv_variation_feature_onehotCoding))
cv_x_onehotCoding = normalize(cv_x_onehotCoding, axis=0)
```

In [349]:

```
test_x_onehotCoding = normalize(test_x_onehotCoding, axis=0)
train_x_onehotCoding = normalize(train_x_onehotCoding, axis=0)
```

In [350]:

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape[1])
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape[1])
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape[1])
```

One hot encoding features :

(number of data points * number of features) in train data = (2124, 131686)
 (number of data points * number of features) in test data = (665, 131686)
 (number of data points * number of features) in cross validation data = (532, 131686)

Linear regression hyperparameter tuning

In [354]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/gen
# -----
# default parameters
# 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_
    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 estimat
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

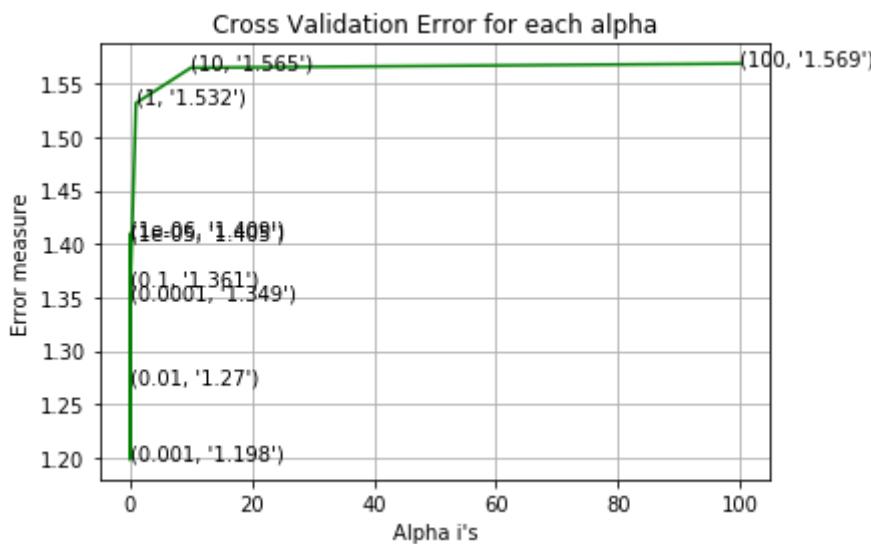
```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 1e-06
Log Loss : 1.4090992729663327
for alpha = 1e-05
Log Loss : 1.4054121255241787
for alpha = 0.0001
Log Loss : 1.34939718863081
for alpha = 0.001
Log Loss : 1.1980296380288478
for alpha = 0.01
Log Loss : 1.2698346984808229
for alpha = 0.1
Log Loss : 1.360810360325156
for alpha = 1
Log Loss : 1.5316859231099211
for alpha = 10
Log Loss : 1.5647088434811385
for alpha = 100
Log Loss : 1.5686066498665467

```



For values of best alpha = 0.001 The train log loss is: 0.5984903254737128
 For values of best alpha = 0.001 The cross validation log loss is: 1.198029
 6380288478
 For values of best alpha = 0.001 The test log loss is: 1.131241633757007

Testing the model with best hyper parameters

In [355]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c

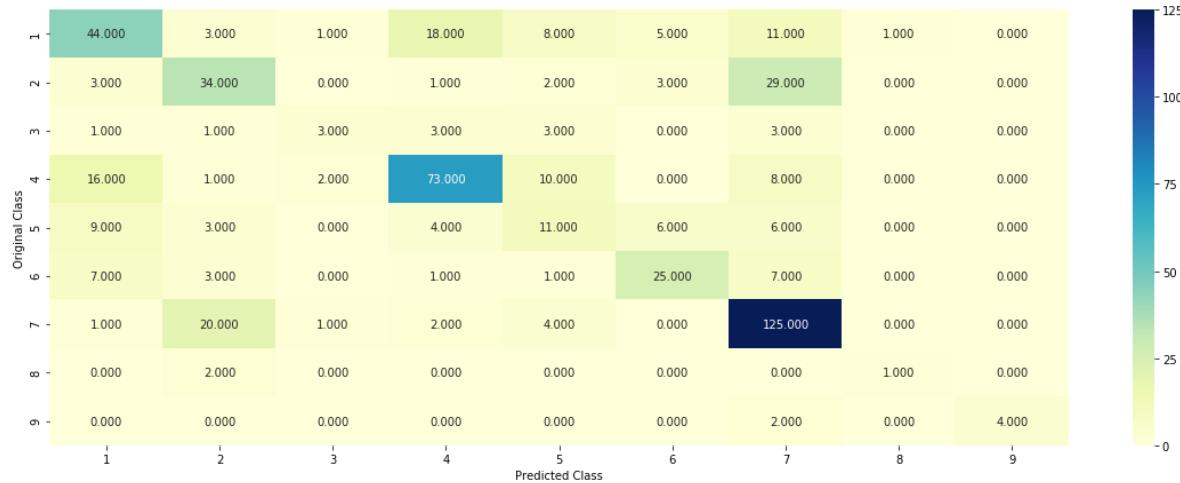
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

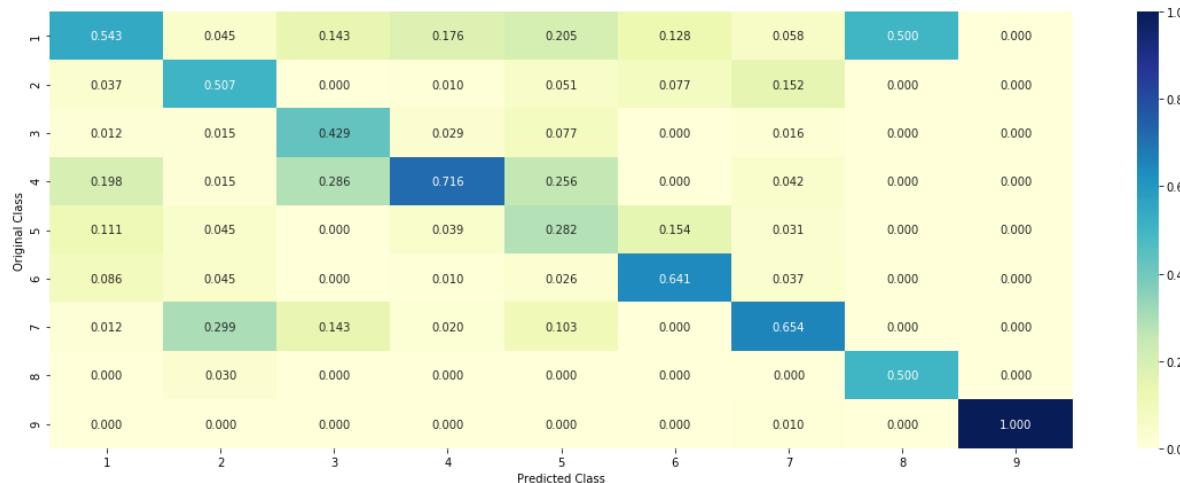
Log loss : 1.1980296380288478

Number of mis-classified points : 0.39849624060150374

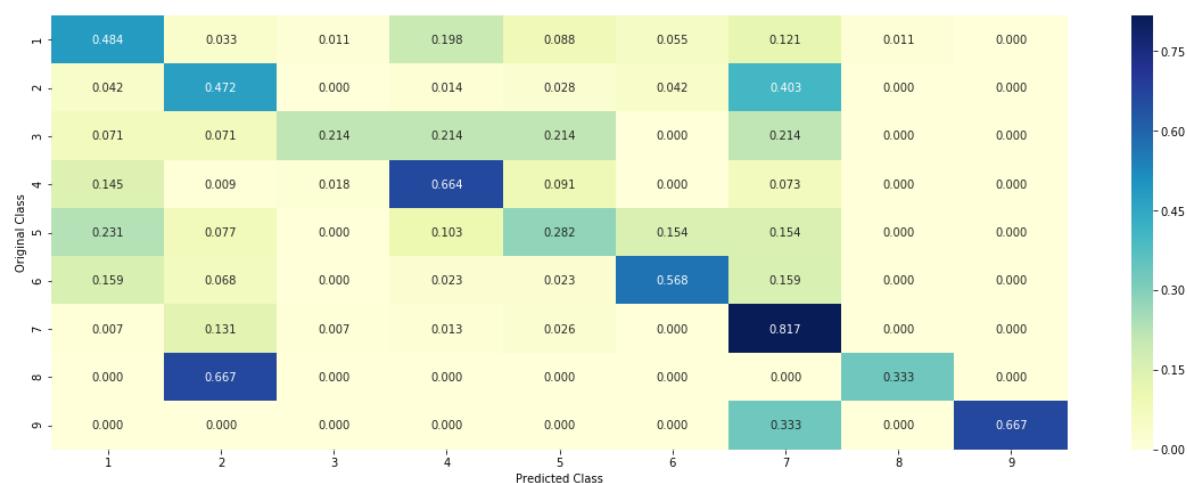
----- Confusion matrix -----



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



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



Without Class balancing

Hyper parameter tuning

In [359]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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 = [.0001 * x for x in range(1, 20)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

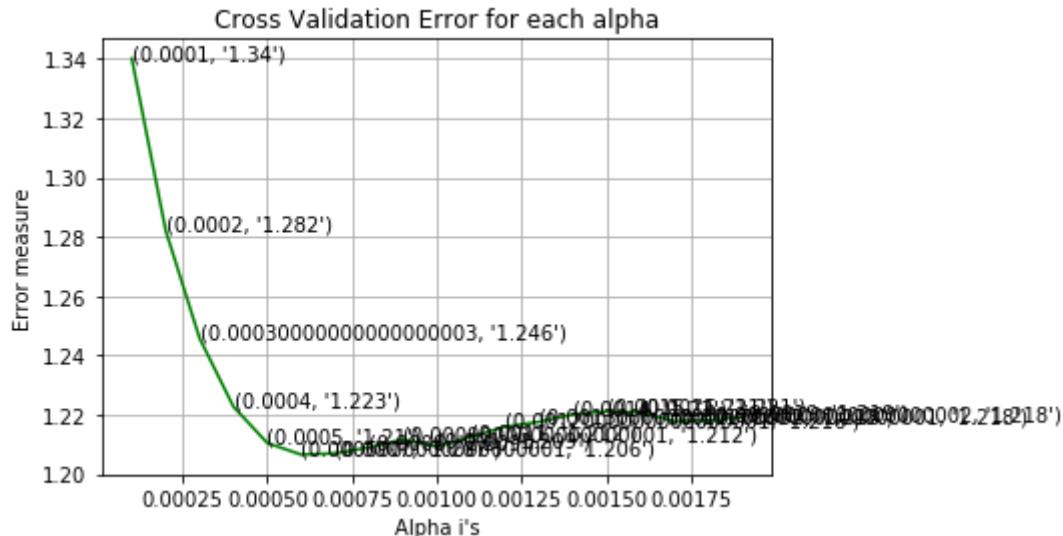


```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

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

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

for alpha = 0.0001
Log Loss : 1.3400908945968404
for alpha = 0.0002
Log Loss : 1.282070181203759
for alpha = 0.00030000000000000003
Log Loss : 1.245755661746107
for alpha = 0.0004
Log Loss : 1.2226532202316003
for alpha = 0.0005
Log Loss : 1.2104122746054968
for alpha = 0.00060000000000000001
Log Loss : 1.2064890352376794
for alpha = 0.0007
Log Loss : 1.2069327453046954
for alpha = 0.0008
Log Loss : 1.20892391269454
for alpha = 0.00090000000000000001
Log Loss : 1.2116161386211428
for alpha = 0.001
Log Loss : 1.2092086777368765
for alpha = 0.0011
Log Loss : 1.2123969325173152
for alpha = 0.00120000000000000001
Log Loss : 1.2158738951571295
for alpha = 0.00130000000000000002
Log Loss : 1.217516630200302
for alpha = 0.0014
Log Loss : 1.2200138555431896
for alpha = 0.0015
Log Loss : 1.2211254713344293
for alpha = 0.0016
Log Loss : 1.2207136421483051
for alpha = 0.00170000000000000001
Log Loss : 1.2178064981926535
for alpha = 0.00180000000000000002
Log Loss : 1.2180850534061307
for alpha = 0.0019
Log Loss : 1.2191429991365021
```



For values of best alpha = 0.0006000000000000001 The train log loss is: 0.6370201485851975

For values of best alpha = 0.0006000000000000001 The cross validation log loss is: 1.2064890352376794

For values of best alpha = 0.0006000000000000001 The test log loss is: 1.1457121283162601

Testing model with best hyper parameters

In [360]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# 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, c

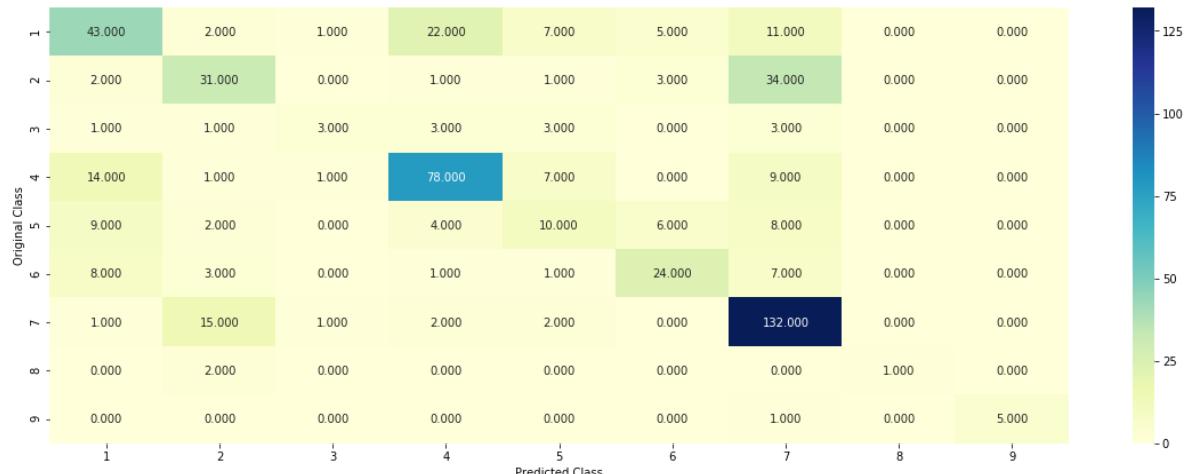
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

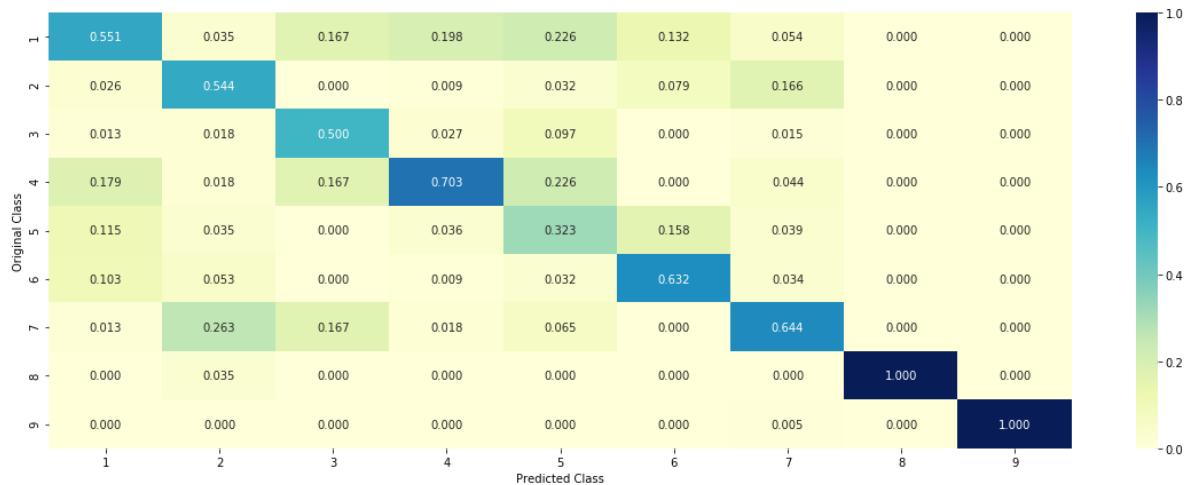
Log loss : 1.2064890352376794

Number of mis-classified points : 0.38533834586466165

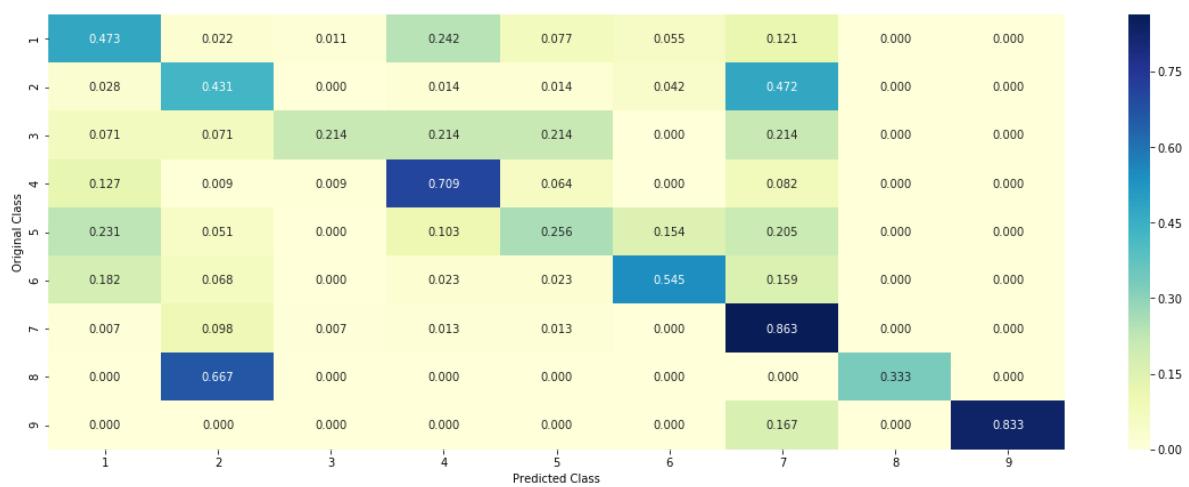
----- Confusion matrix -----



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



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



Linear SVM

In [362]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/gen
# -----
# default parameters
# 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 = [.010 * x for x in range(1, 10)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hinge', rand
    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 estimat
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

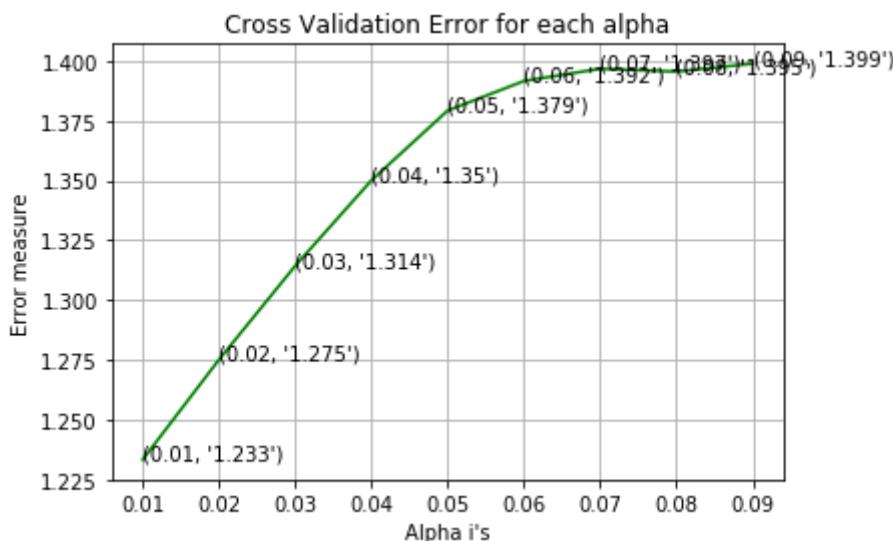
```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 0.01
Log Loss : 1.2334308730548527
for alpha = 0.02
Log Loss : 1.275143632052075
for alpha = 0.03
Log Loss : 1.3143423950997135
for alpha = 0.04
Log Loss : 1.3500657514960692
for alpha = 0.05
Log Loss : 1.3792207275033916
for alpha = 0.06
Log Loss : 1.3915634311851948
for alpha = 0.07
Log Loss : 1.3966872495055618
for alpha = 0.08
Log Loss : 1.395495514388444
for alpha = 0.09
Log Loss : 1.3988311934541604

```



For values of best alpha = 0.01 The train log loss is: 0.6081885433542652
 For values of best alpha = 0.01 The cross validation log loss is: 1.2698346984808229
 For values of best alpha = 0.01 The test log loss is: 1.171607720392886

Testing the model with best hyper parameters

In [363]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='h
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c

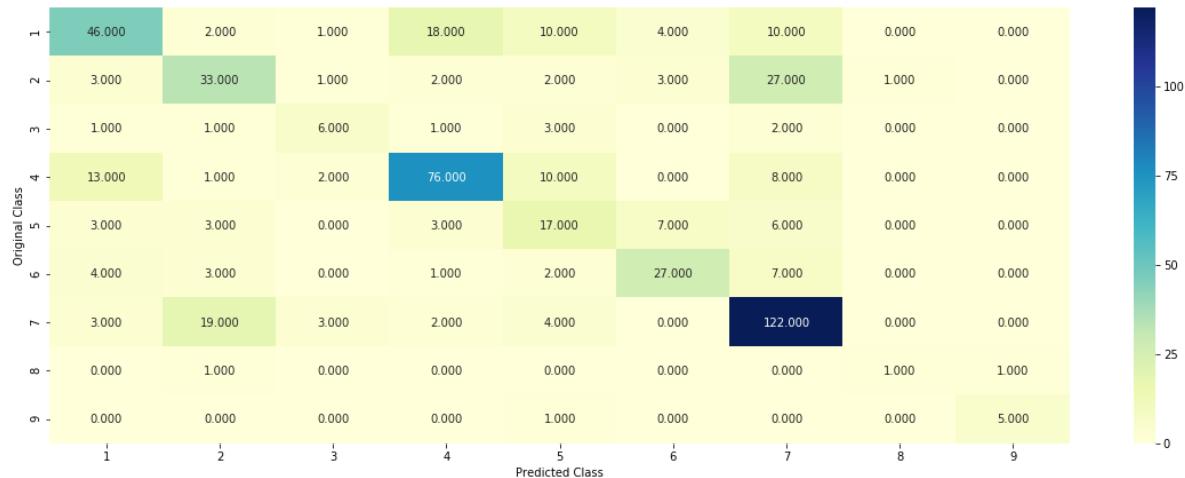
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

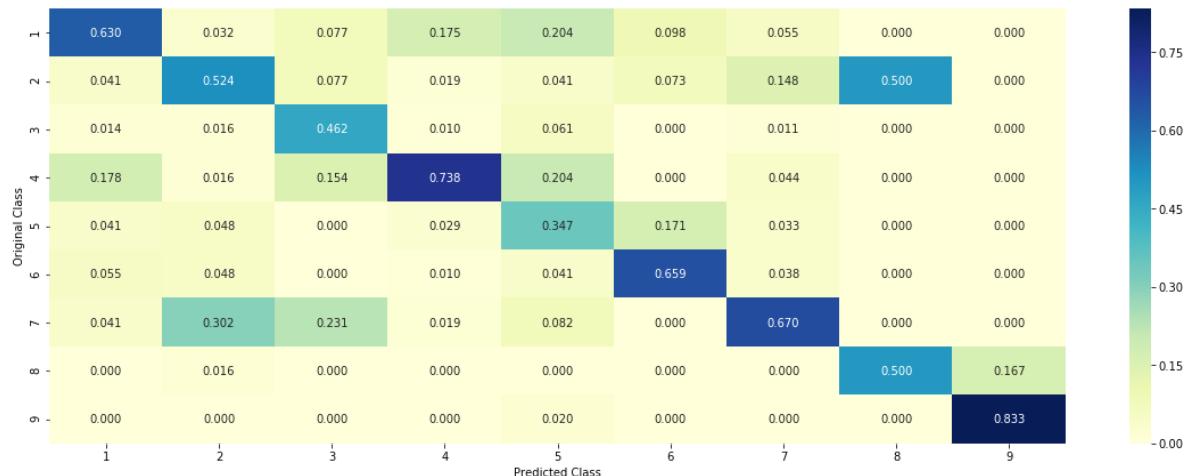
Log loss : 1.2334308730548527

Number of mis-classified points : 0.37406015037593987

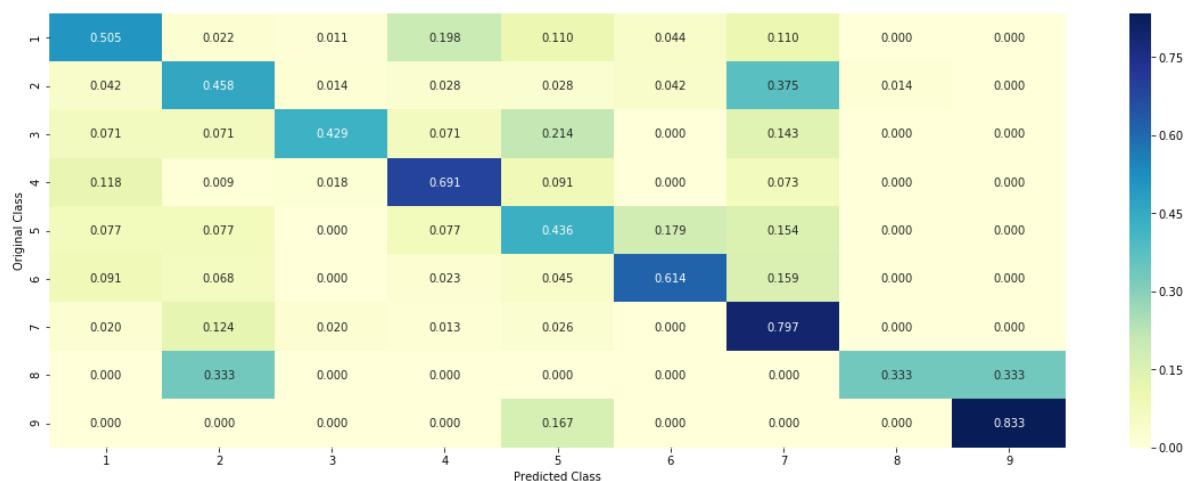
----- Confusion matrix -----



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



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



In [365]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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 = [.010 * x for x in range(1, 10)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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



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

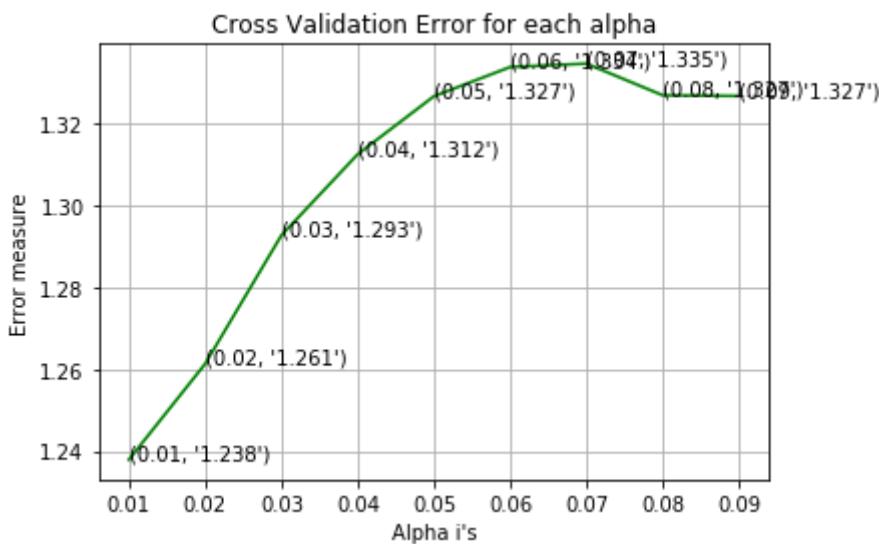
```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 0.01
Log Loss : 1.2378930078415091
for alpha = 0.02
Log Loss : 1.2612875630673706
for alpha = 0.03
Log Loss : 1.2926853283903503
for alpha = 0.04
Log Loss : 1.3124655842648842
for alpha = 0.05
Log Loss : 1.3266522905075564
for alpha = 0.06
Log Loss : 1.33378356922595
for alpha = 0.07
Log Loss : 1.3346231431653819
for alpha = 0.08
Log Loss : 1.3268882949614995
for alpha = 0.09
Log Loss : 1.3267143621482378

```



For values of best alpha = 0.01 The train log loss is: 0.6013319787567547
 For values of best alpha = 0.01 The cross validation log loss is: 1.271518198647418
 For values of best alpha = 0.01 The test log loss is: 1.1670885369325852

Testing model with best hyper parameters

In [367]:

```
#SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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='hinge', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c)

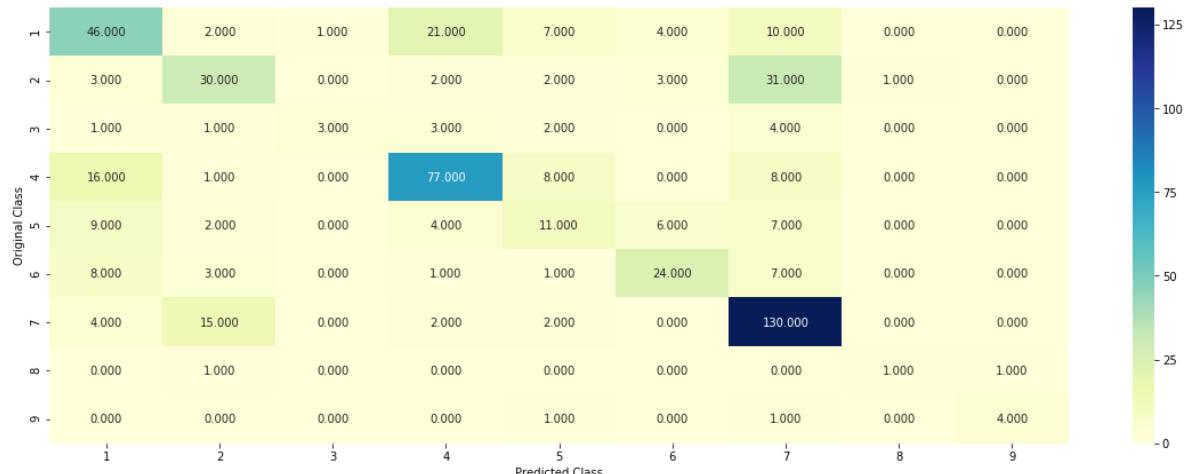
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

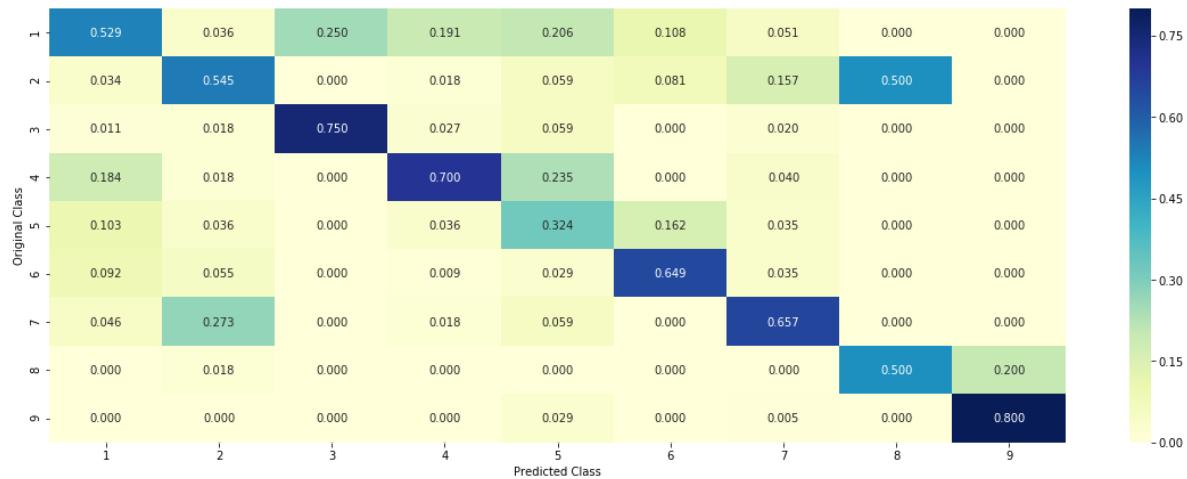
Log loss : 1.2378930078415091

Number of mis-classified points : 0.38721804511278196

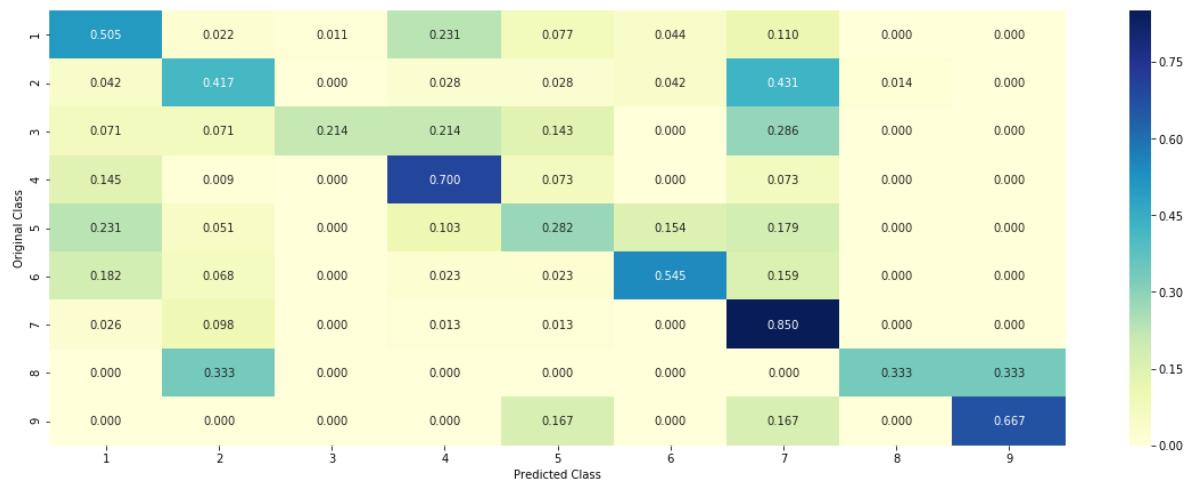
----- Confusion matrix -----



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



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



In [99]:

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

# Names of models
names =['LR With Class Balancing','LR Without Class Balancing','Linear SVM balanced','Linear SVM UNbalanced']

# Training Loss
train_loss = [0.59849032,0.637020,0.60818854 ,0.60133197 ]

# Cross Validation Loss
cv_loss = [1.198029638, 1.20648903,1.26983469,1.2715181 ]
# Test Loss
test_loss = [1.13124163,1.1457121,1.1716077,1.1670885 ]

log_loss=[1.19802963 ,1.20648903,1.23343087305,1.23789300 ]

alpha=[0.001, 0.0006 ,.01,.01, ]

# Percentage Misclassified points
misclassified = [0.39849624,0.38533834,0.374060150,0.38721804 ]

numbering = [1,2,3,4]

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
ss	Test_loss	log_loss	Misclassified(%)	
1	LR With Class Balancing	0.001	0.59849032	1.198029638
2	LR Without Class Balancing	0.0006	0.637020	1.20648903
3	Linear SVM balanced	0.01	0.60818854	1.26983469
4	Linear SVM UNbalanced	0.01	0.60133197	1.2715181

In []:

In [64]:

```
train_x_onehotCoding = hstack((train_x_onehotCoding , df1)).tocsr()

test_x_onehotCoding = hstack((test_x_onehotCoding, df2)).tocsr()

cv_x_onehotCoding = hstack((cv_x_onehotCoding, df3)).tocsr()

cv_x_onehotCoding = normalize(cv_x_onehotCoding, axis=0)
test_x_onehotCoding = normalize(test_x_onehotCoding, axis=0)
train_x_onehotCoding = normalize(train_x_onehotCoding, axis=0)
```

In [65]:

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

One hot encoding features :
(number of data points * number of features) in train data = (2124, 209698)
(number of data points * number of features) in test data = (665, 209698)
(number of data points * number of features) in cross validation data = (53
2, 209698)

. Logistic Regression

With Class balancing

. Hyper parameter tuning

In [82]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/gen
# -----
# default parameters
# 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 = [.00010 * x for x in range(1, 15)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_
    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 estimat
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

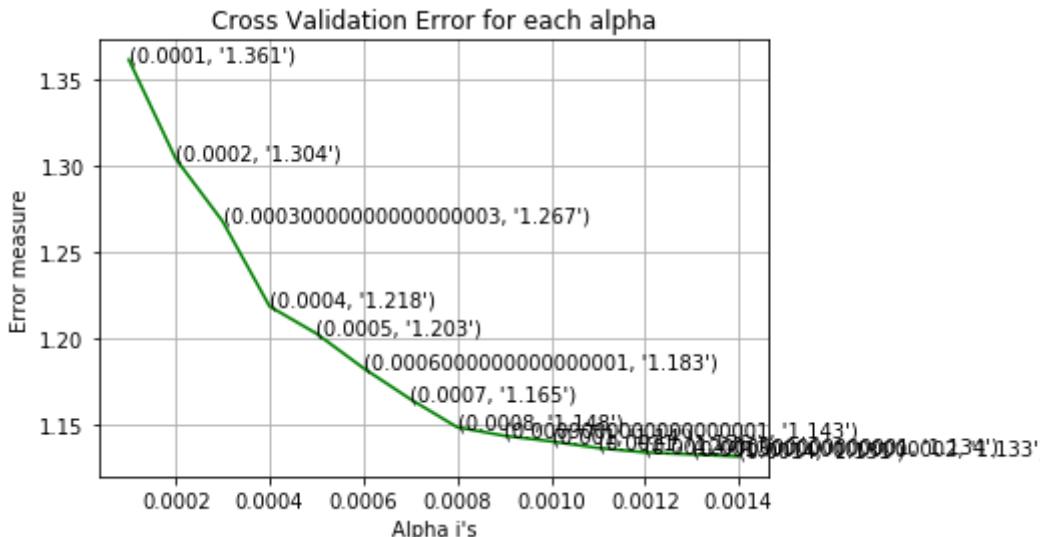
```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_)

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

for alpha = 0.0001
Log Loss : 1.36103062126
for alpha = 0.0002
Log Loss : 1.30369422653
for alpha = 0.00030000000000000003
Log Loss : 1.2674784388
for alpha = 0.0004
Log Loss : 1.21832943827
for alpha = 0.0005
Log Loss : 1.20259280734
for alpha = 0.00060000000000000001
Log Loss : 1.18260598674
for alpha = 0.0007
Log Loss : 1.16467531687
for alpha = 0.0008
Log Loss : 1.14838330014
for alpha = 0.00090000000000000001
Log Loss : 1.14347154335
for alpha = 0.001
Log Loss : 1.14023231745
for alpha = 0.0011
Log Loss : 1.13638865264
for alpha = 0.00120000000000000001
Log Loss : 1.13384211872
for alpha = 0.00130000000000000002
Log Loss : 1.13268536913
for alpha = 0.0014
Log Loss : 1.13145971591

```



For values of best alpha = 0.0014 The train log loss is: 0.659144571696

For values of best alpha = 0.0014 The cross validation log loss is: 1.13145

971591

For values of best alpha = 0.0014 The test log loss is: 1.15641103571

Testing the model with best hyper paramters

In [83]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c

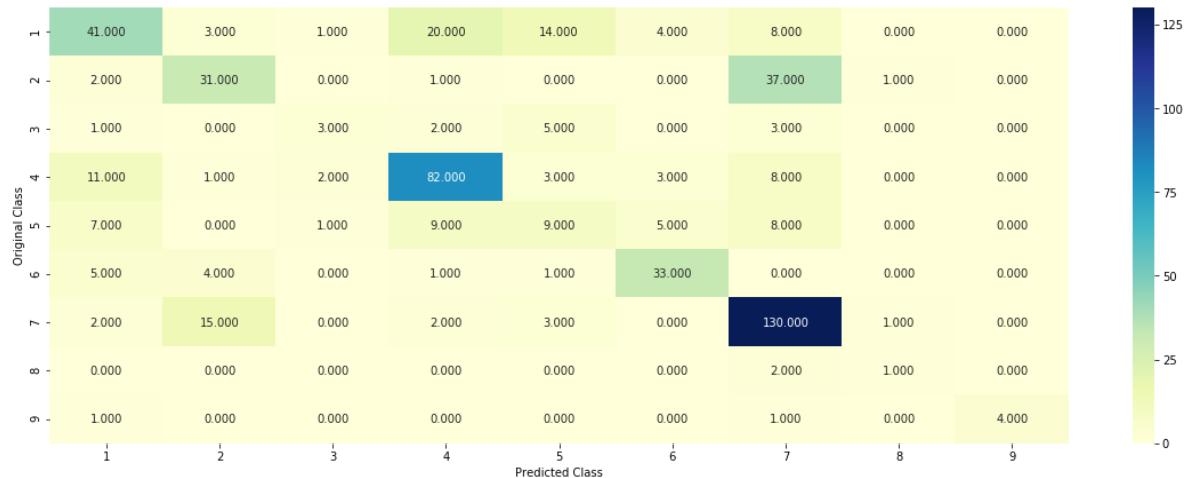
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

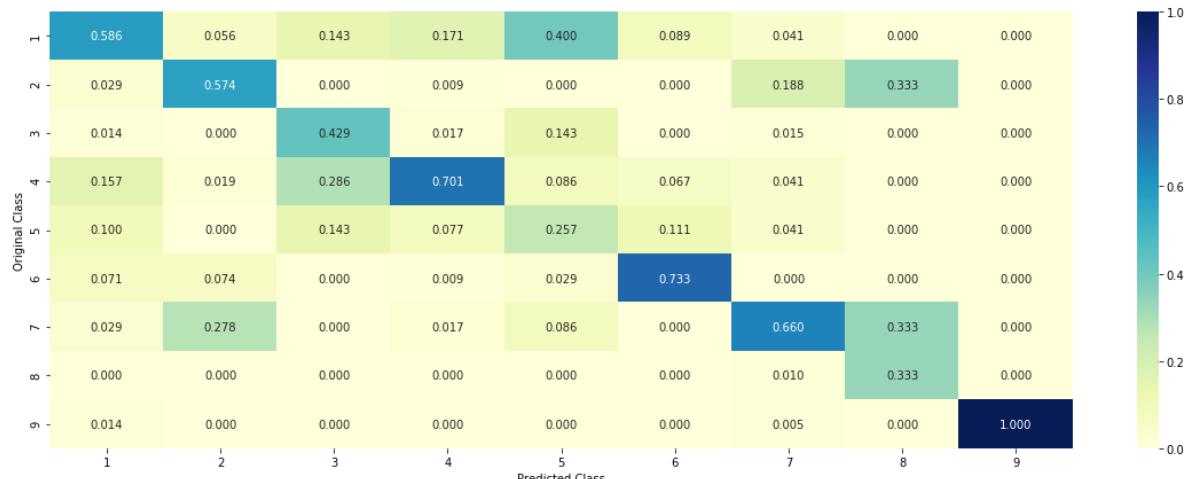
Log loss : 1.13145971591

Number of mis-classified points : 0.37218045112781956

----- Confusion matrix -----



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



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



Correctly Classified point

In [84]:

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

Predicted Class : 4

Predicted Class Probabilities: [[0.03 0.0082 0.0094 0.6384 0.2989 0.

0.0066 0.0014 0.0054 0.0018]]

Actual Class : 4

. Incorrectly Classified point

In [85]:

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

Predicted Class : 7

Predicted Class Probabilities: [[0.0519 0.22 0.0203 0.0288 0.0294 0.

0.006 0.6253 0.0129 0.0054]]

Actual Class : 6

Without Class balancing

Hyper parameter tuning

In [73]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



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



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

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



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

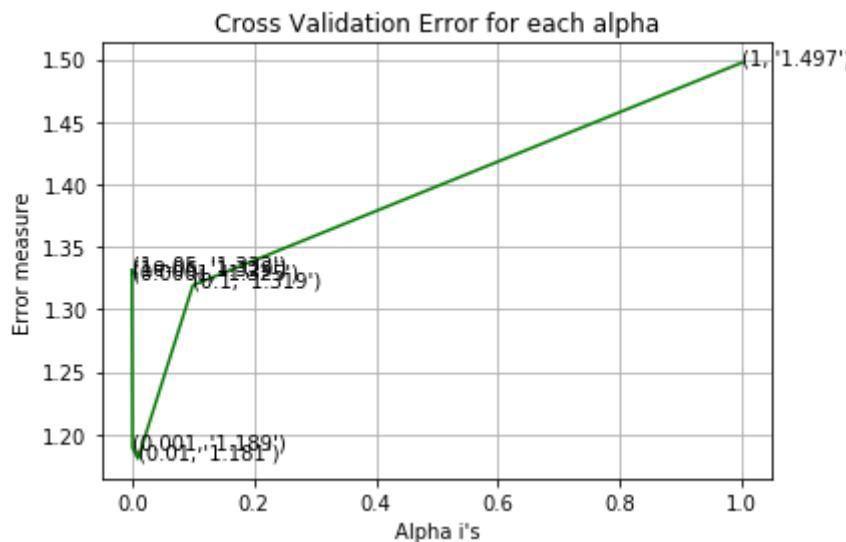
```

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:")
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_)

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

for alpha = 1e-06
Log Loss : 1.32878896508
for alpha = 1e-05
Log Loss : 1.33168781454
for alpha = 0.0001
Log Loss : 1.32473463181
for alpha = 0.001
Log Loss : 1.18918671648
for alpha = 0.01
Log Loss : 1.18055764631
for alpha = 0.1
Log Loss : 1.31924459546
for alpha = 1
Log Loss : 1.49738616915

```



```

For values of best alpha =  0.01 The train log loss is: 0.604869197517
For values of best alpha =  0.01 The cross validation log loss is: 1.1805576
4631
For values of best alpha =  0.01 The test log loss is: 1.18231017511

```

Testing model with best hyper parameters

In [74]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# 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, c

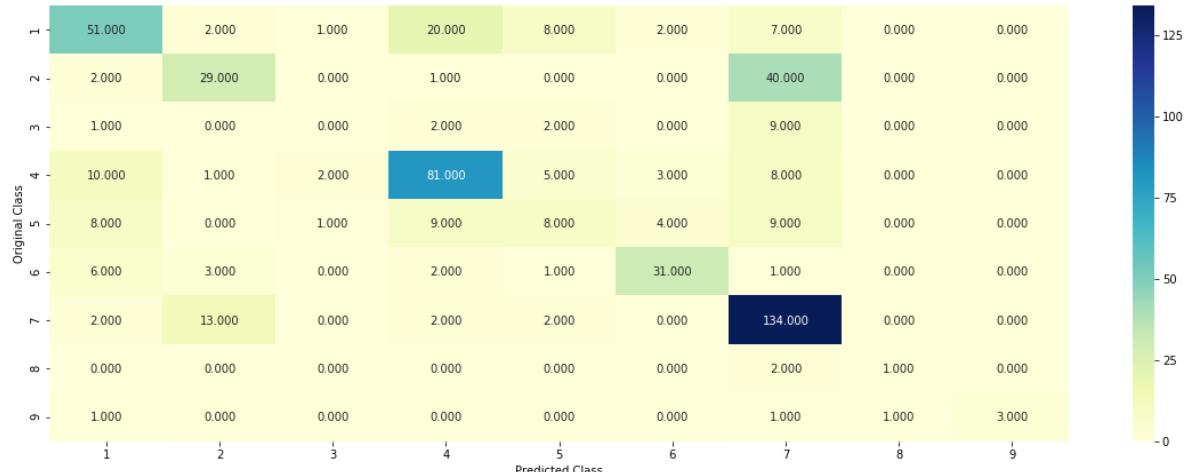
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

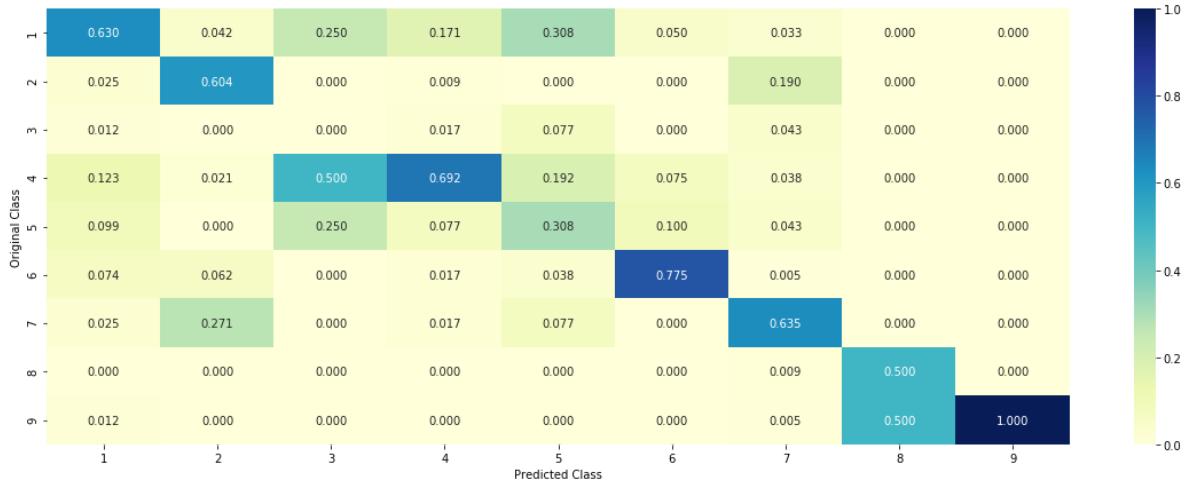
Log loss : 1.18055764631

Number of mis-classified points : 0.36466165413533835

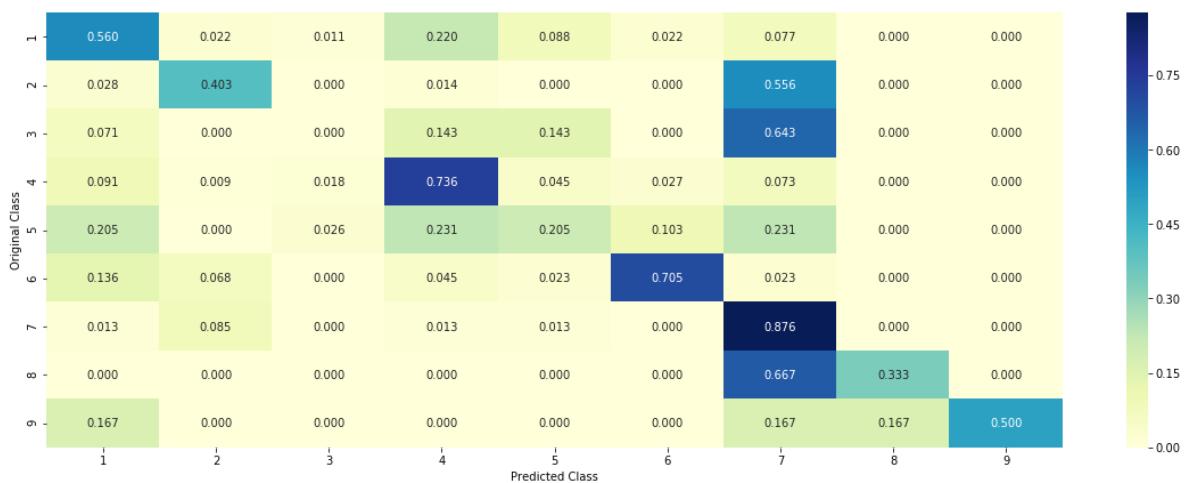
----- Confusion matrix -----



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



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



Feature Importance, Correctly Classified point

In [75]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class : ", predicted_cls[0])
print("Predicted Class Probabilities: ", np.round(sig_clf.predict_proba(test_x_onehotCoding[0]), 3))
print("Actual Class : ", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 1.3500000e-02   8.0000000e-04   3.0000000e-04
  0.000e+00   5.5940000e-01
   4.2460000e-01   1.0000000e-03   2.0000000e-04   1.0000000e-04
   0.0000000e+00]]
Actual Class : 4
```

Feature Importance, Incorrectly Classified point

In [76]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[ 2.1000000e-02   1.1610000e-01   9.00000
000e-04   1.7700000e-02
 4.7000000e-03   1.8000000e-03   8.3710000e-01   7.0000000e-04
 1.0000000e-04]]
Actual Class : 6
-----
```

Linear Support Vector Machines

Hyper parameter tuning

In [77]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



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



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

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



```
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge')
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_)

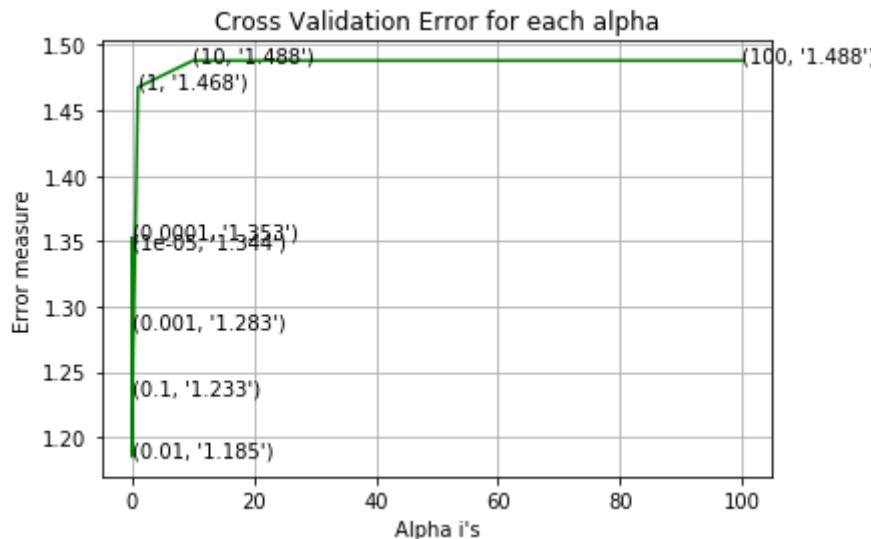
# Variables that will be used in the end to make comparison table of all models
svm_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_
svm_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=
svm_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding),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

```

```

for C = 1e-05
Log Loss : 1.3441855966
for C = 0.0001
Log Loss : 1.35256243482
for C = 0.001
Log Loss : 1.28332515272
for C = 0.01
Log Loss : 1.18502464403
for C = 0.1
Log Loss : 1.23270408332
for C = 1
Log Loss : 1.46763742686
for C = 10
Log Loss : 1.48802628889
for C = 100
Log Loss : 1.48802629003

```



```

For values of best alpha = 0.01 The train log loss is: 0.705455234695
For values of best alpha = 0.01 The cross validation log loss is: 1.1850246
4403
For values of best alpha = 0.01 The test log loss is: 1.21357810253

```

Testing model with best hyper parameters

In [78]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video Link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM
# -----
```

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

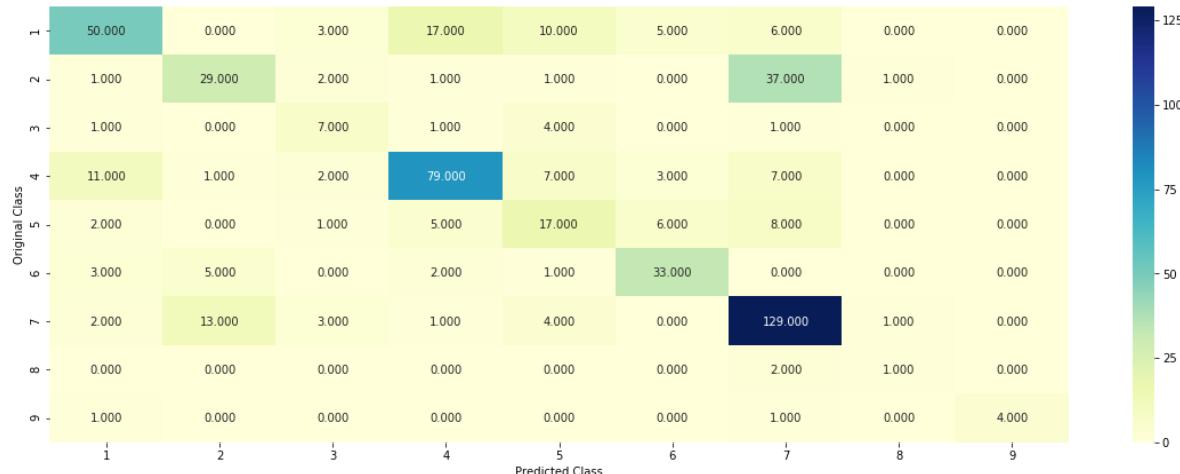
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

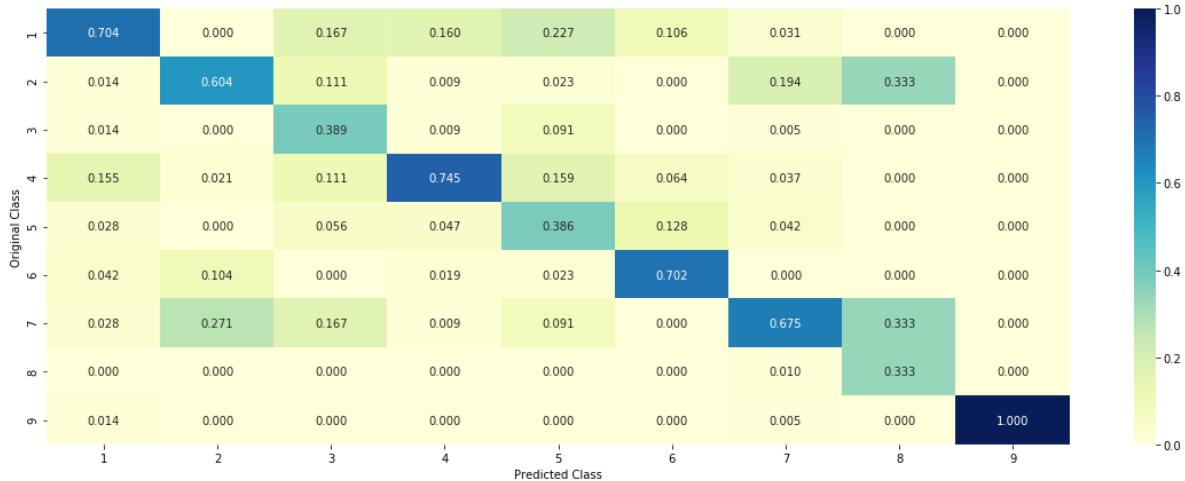
Log loss : 1.18502464403

Number of mis-classified points : 0.34398496240601506

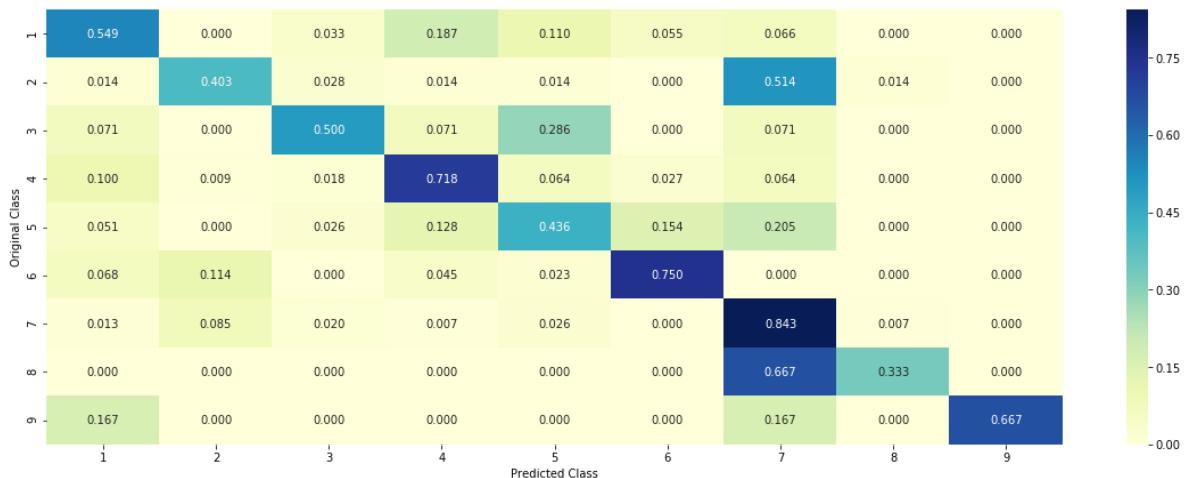
----- Confusion matrix -----



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



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



Feature Importance

For Correctly classified point

In [79]:

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

Predicted Class : 4

Predicted Class Probabilities: [[0.0658 0.0246 0.0062 0.5273 0.3465 0.0039 0.0183 0.0057 0.0017]]

Actual Class : 4

For Incorrectly classified point

In [80]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0795  0.2008  0.0111  0.0688  0.0364  0.
0.085  0.5778  0.0131  0.004 ]]
Actual Class : 6
-----
```

In [97]:

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

# Names of models
names = ['LR With Class Balancing', 'LR Without Class Balancing', 'Linear SVM ']

# Training Loss
train_loss = [0.659144, 0.604869, 0.705455]

# Cross Validation Loss
cv_loss = [1.13145978, 1.180557, 1.185024]
# Test Loss
test_loss = [1.156411, 1.18231017, 1.2135781]

log_loss=[1.1314597 ,1.180557,1.1850246]

alpha=[0.0014, 0.01 ,.01 ]

# Percentage Misclassified points
misclassified = [0.37218045, 0.3646616541, 0.3439849624]

numbering = [1,2,3]

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
s	Test_loss	log_loss	Misclassified(%)	
978	LR With Class Balancing	0.0014	0.659144	1.13145
57	LR Without Class Balancing	0.01	0.604869	1.1805
24	Linear SVM	0.01	0.705455	1.1850
		0.37218045	0.3646616541	0.3439849624

TASK4.3

Apply models with TfidfVectorizer(ngram_range=(1, 4)) to reduce log loss.

In [65]:

```
# Tfidf encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(ngram_range=(1, 4))
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 [66]:

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

Out[66]:

```
2219      PTEN
334       ROS1
745       ERBB2
3287      RET
1941      CARD11
Name: Gene, dtype: object
```

In [67]:

```
print("train_gene_feature_Tfidf is converted feature using Tfidf encoding method. The shape
```

```
train_gene_feature_Tfidf is converted feature using Tfidf encoding method. The shape of gene feature: (2124, 242)
```

In [68]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(ngram_range=(1, 4))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [69]:

```
print("train_variation_feature_Tfidf is converted feature using the TfidfVectorizer encoding
```

```
train_variation_feature_Tfidf is converted feature using the TfidfVectorizer encoding method. The shape of Variation feature: (2124, 2077)
```

In [71]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,ngram_range=(1, 4))
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)
train_textfea_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 occurred
text_fea_dict = dict(zip(list(train_text_features),train_textfea_counts))

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

Total number of unique words in train data : 2947188

In [73]:

```
dict_list = []
# dict_list =[] contains 9 dictionaries 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 [74]:

```
# 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 [75]:

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

In [76]:

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

IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:

```
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [77]:

```
# 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))
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding))
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 [78]:

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

One hot encoding features :
(number of data points * number of features) in train data = (2124, 294950
7)
(number of data points * number of features) in test data = (665, 2949507)
(number of data points * number of features) in cross validation data = (53
2, 2949507)

In [79]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCod
print("(number of data points * number of features) in test data = ", test_x_responseCoding
print("(number of data points * number of features) in cross validation data =", cv_x_respo
```

Response encoding features :
(number of data points * number of features) in train data = (2124, 27)
(number of data points * number of features) in test data = (665, 27)
(number of data points * number of features) in cross validation data = (53
2, 27)

4.1. Base Line Model

4.1.1. Naive Bayes 4.1.1.1. Hyper parameter tuning

In [80]:

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules
# -----
# 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
# -----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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
# -----


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

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

```

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

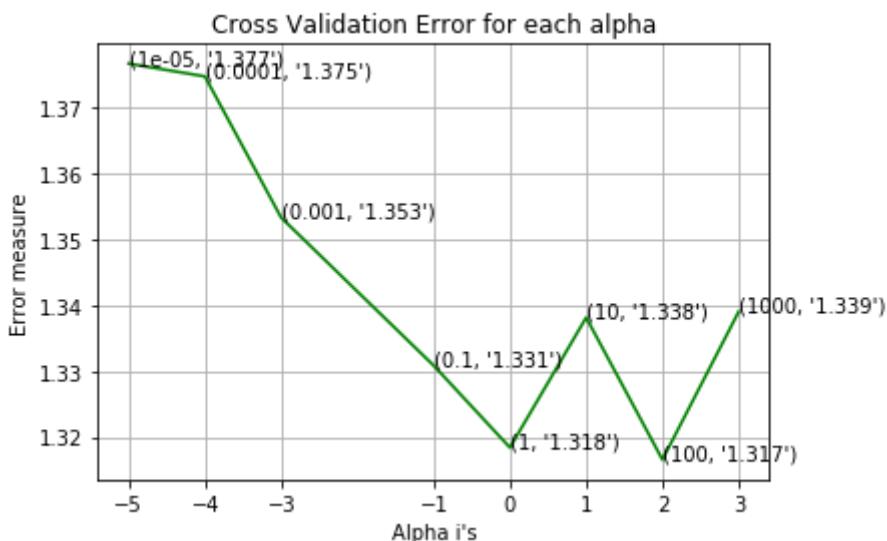
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-05
Log Loss : 1.3765800185134995
for alpha = 0.0001
Log Loss : 1.3746417736373668
for alpha = 0.001
Log Loss : 1.353256389897647
for alpha = 0.1
Log Loss : 1.3308372294752544
for alpha = 1
Log Loss : 1.3184053689881017
for alpha = 10
Log Loss : 1.3381050105533234
for alpha = 100
Log Loss : 1.316569482271147
for alpha = 1000
Log Loss : 1.3389873938247974

```



```

For values of best alpha = 100 The train log loss is: 0.8918914732930308
For values of best alpha = 100 The cross validation log loss is: 1.31656948
2271147
For values of best alpha = 100 The test log loss is: 1.2868493579545006

```

Testing the model with best hyper parameters

In [81]:

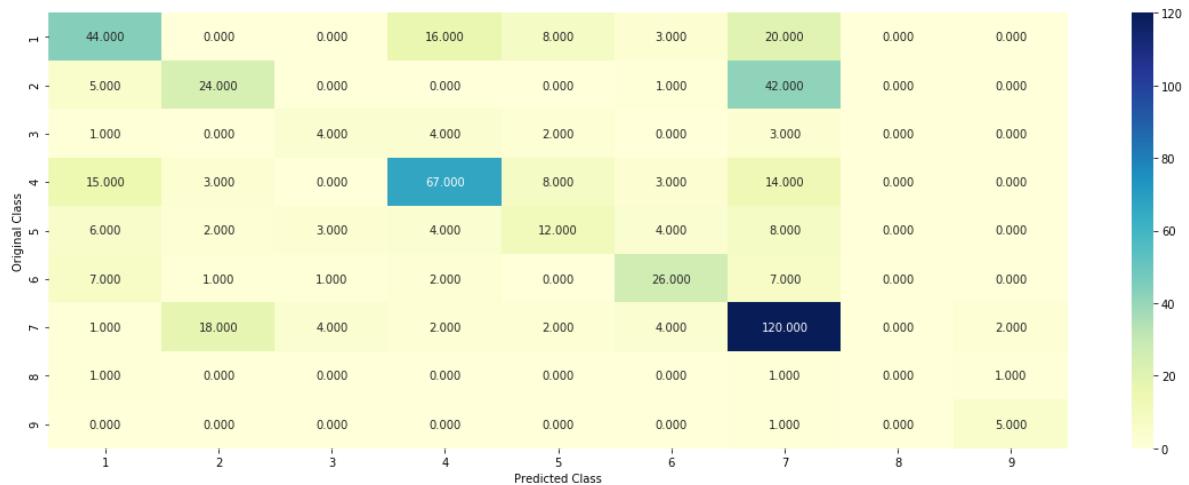
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/
# -----
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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
# -----

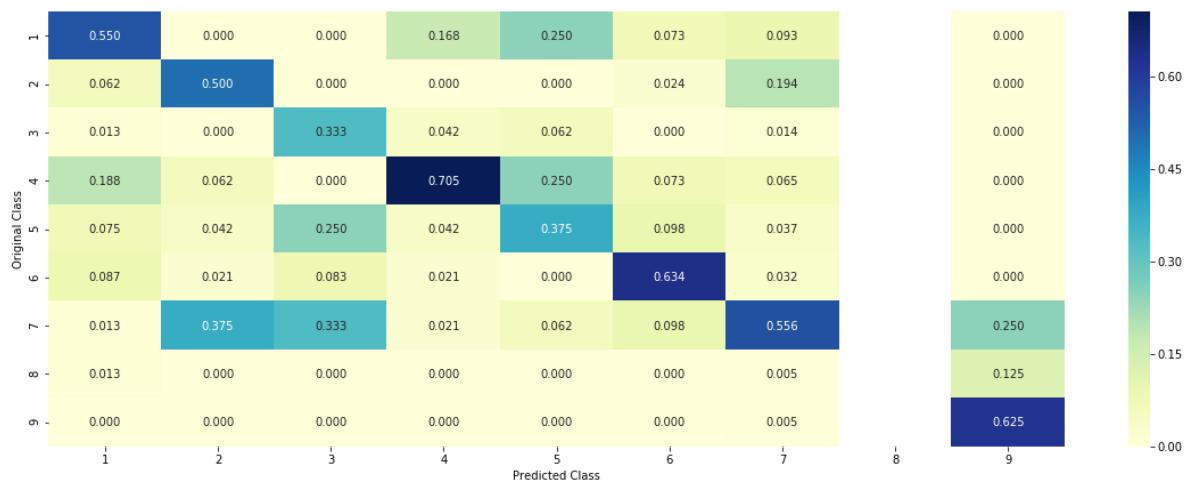

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated 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 probalites 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)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

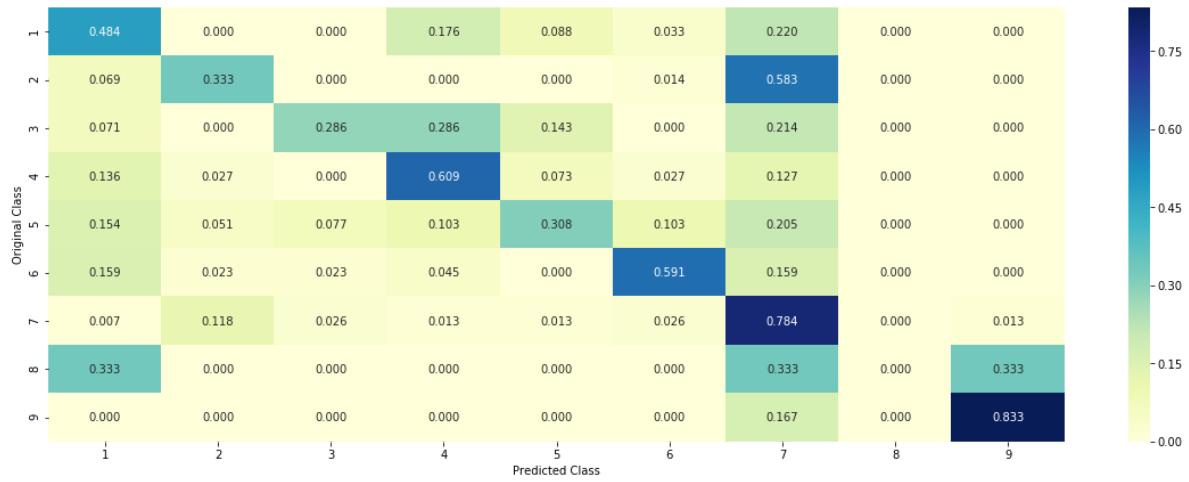
```
Log Loss : 1.316569482271147
Number of missclassified point : 0.4323308270676692
----- Confusion matrix -----
```



Precision matrix (Column Sum=1)



Recall matrix (Row sum=1)



Logistic Regression

With Class balancing

Hyper parameter tuning

In [83]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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/geom
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
        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 estimat
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```

sig_clf.fit(train_x_onehotCoding, train_y)

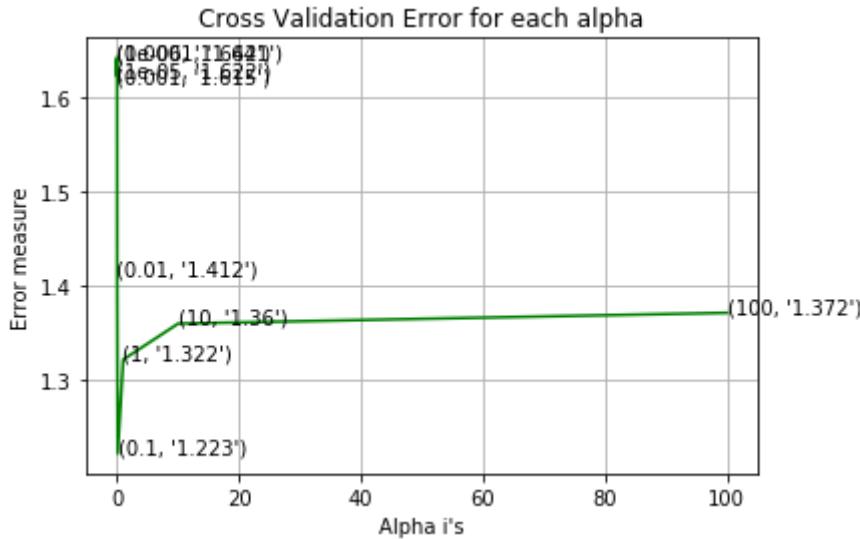
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-06
Log Loss : 1.6420163008254358
for alpha = 1e-05
Log Loss : 1.6221941970692166
for alpha = 0.0001
Log Loss : 1.6410528064015917
for alpha = 0.001
Log Loss : 1.614806758899185
for alpha = 0.01
Log Loss : 1.4121156444696379
for alpha = 0.1
Log Loss : 1.2226936189965052
for alpha = 1
Log Loss : 1.3222533531553637
for alpha = 10
Log Loss : 1.360368580800491
for alpha = 100
Log Loss : 1.3717819701178908

```



```

For values of best alpha = 0.1 The train log loss is: 0.8374172096853272
For values of best alpha = 0.1 The cross validation log loss is: 1.22269361
89965052
For values of best alpha = 0.1 The test log loss is: 1.226256153828243

```

Testing the model with best hyper paramters

In [84]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# class_weight=None, warm_start=False, average=False, n_iter=None)

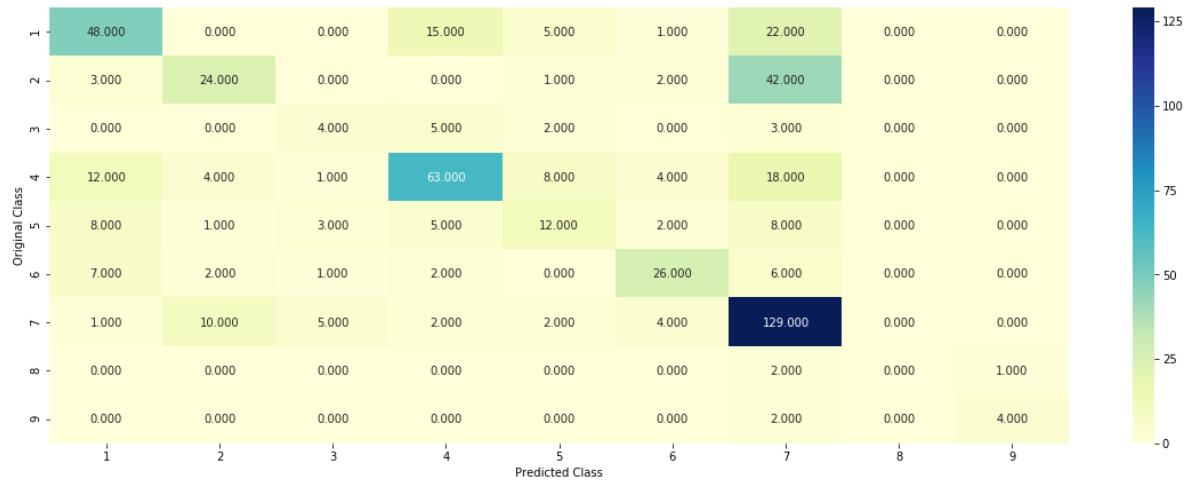
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent
# predict(X) Predict class labels for samples in X.

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
# -----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

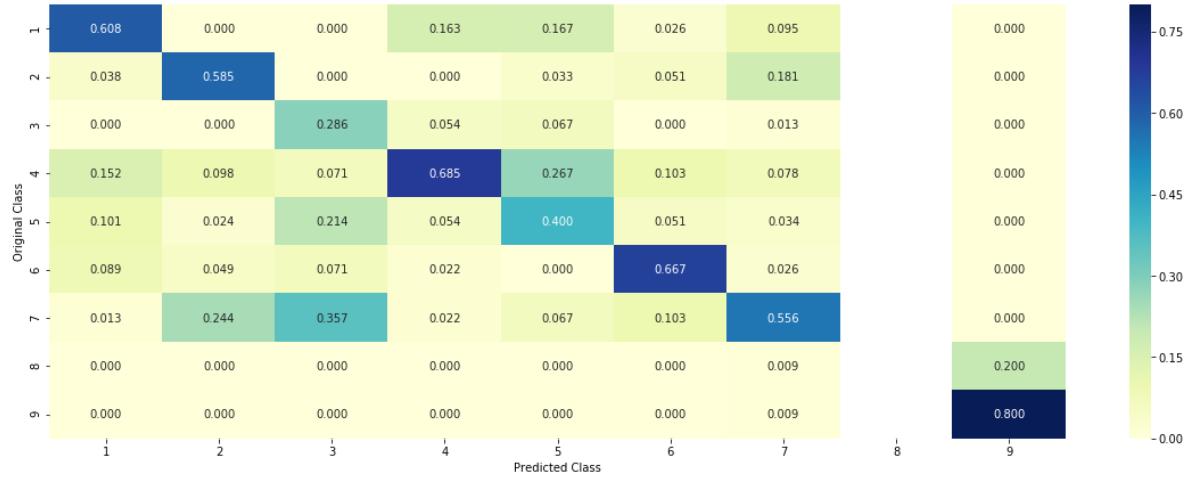
Log loss : 1.2226936189965052

Number of mis-classified points : 0.41729323308270677

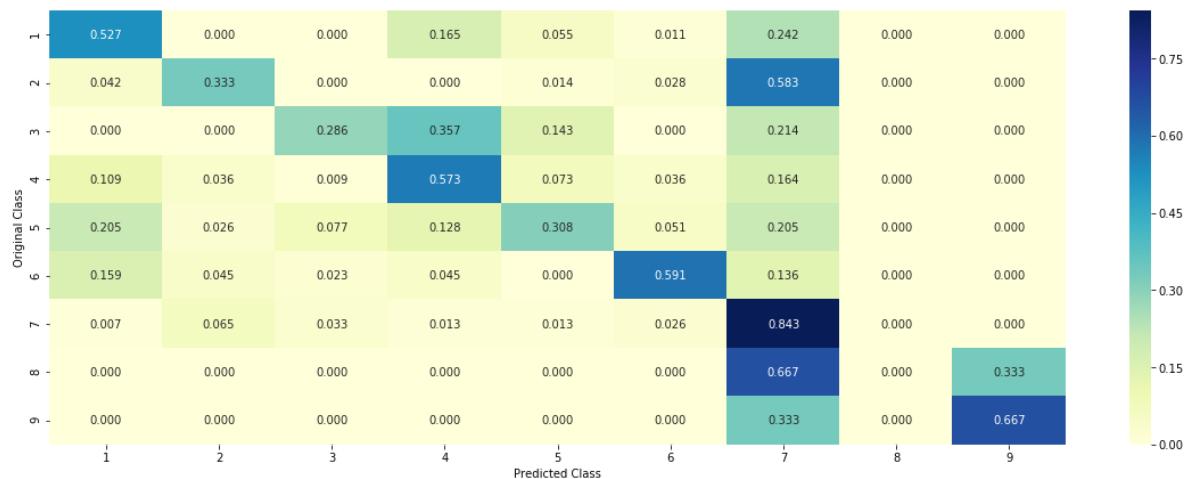
----- Confusion matrix -----



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



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



Without Class balancing

Hyper parameter tuning

In [85]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



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



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

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



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

```

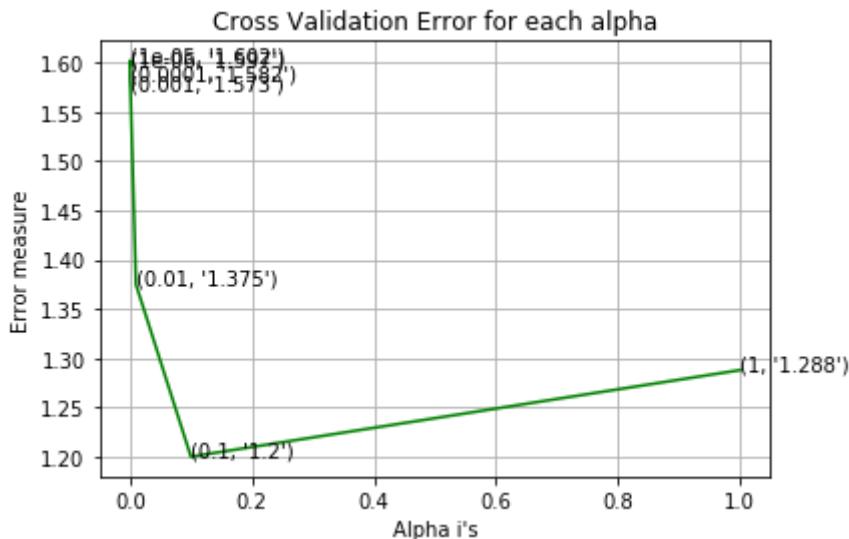
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-06
Log Loss : 1.597374224089425
for alpha = 1e-05
Log Loss : 1.601963555682457
for alpha = 0.0001
Log Loss : 1.5819362460896964
for alpha = 0.001
Log Loss : 1.5725462799020395
for alpha = 0.01
Log Loss : 1.3747990125564926
for alpha = 0.1
Log Loss : 1.2001679265647536
for alpha = 1
Log Loss : 1.2879653821087518

```



For values of best alpha = 0.1 The train log loss is: 0.8158282000564178
 For values of best alpha = 0.1 The cross validation log loss is: 1.20016792
 65647536
 For values of best alpha = 0.1 The test log loss is: 1.2106645705199262

Testing model with best hyper parameters

In [86]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

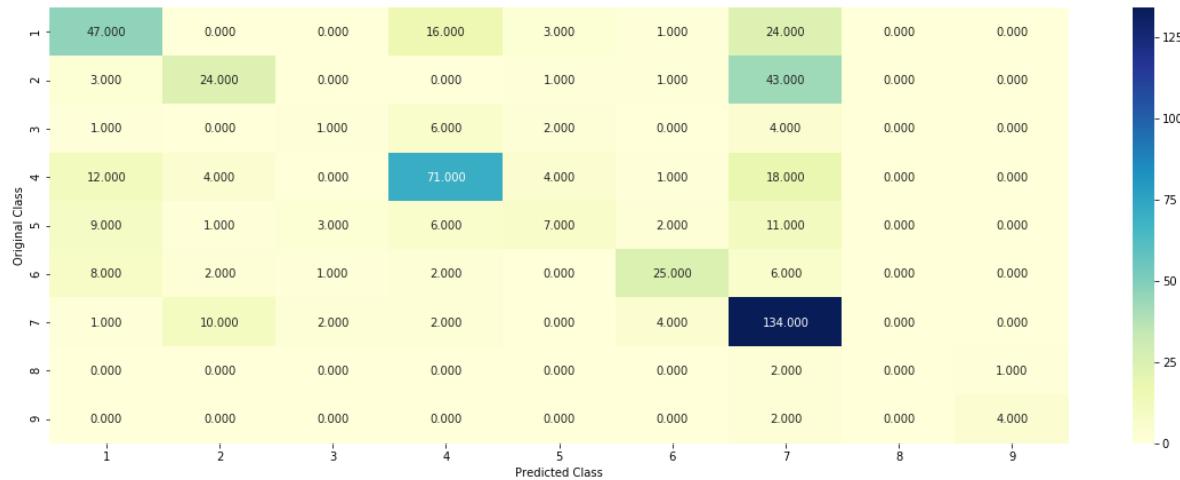
#-----
# video link:
#-----
```

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, c

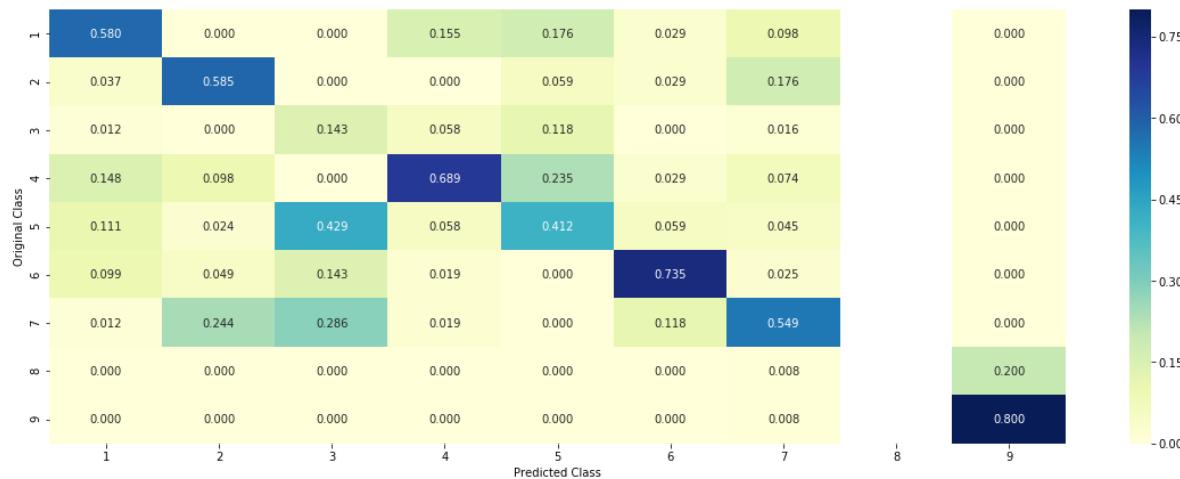
Log loss : 1.2001679265647536

Number of mis-classified points : 0.4116541353383459

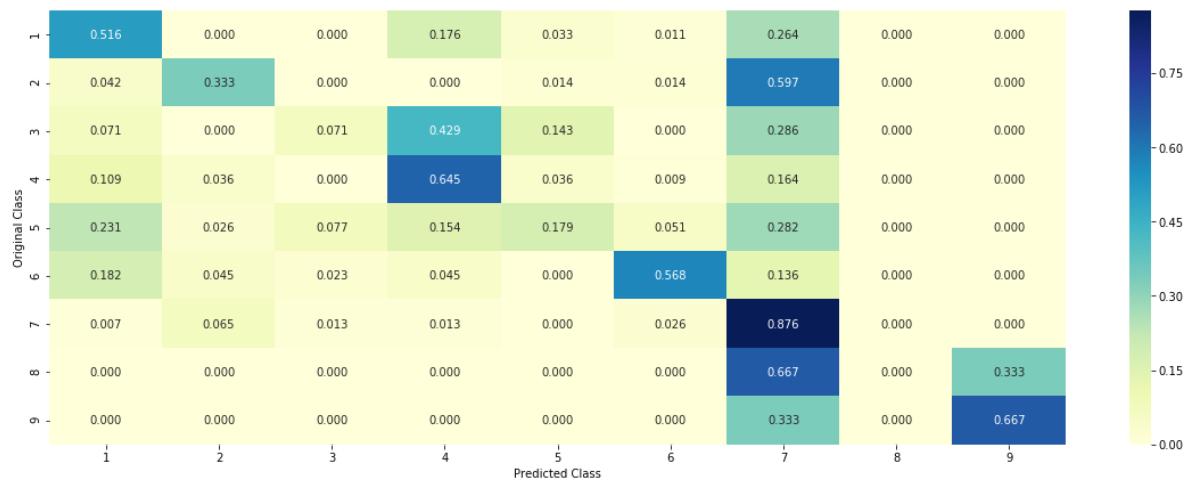
----- Confusion matrix -----



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



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



Linear Support Vector Machines

Hyper parameter tuning

In [87]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



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



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

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

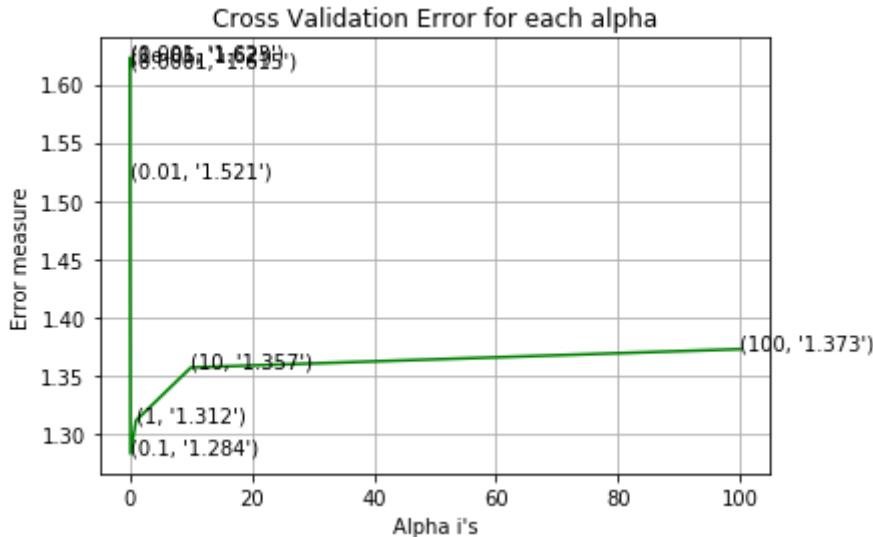


```
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge')
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")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

```
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
```

for C = 1e-05
Log Loss : 1.6201596751684968
for C = 0.0001
Log Loss : 1.6152389706375527
for C = 0.001
Log Loss : 1.6228005035878061
for C = 0.01
Log Loss : 1.5214373458263075
for C = 0.1
Log Loss : 1.28350640392116
for C = 1
Log Loss : 1.3118620939537922
for C = 10
Log Loss : 1.3572901452347381
for C = 100
Log Loss : 1.3730420157439664



For values of best alpha = 0.1 The train log loss is: 0.9736046834943349
For values of best alpha = 0.1 The cross validation log loss is: 1.28350640392116
For values of best alpha = 0.1 The test log loss is: 1.3024503489897257

In [89]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



```
# find more about CalibratedClassifierCV here at http://scikit-Learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# 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 = [.010 * x for x in range(1, 10)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
    clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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



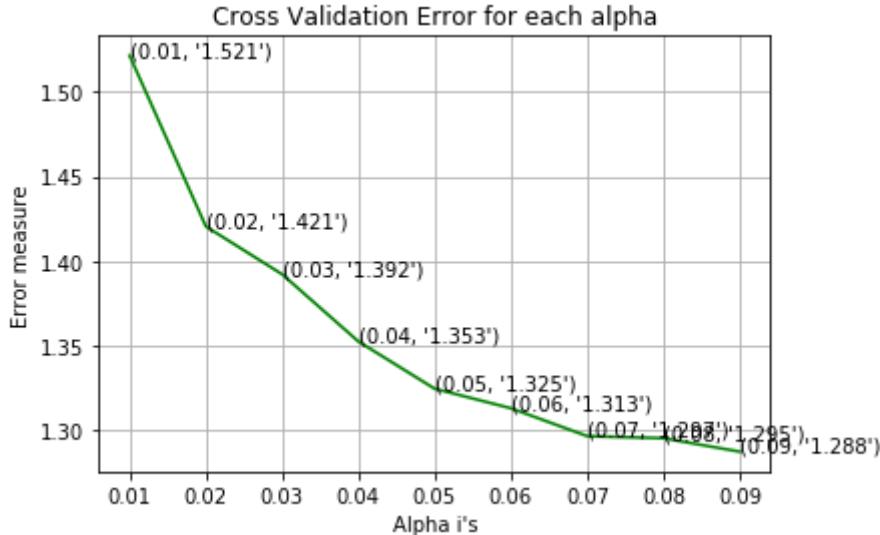
```
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")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

```

sig_clf.fit(train_x_onehotCoding, train_y)

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

for C = 0.01
Log Loss : 1.5214373458263075
for C = 0.02
Log Loss : 1.4205465386825573
for C = 0.03
Log Loss : 1.392306986398047
for C = 0.04
Log Loss : 1.3526296965024613
for C = 0.05
Log Loss : 1.3247111208586866
for C = 0.06
Log Loss : 1.313172991667708
for C = 0.07
Log Loss : 1.296723303595621
for C = 0.08
Log Loss : 1.295484612995185
for C = 0.09
Log Loss : 1.287544457801556



For values of best alpha = 0.09 The train log loss is: 0.9997779473893065
For values of best alpha = 0.09 The cross validation log loss is: 1.287544457801556
For values of best alpha = 0.09 The test log loss is: 1.3135819969334754

Testing model with best hyper parameters

In [90]:

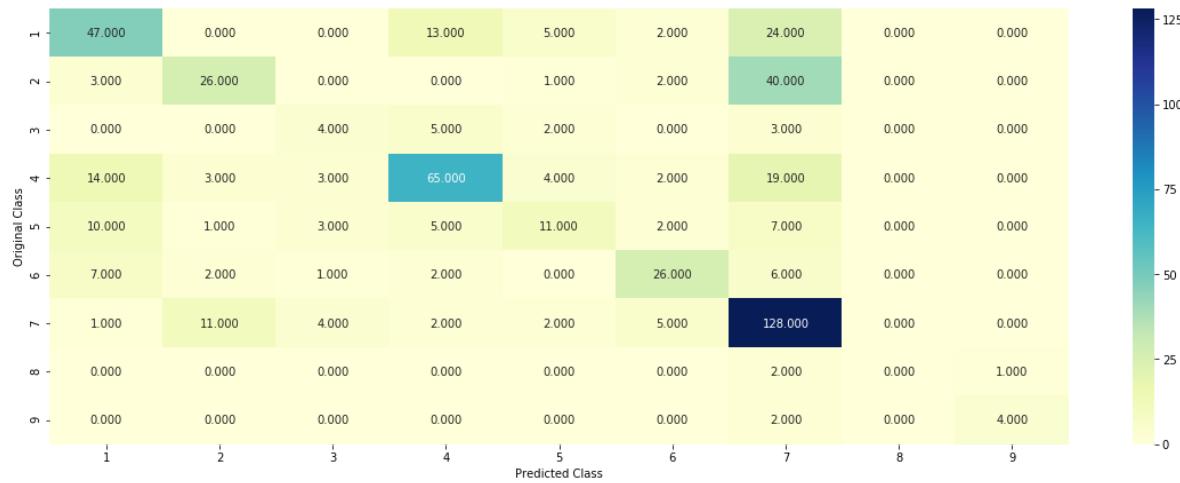
```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video Link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM
# -----
```

`clf = SVC(C=alpha[best_alpha],kernel='Linear',probability=True, class_weight='balanced')`
`clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42, class_weight='balanced')`

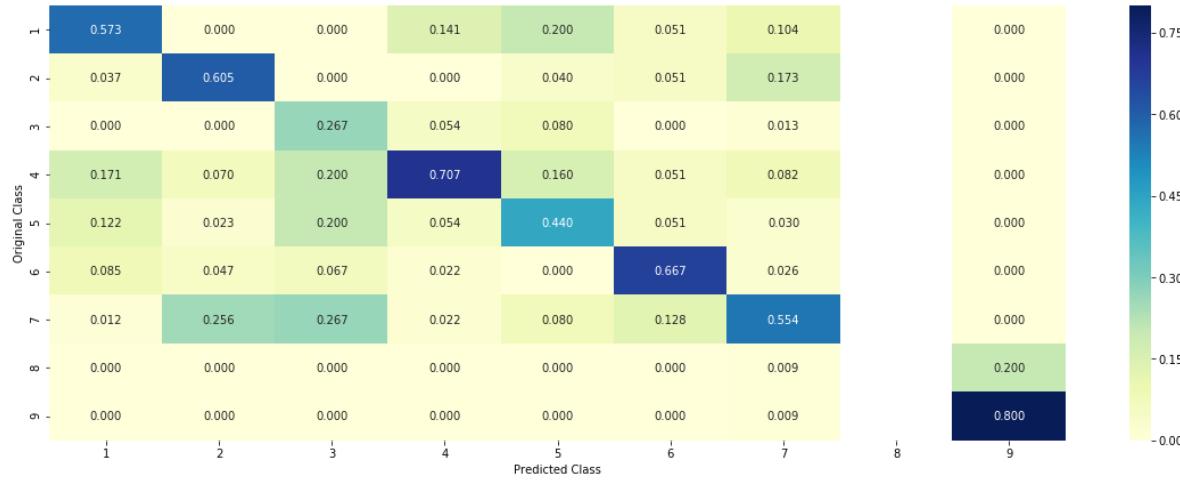
Log loss : 1.287544457801556

Number of mis-classified points : 0.41541353383458646

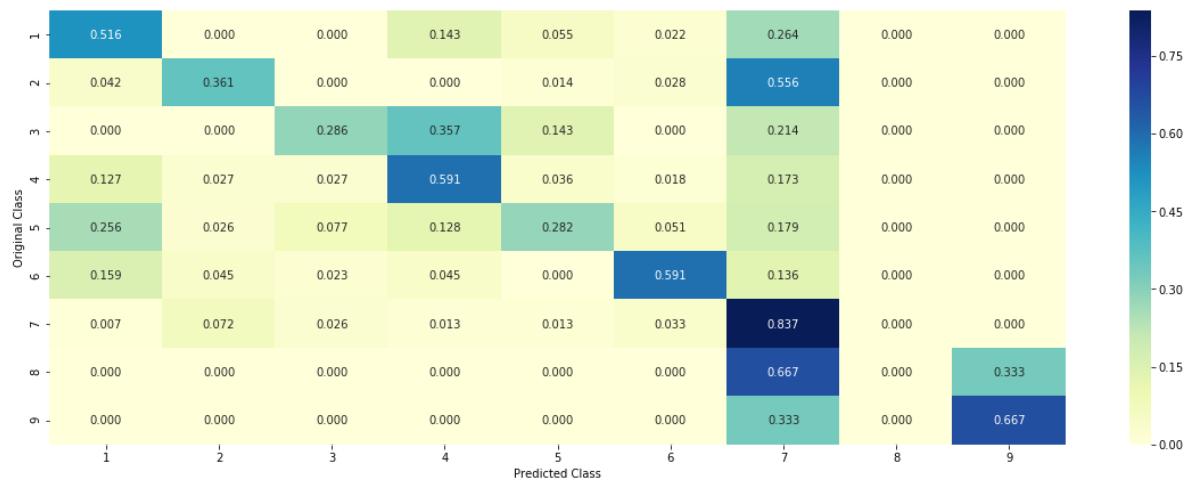
----- Confusion matrix -----



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



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



In [117]:

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

# Names of models
names =['LR With Class Balancing','LR Without Class Balancing','Linear SVM balanced']

# Training Loss
train_loss = [0.891891, 0.81582820,0.99977]

# Cross Validation Loss
cv_loss = [1.316569, 1.2001679,1.283506 ]
# Test Loss
test_loss = [1.28684,1.210664,1.31358 ]

log_loss=[1.316569 ,0.41165,1.28754445 ]

alpha=[100, 0.1 , 0.09 ]

# Percentage Misclassified points
misclassified= [0.43233,0.38533834,0.4154135]

numbering = [1,2,3]

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
s	Test_loss	log_loss	Misclassified(%)	
1	LR With Class Balancing	100	0.891891	1.3165
2	LR Without Class Balancing	0.1	0.8158282	1.20016
3	Linear SVM balanced	0.09	0.99977	1.2835

vectorising via TF-IDF(n-gram (1,4)) with taken all words into consideration from gene, variation and text

features

In [91]:

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

In [92]:

```
len(corpus)
```

Out[92]:

9963

In [93]:

```
text1 = TfidfVectorizer(ngram_range=(1, 4))
text2 = text1.fit_transform(corpus)
text1_features = text1.get_feature_names()

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

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

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

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

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

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

In [94]:

```
# merging gene, variance and text features

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

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

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

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

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

In [95]:

```
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, 1904564)
2)
(number of data points * number of features) in test data = (665, 19045642)
(number of data points * number of features) in cross validation data = (53
2, 19045642)
```

Logistic Regression

With Class balancing

Hyper parameter tuning

In [96]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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/geom
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_
    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 estimat
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```

sig_clf.fit(train_x_onehotCoding, train_y)

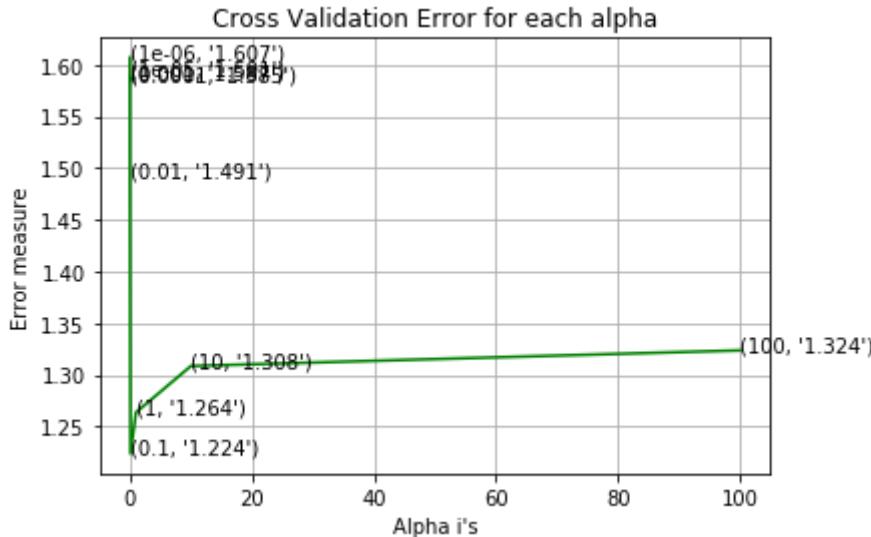
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y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

```

```

for alpha = 1e-06
Log Loss : 1.6067758934901166
for alpha = 1e-05
Log Loss : 1.5911466221437751
for alpha = 0.0001
Log Loss : 1.5845514389138287
for alpha = 0.001
Log Loss : 1.5872330736047235
for alpha = 0.01
Log Loss : 1.4909561966949971
for alpha = 0.1
Log Loss : 1.2236477292063392
for alpha = 1
Log Loss : 1.2635131308686687
for alpha = 10
Log Loss : 1.30840833790557
for alpha = 100
Log Loss : 1.3235352222075327

```



```

For values of best alpha = 0.1 The train log loss is: 0.7244437263724932
For values of best alpha = 0.1 The cross validation log loss is: 1.22364772
92063392
For values of best alpha = 0.1 The test log loss is: 1.2429892277882926

```

In [97]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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/geom
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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 = [.01 * x for x in range(1, 20)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_
        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 estimat
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

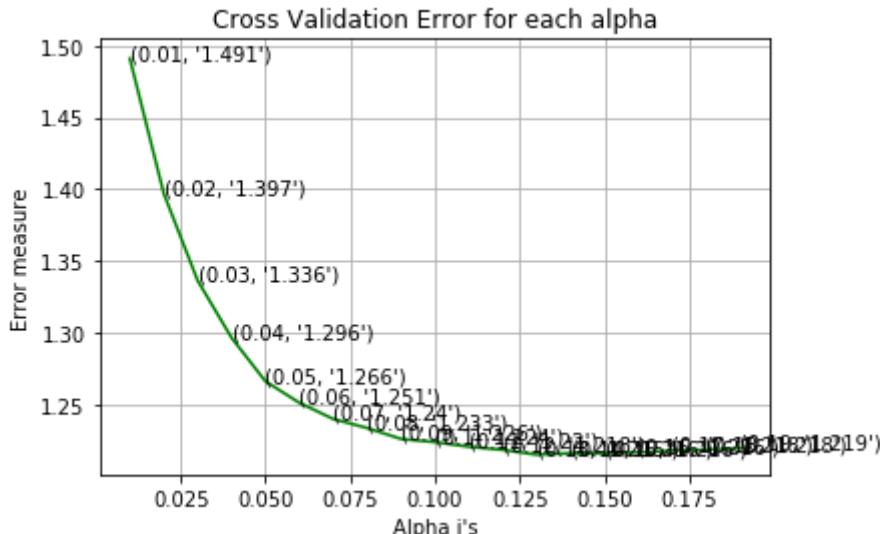
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
```

```
for alpha = 0.01
Log Loss : 1.4909561966949971
for alpha = 0.02
Log Loss : 1.3969281805373277
for alpha = 0.03
Log Loss : 1.3362211291025183
for alpha = 0.04
Log Loss : 1.2963728252226645
for alpha = 0.05
Log Loss : 1.2660294505156182
for alpha = 0.06
Log Loss : 1.2510563870784277
for alpha = 0.07
Log Loss : 1.2397551240854303
for alpha = 0.08
Log Loss : 1.2334579076964431
for alpha = 0.09
Log Loss : 1.225855616969996
for alpha = 0.1
Log Loss : 1.2236477292063392
for alpha = 0.11
Log Loss : 1.220426562581412
for alpha = 0.12
Log Loss : 1.2182477432967538
for alpha = 0.13
Log Loss : 1.2148922637927104
for alpha = 0.14
Log Loss : 1.215922372973232
for alpha = 0.15
Log Loss : 1.2159160159796198
for alpha = 0.16
Log Loss : 1.2164306226532715
for alpha = 0.17
Log Loss : 1.217655024249913
for alpha = 0.18
Log Loss : 1.2183711359629399
for alpha = 0.19
Log Loss : 1.219272506813675
```



For values of best alpha = 0.13 The train log loss is: 0.683339758424691

For values of best alpha = 0.13 The cross validation log loss is: 1.2148922637927104

For values of best alpha = 0.13 The test log loss is: 1.2337167131146358

In [98]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

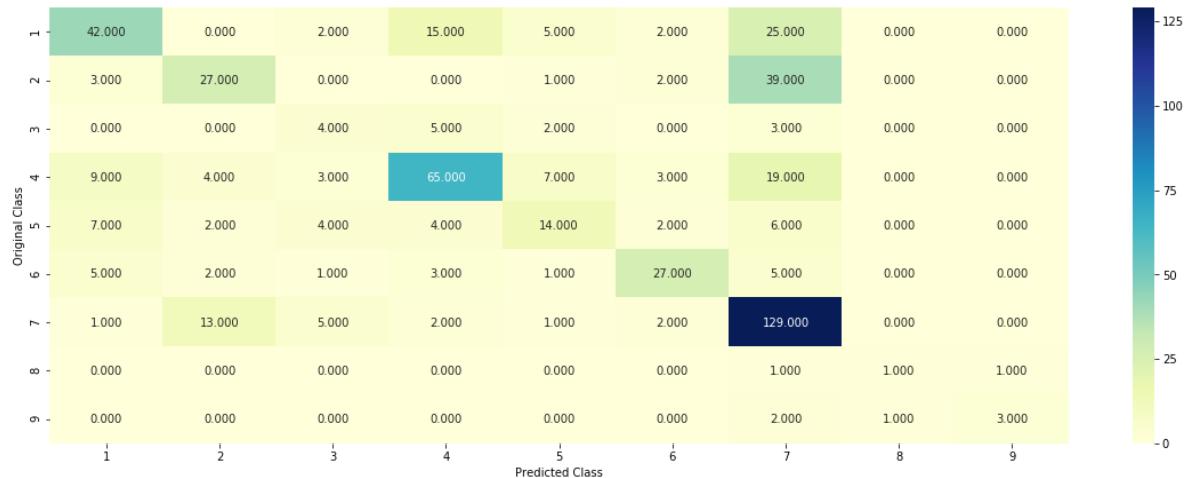
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

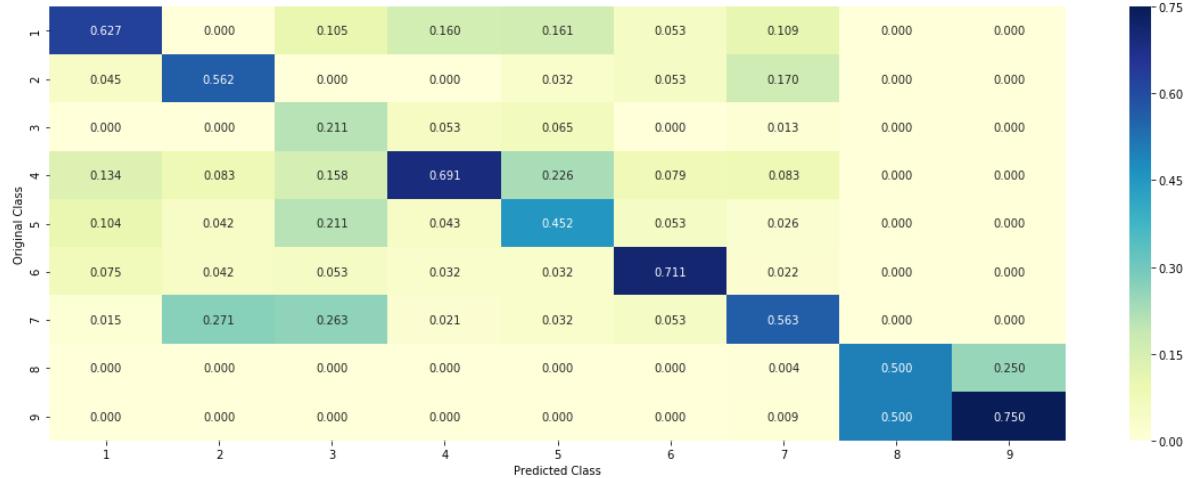
Log loss : 1.2148922637927104

Number of mis-classified points : 0.41353383458646614

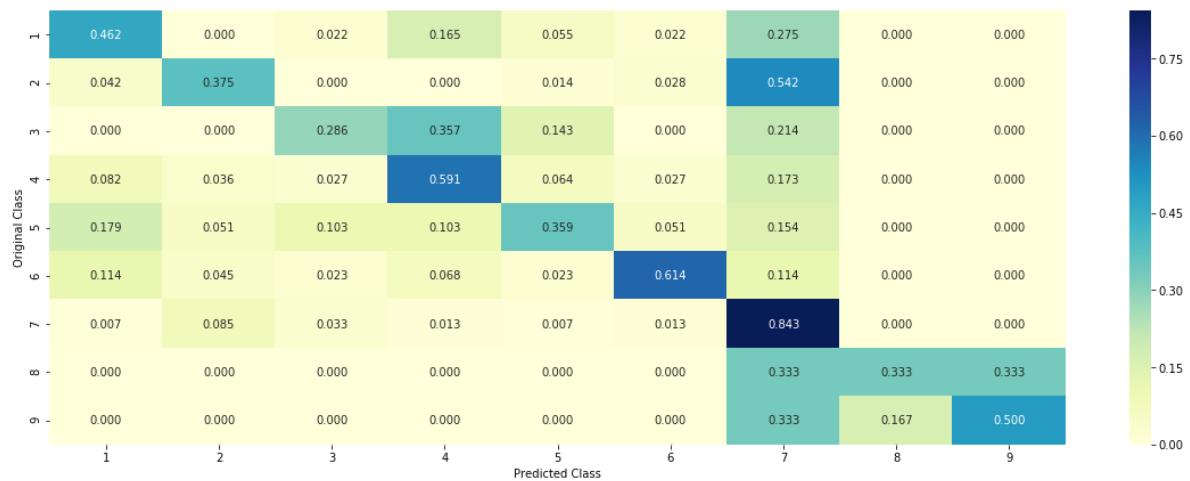
----- Confusion matrix -----



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



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



Without Class balancing

Hyper parameter tuning

In [99]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

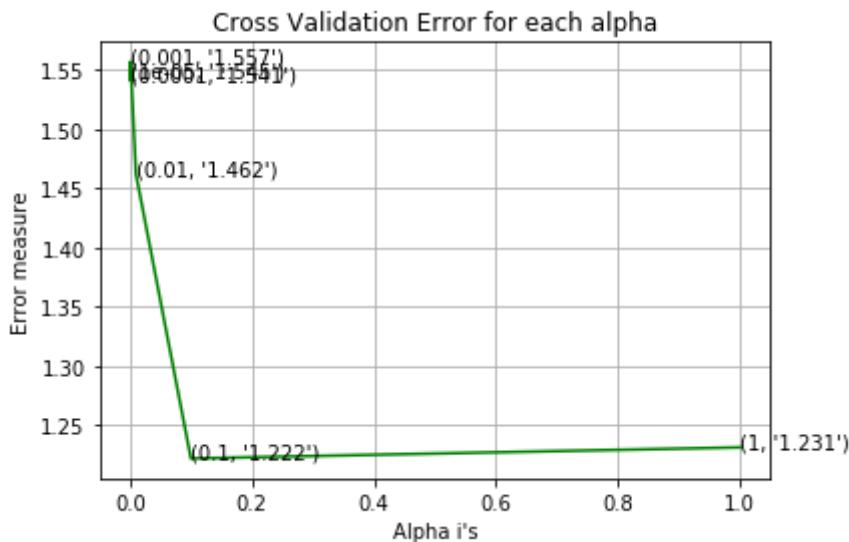
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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



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

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

```
for alpha = 1e-05
Log Loss : 1.5451885708264728
for alpha = 0.0001
Log Loss : 1.5413316076742016
for alpha = 0.001
Log Loss : 1.5569593730890259
for alpha = 0.01
Log Loss : 1.461980066650048
for alpha = 0.1
Log Loss : 1.2218897099115962
for alpha = 1
Log Loss : 1.2312559622227899
```



```
For values of best alpha =  0.1 The train log loss is: 0.7071762467421758
For values of best alpha =  0.1 The cross validation log loss is: 1.22188970
99115962
For values of best alpha =  0.1 The test log loss is: 1.2328727466811948
```

Testing model with best hyper parameters

In [100]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

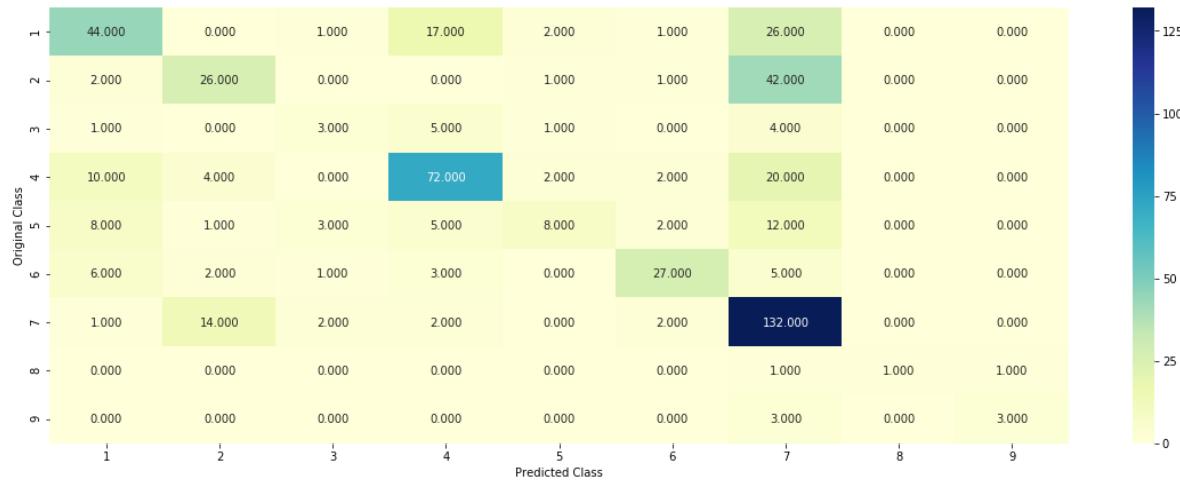
#-----
# video link:
#-----
```

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, c

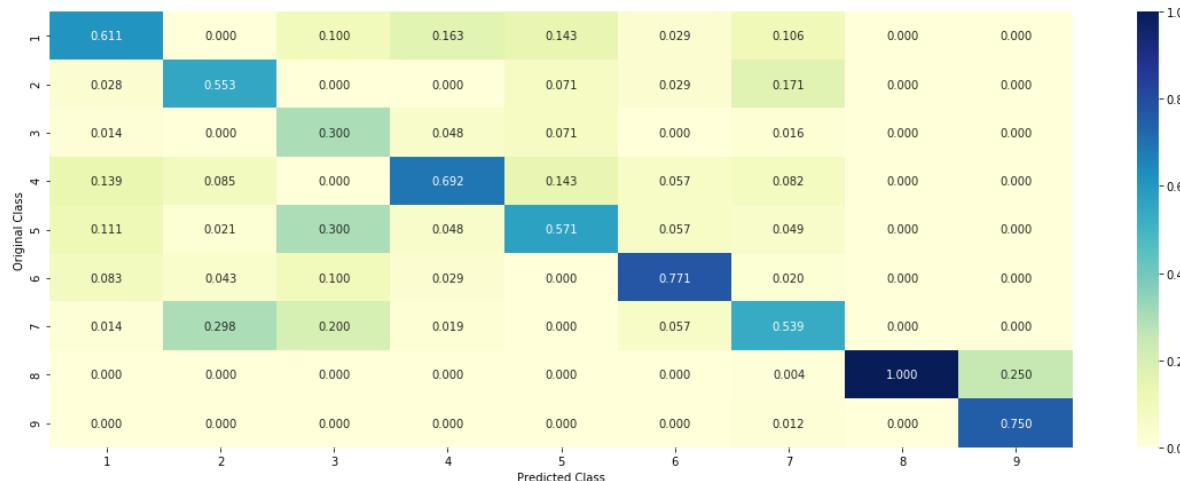
Log loss : 1.2218897099115962

Number of mis-classified points : 0.40601503759398494

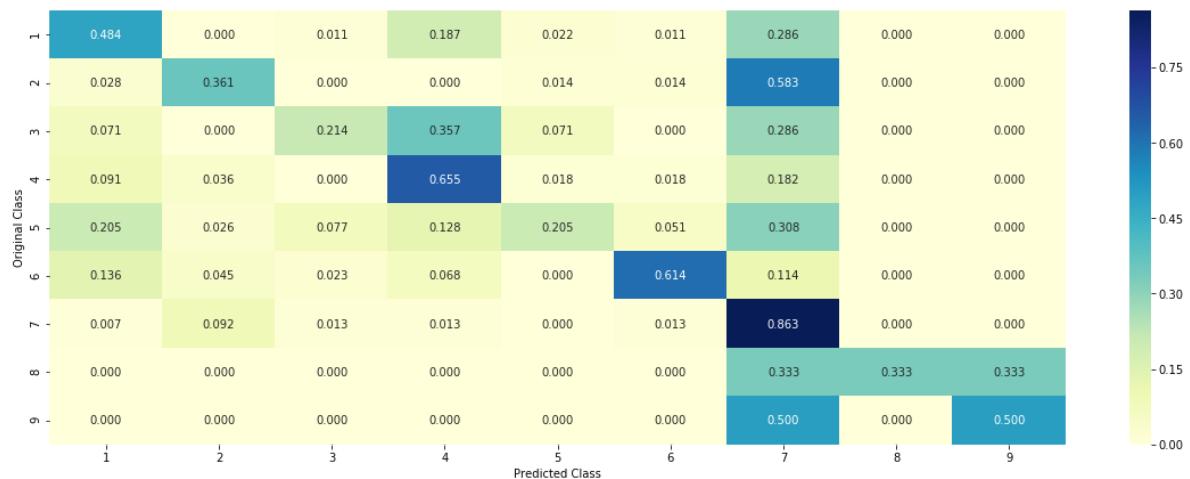
----- Confusion matrix -----



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



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



Linear Support Vector Machines

Hyper parameter tuning

In [101]:

```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM.ipynb
# -----
```



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



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

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



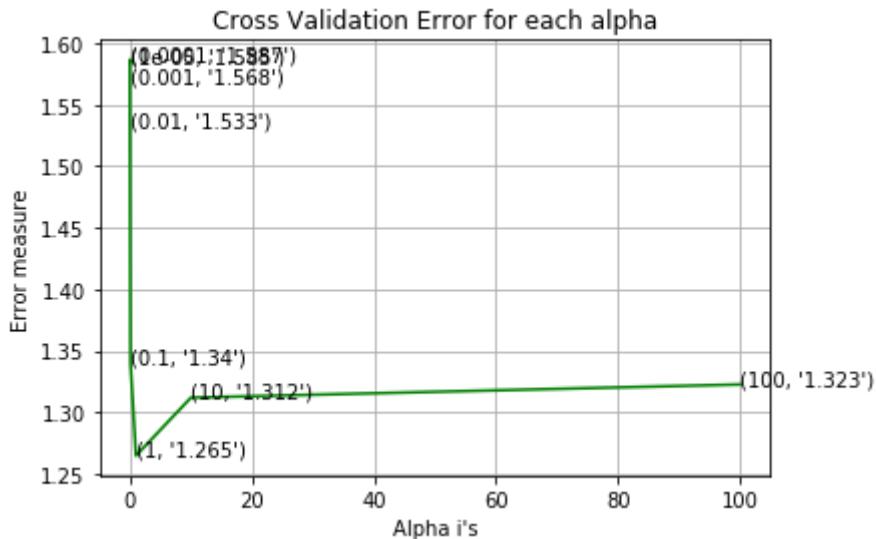
```
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge')
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")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

```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
```

for C = 1e-05
Log Loss : 1.584787495814163
for C = 0.0001
Log Loss : 1.586615652143332
for C = 0.001
Log Loss : 1.5680607951551926
for C = 0.01
Log Loss : 1.53295755981749
for C = 0.1
Log Loss : 1.340026197418921
for C = 1
Log Loss : 1.2646770843495638
for C = 10
Log Loss : 1.3118042946221815
for C = 100
Log Loss : 1.322524113320362



For values of best alpha = 1 The train log loss is: 0.6821351235377398
For values of best alpha = 1 The cross validation log loss is: 1.2646770843
495638
For values of best alpha = 1 The test log loss is: 1.2566752294712633

Testing model with best hyper parameters

In [105]:

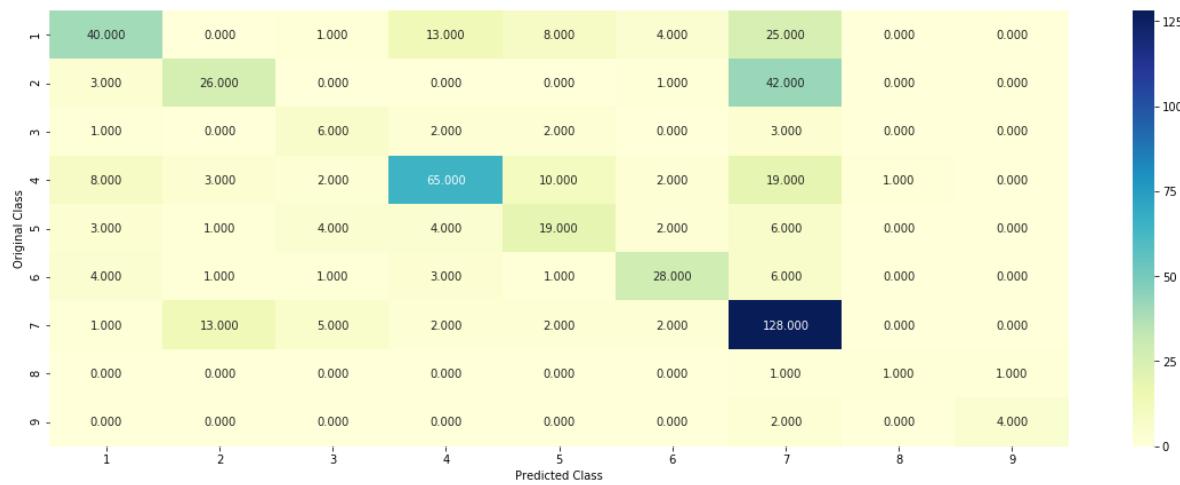
```
# read more about support vector machines with linear kernels here http://scikit-Learn.org/
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo')
# -----
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video Link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/mathematics/SVM
# -----
```

`clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')`
`clf = SGDClassifier(alpha=1, penalty='l2', loss='hinge', random_state=42, class_weight='balanced')`
`predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)`

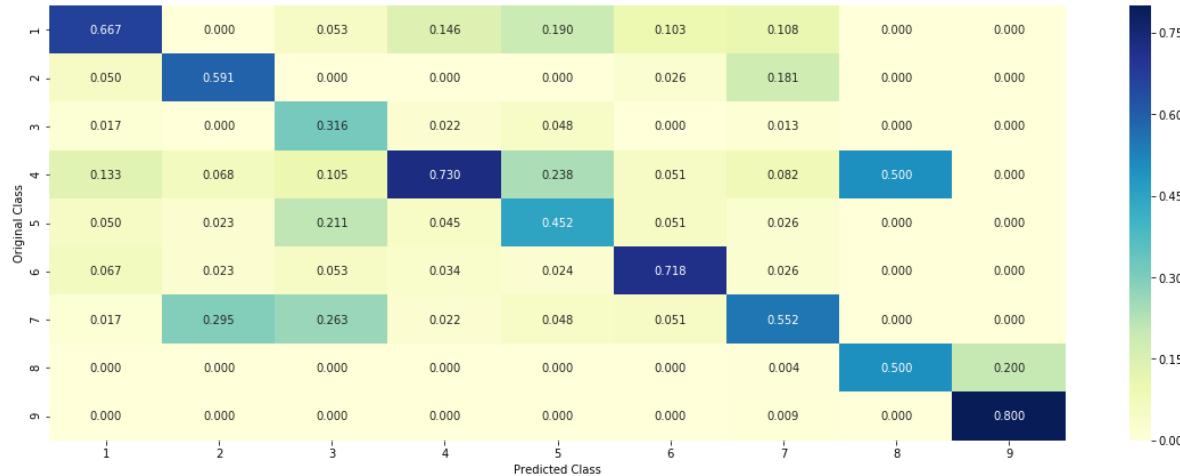
Log loss : 1.2646770843495638

Number of mis-classified points : 0.4041353383458647

----- Confusion matrix -----

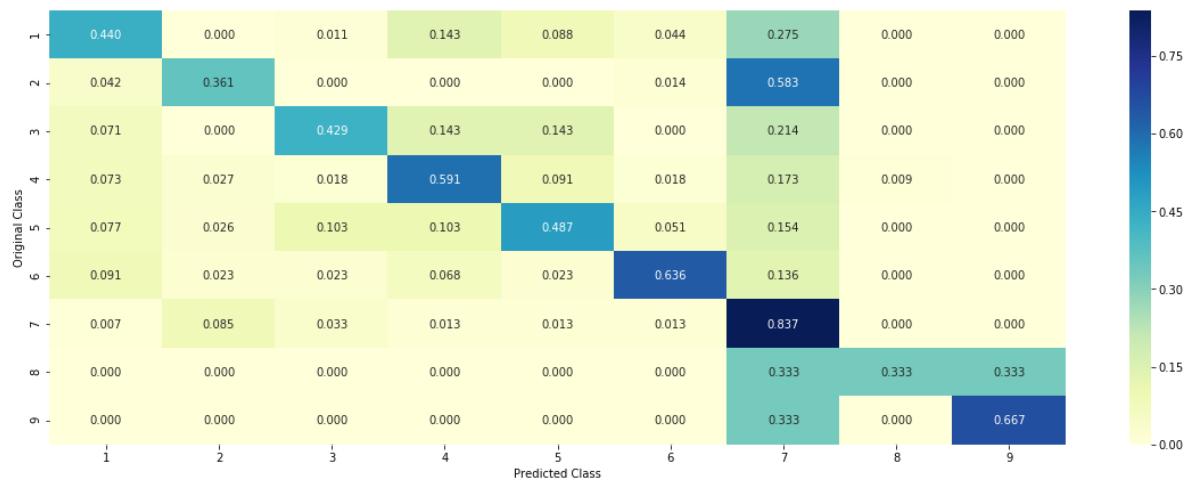


----- Precision matrix (Column Sum=1) -----



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

PersonalizedCancerDiagnosis



In [118]:

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

# Names of models
names =['LR With Class Balancing','LR Without Class Balancing','Linear SVM balanced']

# Training Loss
train_loss = [0.72444, 0.70717,0.6821]

# Cross Validation Loss
cv_loss = [1.2236, 1.2218897,1.264677 ]
# Test loss
test_loss = [ 1.2429892,1.232872,1.256675 ]

log_loss=[1.214892 ,1.221889,1.26467]

alpha=[0.13, 0.1 , 0.09 ]

# Percentage Misclassified points
misclassified =[0.4135,0.40601,0.4041]

numbering = [1,2,3]

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_loss
s	Test_loss	log_loss	Misclassified(%)	
1	LR With Class Balancing	0.13	0.72444	1.223
2	LR Without Class Balancing	0.1	0.70717	1.22188
3	Linear SVM balanced	0.09	0.6821	1.2646

TASK:4.3---->>>

Tf-IDF(n-gram(1,4)) with taken all words into consideration from gene, variation and text features + (Vectorise our text via spaCy NLP en_core_web_sm)

In [106]:

```
FEATUREtfidfw2v=pd.read_csv("FEATUREtfidfw2v.csv",encoding='latin-1')
FEATUREtfidfw2vTEST=pd.read_csv("FEATUREtfidfw2vTEST.csv",encoding='latin-1')
FEATUREtfidfw2vCV=pd.read_csv("FEATUREtfidfw2vCV.csv",encoding='latin-1')
```

In [107]:

```
df1 = FEATUREtfidfw2v.drop(['q1_feats_m','ID','Gene','Variation','Class','TEXT'],axis=1)
df2 = FEATUREtfidfw2vTEST.drop(['q1_feats_m','ID','Gene','Variation','Class','TEXT'],axis=1)
df3 = FEATUREtfidfw2vCV.drop(['q1_feats_m','ID','Gene','Variation','Class','TEXT'],axis=1)
```

In [108]:

```
train_x_onehotCoding = hstack((train_x_onehotCoding , df1)).tocsr()

test_x_onehotCoding = hstack((test_x_onehotCoding, df2)).tocsr()

cv_x_onehotCoding = hstack((cv_x_onehotCoding, df3)).tocsr()

cv_x_onehotCoding = normalize(cv_x_onehotCoding, axis=0)
test_x_onehotCoding = normalize(test_x_onehotCoding, axis=0)
train_x_onehotCoding = normalize(train_x_onehotCoding, axis=0)
```

In [109]:

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

One hot encoding features :
 (number of data points * number of features) in train data = (2124, 1904573
 9)
 (number of data points * number of features) in test data = (665, 19045739)
 (number of data points * number of features) in cross validation data = (53
 2, 19045739)

Logistic Regression

With Class balancing

Hyper parameter tuning

In [110]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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/geom
#-----


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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(-3, 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_
    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 estimat
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

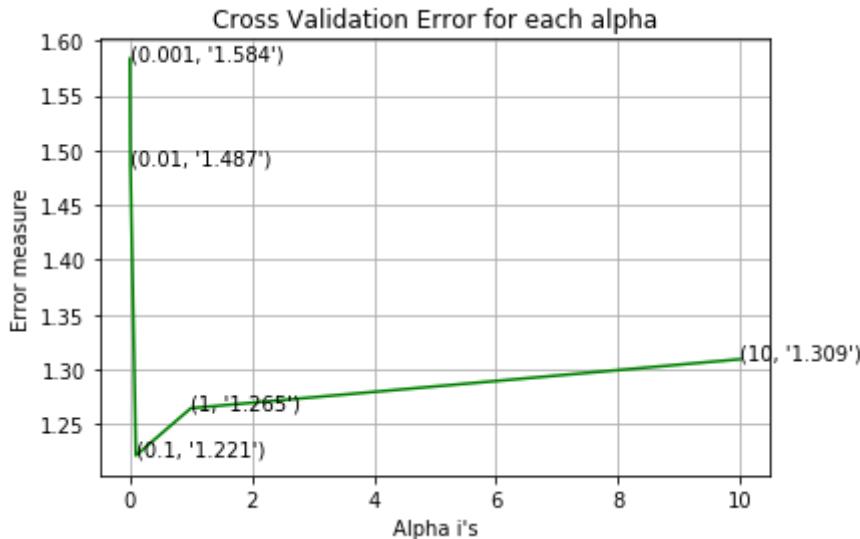
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
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_
```

```
<   >
for alpha = 0.001
Log Loss : 1.5836700191438835
for alpha = 0.01
Log Loss : 1.486870871673939
for alpha = 0.1
Log Loss : 1.2214635714129198
for alpha = 1
Log Loss : 1.264615970571102
for alpha = 10
Log Loss : 1.30934349807457
```



```
For values of best alpha =  0.1 The train log loss is: 0.7236393527176883
For values of best alpha =  0.1 The cross validation log loss is: 1.22146357
14129198
For values of best alpha =  0.1 The test log loss is: 1.2436389470463154
```

Testing model with best hyper parameters

In [111]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

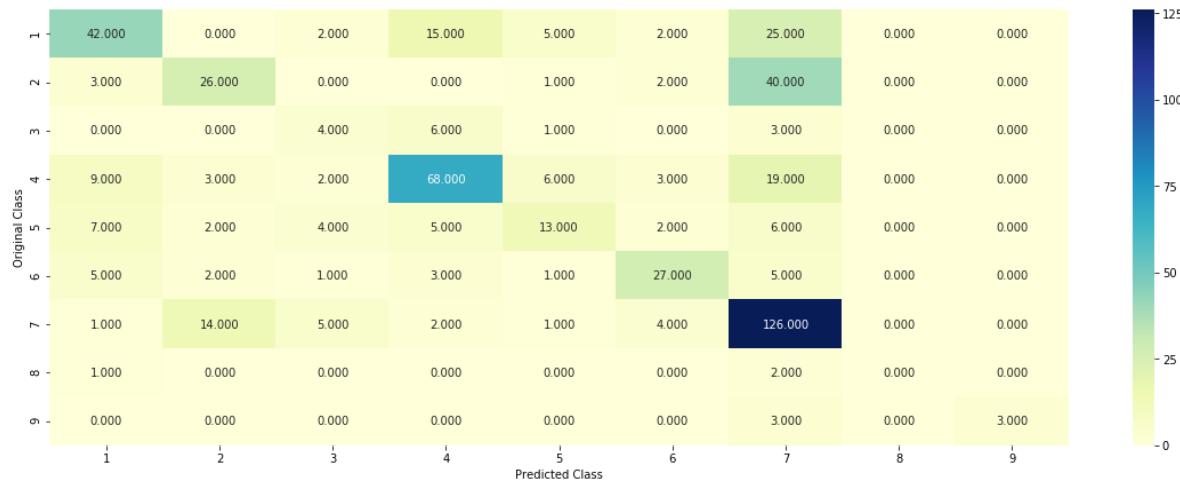
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/geome
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, c
```

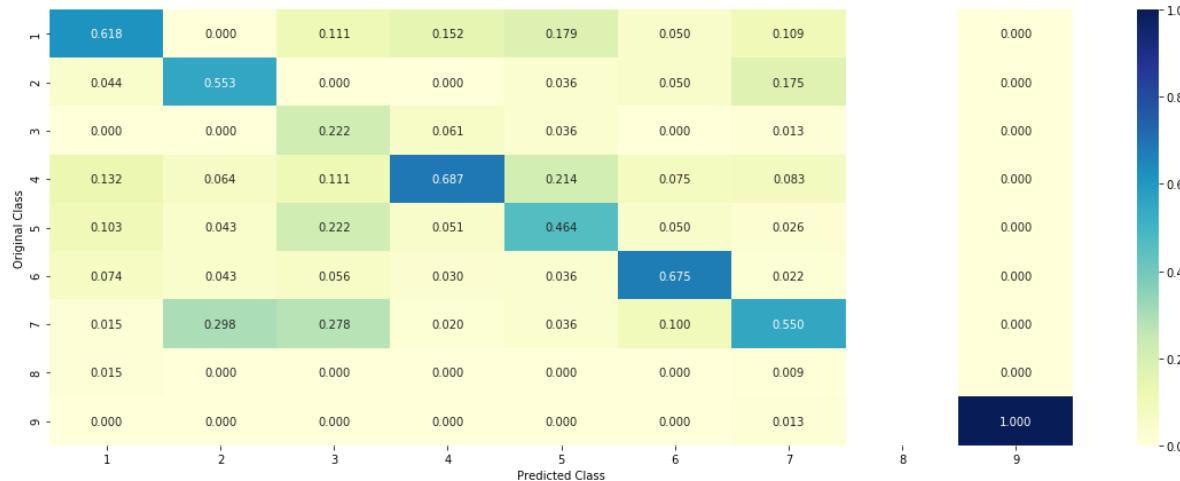
Log loss : 1.2214635714129198

Number of mis-classified points : 0.4191729323308271

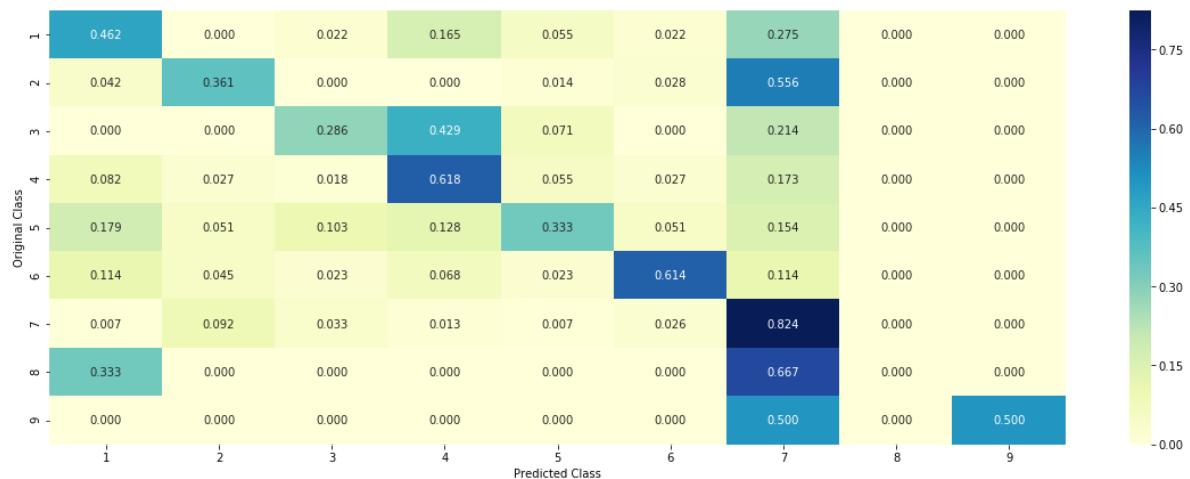
----- Confusion matrix -----



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



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



Logistic Regression

Without Class balancing

Hyper parameter tuning

In [112]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

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



```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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(-3, 2)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

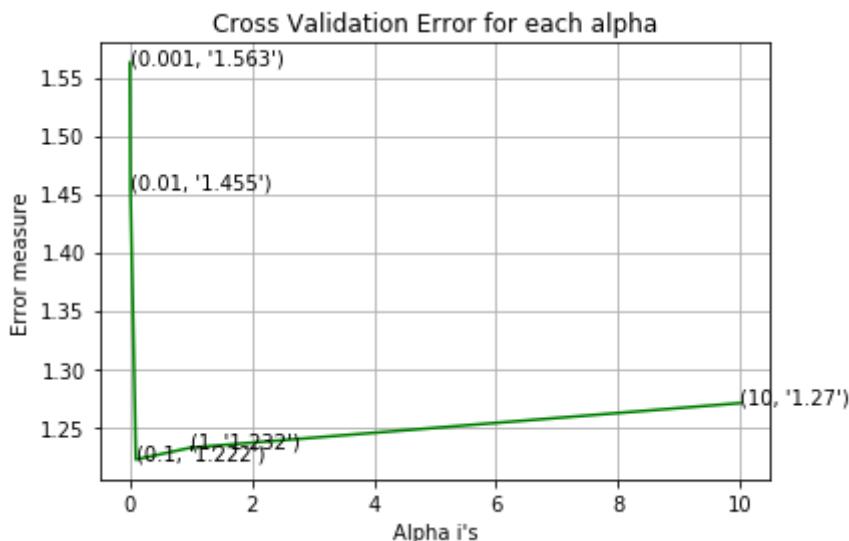
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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



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

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

```
for alpha = 0.001
Log Loss : 1.5626148249681933
for alpha = 0.01
Log Loss : 1.4552337116303742
for alpha = 0.1
Log Loss : 1.2220511179349027
for alpha = 1
Log Loss : 1.2322105743032936
for alpha = 10
Log Loss : 1.2704423133015814
```



```
For values of best alpha = 0.1 The train log loss is: 0.7077004612434604
For values of best alpha = 0.1 The cross validation log loss is: 1.22205111
79349027
For values of best alpha = 0.1 The test log loss is: 1.2286362725378064
```

Testing model with best hyper parameters

In [113]:

```
# read more about SGDClassifier() at http://scikit-Learn.org/stable/modules/generated/sklear
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='L2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal'
# 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 Desce
# predict(X) Predict class labels for samples in X.

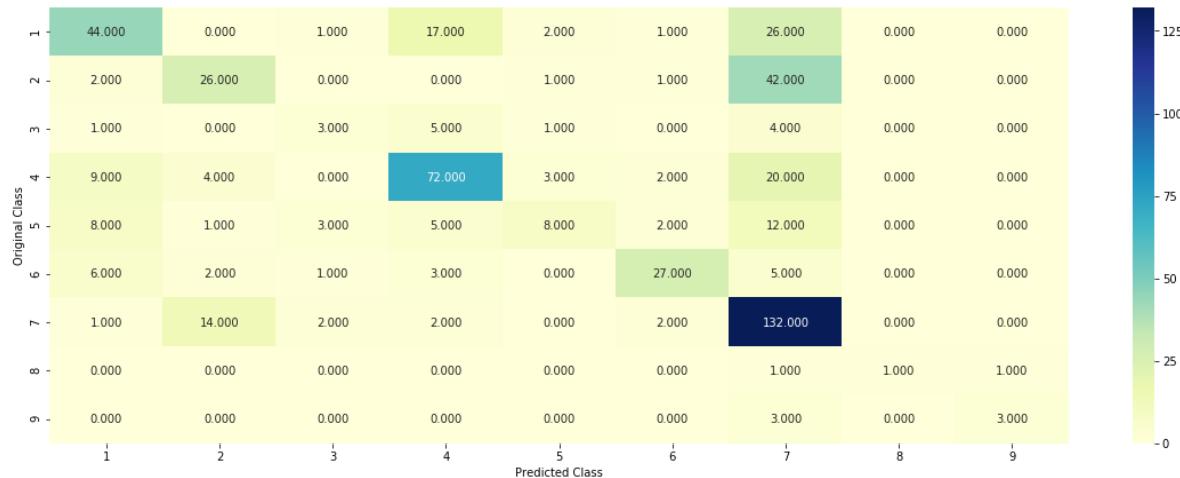
#-----
# video link:
#-----
```

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, c

Log loss : 1.2220511179349027

Number of mis-classified points : 0.40601503759398494

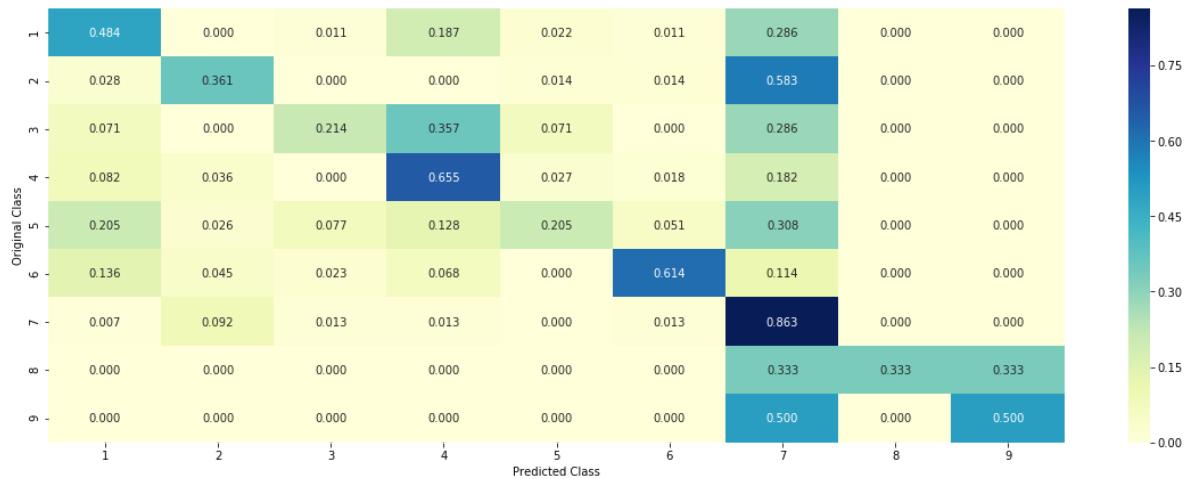
----- Confusion matrix -----



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



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



Linear Support Vector Machines

Hyper parameter tuning

In [121]:

```
train_x_onehotCoding
```

Out[121]:

```
<2124x19045739 sparse matrix of type '<class 'numpy.float64'>'  
with 65243888 stored elements in Compressed Sparse Column format>
```

In [124]:

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

# Names of models
names =['LR With Class Balancing','LR Without Class imBalancing']

# Training Loss
train_loss = [0.72444, 0.70770]

# Cross Validation Loss
cv_loss = [1.2214, 1.2220]
# Test Loss
test_loss = [ 1.24363,1.22863]

log_loss=[1.22146 ,1.22205 ]

alpha=[0.1, 0.1]

# Percentage Misclassified points
misclassified =[0.4191,0.40601]

numbering = [1,2]

# Initializing prettytable
ptable = PrettyTable()

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

# Printing the Table
print(ptable)
```

S.NO.	MODEL	hyperparameter	Train_loss	CV_lo
	Test_loss	log_loss	Misclassified(%)	
14	LR With Class Balancing	0.1	0.72444	1.22
2	LR Without Class imBalancing	0.1	0.7077	1.22

-----STEPS FOLLOWED-----

1--->>> Exploratory Data Analysis.

2--->>> Reading Gene and Variation Data.

3--->>> Preprocessing of text stop_words nlp.

4--->>> merging both gene_variations and text data based on ID.

5--->>> Test, Train and Cross Validation Split Split the dataset randomly into three parts train, cross validation and test with 64%, 16%, 20% of data respectively

6--->>> Prediction using a 'Random' Model

7--->>> Univariate Analysis Gene, Variation AND TEXT FeatureS What type of featureS THESE ARE How many categories are there and How they are distributed.

8--->>> How to featurize this Gene feature AND Variation feature:- a: One hot Encoding. b: Response coding.

9--->>> Test each gene How good is this gene feature in predicting y_i via l.r.

10--->>> Try out various different algorithms with different feature engineering on different vectorised data.

11--->>> DONE 4 TASKS AS FOLLOWS:-

1--->>> Replace CountVectorizer() by TfidfVectorizer() in all the one hot encoding section of gene , variation and text features.

2--->>> Replace TfidfVectorizer() by TfidfVectorizer(max_features=1000) in the one hot encoding section of text feature to get top 1000 words based of tf-idf values.

3--->>> Replace CountVectorizer() by CountVectorizer(ngram_range=(1,2)) in all the one hot encoding section of gene , variation and text features to get unigrams and bigrams .

Then Apply Logistic Regression.

4--->>> Trying different feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less as possible i.e 1 or near to 1.

NOTE: HERE WE ARE TRYING DIFFERENT ENGINEERING HACKS:

1--> We also vectorise via TF-IDF with taken all words into consideration from gene, variation and text feature.

2--> We also vectorise our text via spaCy NLP en_core_web_sm and concat it with all above feature to improve our results.

3--> Also done all above task in 4th task with tf-idf(ngram(1,4))

CONCLUSIONS:-

AS WE APPLY VARIOUS TECHNIQUES ,FEATURE ENGINEERING AND ALGORITHMS OUR LOG LOSS TENDS TO NEAR 1 IN APPLYING VARIOUS TRICKS(HACKS).

WE CAN IMPROVE OUR RESULTS WITH DOMAIN FEATURE ENGINEERING MADE BY DOMAIN EXPERTS.

WE CAN ALSO APPLY DEEP LEARNING TO AUTOMATIC FEATURIZATION .

HERE WE ARE HAVING LOW COMPUTATION POWER SYSTEM SO WE CANT APPLY VARIOUS OTHER POWERFUL TECHNIQUES LIKE DEEP LEARNING AND OTHER POWERFUL NLP TECHNIQUES WHICH TAKES LOTS OF COMPUTATION VIA WHICH WE CAN IMPROVE OUR SOLUTIONS AND PERFORMANCE MATRIX.

In []: